

TECHNISCHE UNIVERSITÄT BERLIN

DOCTORAL THESIS

---

# From Edge to Cloud: Engineering the Industrial Compute Continuum

A Cloud-Native Approach to the Architecture, Implementation, and  
Management of Modern Industrial Systems

---

*Author:*  
Hai Dinh-Tuan

*Supervisors:*  
Prof. Dr. Axel KÜPPER

September 23, 2025



# Declaration of Authorship

I, Hai Dinh-Tuan, declare that this thesis, titled "From Edge to Cloud: Engineering the Industrial Compute Continuum" and the work presented in it are my own. I confirm that:

- This work was done mainly while in candidature for a research degree at the Technische Universität Berlin.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.
- I have made use of generative AI tools. Specifically, OpenAI's GPT-4o was used to improve the language of my own expression and to assist in translating several short drafts originally written in my native language. Google's Gemini 2.5 Pro was used to help with special formats of tables in L<sup>A</sup>T<sub>E</sub>X.

Signed:

---

Date:

---



# Abstract

The Fourth Industrial Revolution (Industry 4.0) fundamentally challenges traditional manufacturing systems and their established design principles. These traditional ISA-95-based systems can no longer meet the contrasting demands of modern production environments: real-time responsiveness on the shop floor and large-scale analytical processing in the cloud. Although the Compute Continuum promises to combine the advantages of cloud and edge infrastructures, the current research landscape still lacks a concrete engineering framework for its implementation.

This research answers the central question: How can a unified cloud-native reference architecture be designed and implemented within the Compute Continuum to meet the changing demands of Industry 4.0? To achieve this, the work follows a multi-stage *Design Science Research* methodology. The process begins with a *Systematic Literature Review* to explore the research landscape and identify key gaps. It then uses *Attribute-Driven Design (ADD 3.0)* to develop a novel cloud-native reference architecture. Finally, the proposed framework are validated through several quantitative and qualitative evaluations of its implementation and core management components.

The main contributions of this dissertation are delivered as a multi-phase engineering framework. This work first establishes a comprehensive *Taxonomy of the Compute Continuum*, which systematically maps the state-of-the-art and identifies the research gaps that motivate the subsequent research. Then, this work proposes the *Cloud-native Reference Architecture for the Compute Continuum in Industry 4.0* (CRACI), which is a decoupled, event-driven architecture designed to address the limitations inherent in traditional industrial designs. Further contributions include: a quantitative analysis of the performance and resource trade-offs between key development paradigms (e.g., various microservice frameworks) and deployment technologies (containers vs. unikernels); a novel, message-based framework for the live migration of stateful services (MS2M); and advanced frameworks for intelligent, decentralized resource management (SAGA and DREAMS) that make use of application-level semantics.

The key argument of this dissertation is that engineering for the Industrial Compute Continuum requires a cloud-native, architecture-driven approach that prioritizes application-awareness and decentralized management. The findings of this work suggest that such an architecture is not only feasible, but also essential for delivering the scalability and resilience that modern industrial systems need. The architecture and accompanying frameworks in this research are therefore offered as a practical foundation for building the next generation of industrial systems, advancing the vision of Industry 4.0 and beyond.



# Zusammenfassung

Die vierte industrielle Revolution (Industrie 4.0) stellt traditionelle Produktionssysteme und ihre bewährten Konstruktionsprinzipien grundlegend in Frage. Klassische, auf ISA-95 basierende Architekturen können den Anforderungen moderner Fertigungsumgebungen mit Echtzeitreaktionen in der Produktion und großskaliger Datenanalyse in der Cloud nicht mehr gerecht werden. Obwohl das Compute Continuum die Verbindung von Cloud- und Edge-Infrastrukturen verspricht, fehlt bisher ein konkreter ingenieurwissenschaftlicher Rahmen für seine Umsetzung.

Diese Dissertation untersucht, wie eine einheitliche, cloud-native Referenzarchitektur im Compute Continuum entworfen und implementiert werden kann, um den Anforderungen der Industrie 4.0 zu entsprechen. Die Arbeit folgt einer mehrstufigen *Design-Science-Research-Methodik* (DSR). Sie beginnt mit einer *Systematic Literature Review* (SLR) zur Analyse des Forschungsstands und zur Identifizierung zentraler Lücken. Anschließend wird mithilfe von *Attribute-Driven Design* (ADD 3.0) eine neue cloud-native Architektur entwickelt und durch quantitative und qualitative Evaluationen ihrer Implementierung und Management-Komponenten validiert.

Die Hauptbeiträge dieser Arbeit sind in einem strukturierten, mehrphasigen Rahmen zusammengefasst. Zunächst wird eine umfassende *Taxonomie des Compute Continuum* vorgestellt, die den Stand der Forschung systematisch erfasst und bestehende Lücken aufzeigt. Darauf aufbauend folgt die *Cloud-Native Reference Architecture for the Compute Continuum in Industry 4.0* (CRACI), eine modulare, ereignisgesteuerte Architektur, die die Einschränkungen klassischer industrieller Modelle überwindet. Weitere Beiträge umfassen eine Analyse von Leistungs- und Ressourcentrade-offs zwischen Entwicklungsparadigmen (z. B. Microservice-Frameworks) und Bereitstellungstechnologien (Container vs. Unikernels), das MS2M-Framework zur Migration zustandsbehafteter Dienste sowie die dezentralen Management-Frameworks SAGA und DREAMS, die Anwendungssemantik für eine intelligente Koordination nutzen.

Das zentrale Argument dieser Dissertation ist, dass die Entwicklung des Industrial Compute Continuum einen cloud-nativen, architekturorientierten Ansatz erfordert, der Anwendungsbewusstsein und dezentrale Verwaltung in den Mittelpunkt stellt. Die Ergebnisse zeigen, dass eine solche Architektur nicht nur realisierbar, sondern auch notwendig ist, um die Skalierbarkeit und Zuverlässigkeit moderner industrieller Systeme zu gewährleisten. Die vorgestellte Architektur und die zugehörigen Frameworks bieten eine praxisorientierte Grundlage für den Aufbau der nächsten Generation industrieller Systeme im Sinne von Industrie 4.0 und darüber hinaus.



## Acknowledgements

I am deeply grateful to everyone who accompanied me on the path that led to this dissertation and contributed to my professional and personal growth.

Foremost, I extend my sincere gratitude to my supervisor, Prof. Axel Küpper, for creating a supportive and stimulating research environment that made this work possible. Your guidance, gently steering ideas toward maturity while allowing room for creativity, has been invaluable. Thank you, Axel, for laying the foundation for the research presented here.

I would also like to express my sincere thanks to Prof. Thomas Magedanz and Prof. Davide Taibi for agreeing to review this dissertation, and to Prof. Stefan Schmid for chairing the committee and supporting the final step of this journey. I am truly grateful for your time, interest, and valuable feedback.

I am deeply thankful to the entire SNET team for promoting a wonderful environment that nurtured my curiosity. Beyond the scientific discussions that broadened my horizons, I will always remember our shared lunches, coffee breaks, and offsite trips with great fondness.

My heartfelt thanks go to Dr. Sandro Rodriguez Garzon, Dr. Felix Beierle, and Prof. Peter Ruppel, who guided my first steps into research and encouraged me to pursue questions with rigor and creativity. The projects we worked on together shaped the early ideas that guided my doctoral journey.

My special thanks also go to Dr. Friedhelm Victor, Dr. Martin Westerkamp, Dr. Aikaterini Katsarou, Dr. Sebastian Göndör, Dr. Philip Raschke, Patrick Herbke, Sanjeet Raj Pandey, Dr. Tobias Eichinger, and Maria Mora Martinez for their friendship, conversations, and steady support. Your camaraderie made the challenges of doctoral work more manageable and the successes more meaningful. The memories of working alongside you will remain an unforgettable part of my life.

To the students I had the pleasure of working with—Van Dung Do, Jialun Jiang, Tien Hung Nguyen, Florian Six, Ksenia Legostay, Yong Wu, and Jianeng Fu—thank you for your collaboration and enthusiasm, which made our time together rewarding. I learned a great deal from you, and your fresh perspectives as well as dedication continually inspired me to grow as a researcher and mentor.

To my family, I owe deep gratitude. Living and working far from home has never been easy, but your faith in me has always been a source of strength and comfort. Those years away have taught me some of the most profound lessons about what family and belonging truly mean.

To my wife, Thi Thuy Van Quach, thank you for sharing this journey of discovery in life and research. Your strength, patience, and warmth have been a constant source of inspiration. Thank you for standing by my side through both the ups and downs, and for the immense care and love you have shown me, especially during times of exhaustion and stress.

Thank you all for being part of this chapter of my life!



# Glossary

- ADD** Attribute-Driven Design: A software architecture method that bases design decisions on a system's Quality Attribute Requirements.
- BFT** Byzantine Fault Tolerance: A property that enables a system to function correctly despite arbitrary or malicious component failures.
- CC** Compute Continuum: A computing model that unifies a spectrum of resources, from centralized cloud to decentralized edge and IoT devices.
- CI/CD** Continuous Integration/Continuous Deployment: A set of practices for automating the build, test, and deployment phases of software development.
- CRACI** Cloud-native Reference Architecture for the Compute Continuum in Industry 4.0: The reference architecture developed in this dissertation for modern industrial systems.
- CPS** Cyber-Physical System: A system that integrates computation with physical processes using embedded systems.
- DREAMS** Decentralized Resource Allocation and Service Management across the Compute Continuum Using Service Affinity: A decentralized, fault-tolerant framework proposed in this dissertation for resource management across the continuum, using autonomous agents.
- DSR** Design Science Research: A problem-solving research methodology focused on building and evaluating IT artifacts to solve practical problems.
- DT** Digital Twin: A digital representation of a physical asset or system used for monitoring and optimization.
- EDA** Event-Driven Architecture: An architectural pattern where services communicate asynchronously through the production and consumption of events, a core principle of CRACI.
- HMI** Human-Machine Interface: Interfaces allowing operators to interact with industrial systems.
- IIoT** Industrial Internet of Things: The application of IoT technologies in industrial settings.
- ISA-95** International standard defines a hierarchical model for integrating enterprise and control systems.
- IT** Information Technology: The enterprise-level computing systems focused on data and business processes.

- IoC** Inversion of Control: A design principle where a framework controls the program flow and calls custom code, as opposed to a library being called by the code.
- LDM** Local Domain Manager: A core component in the DREAMS framework, an autonomous agent responsible for managing resources and coordinating migrations within its local domain.
- MAIA** Microservices-based Architecture for Industrial Data Analytics: A microservice-based architecture developed to demonstrate the core principles of CRACI architecture in industrial analytics.
- MES** Manufacturing Execution System: Software managing and monitoring shop floor operations, corresponding to Level 3 of the ISA-95 pyramid.
- MS2M** Message-based Live Stateful Microservice Migration: A framework for live migration of stateful services through message replay, developed as part of this dissertation.
- OLAP** Online Analytical Processing: Systems optimized for complex analytical queries on multidimensional data.
- OLTP** Online Transaction Processing: Systems optimized for handling transactional workloads.
- OT** Operational Technology: The hardware and software that detects or causes a change through the direct monitoring and/or control of physical devices, processes, and events.
- PRISMA** Preferred Reporting Items for Systematic Reviews and Meta-Analyses: A guideline for reporting systematic literature reviews.
- RAMI 4.0** Reference Architecture Model for Industry 4.0: A structured, three-dimensional model for Industry 4.0 systems.
- SA** Service Affinity: A multi-dimensional metric proposed in this dissertation to quantify the "closeness" between microservices based on data, coupling, functional, operational, and privacy factors.
- SAGA** Service Affinity Graph-based Approach: A centralized framework proposed in this dissertation for service placement optimization.
- SCADA** Supervisory Control and Data Acquisition: Systems for real-time monitoring and control of industrial processes.
- SLO** Service Level Objective: Performance targets agreed upon between service providers and users.

# List of Figures

1.1	The DSR methodology process model adopted in this Dissertation [11]. . . . .	7
1.2	Dissertation Structure. . . . .	9
2.1	Fog computing as defined by the National Institute of Standards and Technology (NIST)[31]. . . . .	13
2.2	Multi-access Edge System Reference Architecture [33]. . . . .	14
3.1	Two-phase approach used in this study. . . . .	25
3.2	Number of Publications by Year, searched on Scopus using the query shown in Table 3.3. . . . .	27
3.3	PRISMA Flow Diagram (created for this study based on [12]). . . . .	29
4.1	Industry-specific Use Cases and Applications. . . . .	35
4.2	Engineering and Technical Solutions. . . . .	38
4.3	System Architectures and Designs. . . . .	57
5.1	Classification of Research in the Collected Publications. . . . .	63
5.2	Publication Trends over the Years. . . . .	65
5.3	Distribution of Terminologies used in the collected Publications. . . . .	66
5.4	The Driving Trends of Compute Continuum Research. . . . .	68
6.1	The ISA-95 Automation Pyramid showing the hierarchical structure from physical processes to enterprise systems [432]. . . . .	76
6.2	Reference Architecture Model Industrie 4.0 (RAMI 4.0) [435]. . . . .	78
6.3	FIWARE Smart Industry Reference Architecture [439, 440]. . . . .	79
6.4	NAMUR Open Architecture [441, 442]. . . . .	80
6.5	LASFA (LAsim Smart FActory) Architecture for Smart Manufacturing [444]. . . . .	82
6.6	Overview of the Stuttgart IT Architecture for Manufacturing [445]. . . . .	83
6.7	Cloud-native Reference Architecture for the Compute Continuum in Industry 4.0 (CRACI). . . . .	92
7.1	MAIA use case: Orchestration of service handover for mobile robotics. . . . .	108
7.2	A Concrete Implementation of the Reference Architecture: The MAIA Architecture. . . . .	109
7.3	Implementation of MAIA Architecture. . . . .	112
7.4	Comparison of Microservice Artifact Size: Raw Executable vs. Containerization. . . . .	114
7.5	Comparison of Build Time for Raw Executable vs. Containerization. . . . .	115
7.6	Impact of JVM Heap-Size Tuning on Container Memory Footprint. . . . .	116

7.7	End-to-end processing time, adapted from a previously published work [457]. . . . .	118
8.1	Overview of the AMR energy estimation pipeline. . . . .	135
8.2	Example of long latency for the first messages processed by the Lagom-based implementation, demonstrating the JVM warm-up effect [463]. . . . .	136
8.3	Comparison of latency performance for the microservices pipeline across frameworks [463]. . . . .	137
8.4	Docker image size comparison for the implemented services [463]. . . .	138
8.5	CPU consumption of the implemented services [463]. . . . .	139
8.6	Memory consumption of the implemented services [463]. . . . .	140
9.1	Architectural comparison showing a Linux container's reliance on a shared host kernel versus a unikernel's specialized, self-contained design running on a hypervisor (recreated from [486]). . . . .	144
9.2	Comparison of image size between container and unikernel. . . . .	147
9.3	Comparison of Time to Ready between container and unikernel. . . . .	148
9.4	Comparison of Throughput between container and unikernel in a Resource-rich Scenario. . . . .	148
9.5	I/O-Bound Throughput for the Go Application under Varying Memory Constraints (1 CPU). The Nanos unikernel consistently delivers an order-of-magnitude higher throughput. . . . .	149
9.6	Average Latency and Standard Deviation for the Go Application under Varying Memory Constraints (1 CPU). Error bars represent the standard deviation. Nanos demonstrates both lower average latency and lower jitter.	150
9.7	I/O-Bound Throughput for the Node.js Application under Varying Memory Constraints (1 CPU). A performance crossover occurs at 128 MB, where Docker performs better. . . . .	150
9.8	Average Latency and Standard Deviation for the Node.js Application (1CPU). Note the increase in latency and jitter for the unikernel at 128 MB and below. . . . .	151
9.9	CPU-Bound Throughput in a Resource-rich Scenario (8 CPU, 2048 MB). .	152
9.10	CPU-Bound Throughput for the Go Application under Varying Memory Constraints (1 CPU). The Nanos unikernel's throughput was an order of magnitude higher than Docker's across all memory levels. . . . .	153
9.11	Average Latency and Standard Deviation for the Go Application (1 CPU). The unikernel demonstrated significantly lower average latency and less performance jitter compared to the container. . . . .	153
9.12	CPU-Bound Throughput for the Node.js Application under Varying Memory Constraints (1 CPU). A performance crossover occurred below the 256 MB memory level, where Docker's stability provided a performance advantage. . . . .	154
9.13	Average Latency and Standard Deviation for the Node.js Application (1 CPU). The chart highlights the increase in latency and performance jitter for the unikernel at 128 MB and below. . . . .	155

10.1	The proposed MS2M migration process. . . . .	165
10.2	Overall time performance of MS2M compared to the baseline stop-and-copy [491]. . . . .	166
10.3	Overall time performance of MS2M [491]. . . . .	168
11.1	Detailed Migration Process of MS2M in Kubernetes for individual Pods. . . . .	171
11.2	Overview of MS2M integration into Kubernetes [505]. . . . .	173
11.3	Detailed migration process of MS2M in Kubernetes for Individual Pods with Threshold-Based Cutoff Mechanism [505]. . . . .	176
11.4	Detailed migration process of MS2M in Kubernetes for StatefulSet Pods [505]. . . . .	178
11.5	Distribution of Total Migration Time across test rounds. . . . .	180
11.6	Distribution of Downtime across test rounds. . . . .	181
11.7	Comparison of MS2M for Individual Pods against Stop-and-Copy. . . . .	182
11.8	Comparison of MS2M for Individual Pods with Cutoff mechanism against Stop-and-Copy. . . . .	183
11.9	Comparison of MS2M for StatefulSet Pods against Stop-and-Copy. . . . .	183
11.10	Latency Analysis of MS2M for Individual Pods. . . . .	184
11.11	Latency Analysis of MS2M for Individual Pods with Cutoff Mechanism. . . . .	185
11.12	Latency Analysis of MS2M for StatefulSet Pods. . . . .	186
12.1	Overview of SAGA Framework prototype integrated within Kubernetes cluster. . . . .	199
12.2	Extended $k$ -way Kernighan-Lin algorithm performance analysis. . . . .	200
12.3	Box plots of the latency performance before and after migration. . . . .	201
13.1	Overview of Modules and Repositories in LDM System Architecture. . . . .	206
13.2	Microservice distribution before (left) and after (right) optimization. . . . .	215
13.3	LDM registration time across cluster sizes. . . . .	216
13.4	Migration voting latencies across cluster sizes. . . . .	217



# List of Tables

3.1 Comparison of Related Surveys on Compute Continuum. . . . .	24
3.3 Search Queries Used for the Systematic Literature Review. . . . .	26
4.1 Summary of Resource Management Strategies . . . . .	47
4.3 Comparative Analysis of Domain-Specific Architectures. . . . .	58
5.1 Distribution of Research Topics in Engineering and Technical Solutions category by Year. . . . .	64
6.1 Feature Comparison of Industry 4.0 Architectures. . . . .	105
7.1 Mapping of MAIA Components to the Reference Architecture . . . . .	109
8.1 Comparative Analysis of Microservice Frameworks: Overview and Project Statistics (updated from [463]). . . . .	126
8.3 Comparative Analysis of Microservice Frameworks: Design and Development Experience (updated from [463]). . . . .	127
8.5 Comparative Analysis of Microservice Frameworks: Maturity and Extensibility (updated from [463]). . . . .	128
8.7 Qualitative Comparison of Microservice Frameworks (Part 1: Communication, Networking, Configuration) (updated from [463]). . .	132
8.9 Qualitative Comparison of Microservice Frameworks (Part 2: Resilience, Telemetry, Runtime) (updated from [463]). . . . .	133
8.11 Mapping of Original Toll System Components to the AMR Fleet Management Architecture. . . . .	135



# Bibliographic Notes

Most of the methods, proposals, analyses, and results presented in this dissertation have previously been published in peer-reviewed venues. This section explains the relationship between the dissertation chapters and their corresponding publications, and also lists additional works by the author.

## **Chapters 3, 4, and 5: A Taxonomy of the Compute Continuum**

The taxonomy presented in these chapters is based on a manuscript currently under review for:

H. Dinh-Tuan, “Mapping the Compute Continuum Landscape: A Systematic Review,” submitted to *IEEE Access*.

## **Chapter 6: CRACI: A Cloud-Native Reference Architecture**

The architecture presented in this chapter is an extension of the work accepted for publication in:

H. Dinh-Tuan, “CRACI: A Cloud-Native Reference Architecture for the Industrial Compute Continuum,” in *2025 8th Conference on Cloud and Internet of Things (CIoT)*, IEEE, 2025 (accepted).

## **Chapter 7: Validation of CRACI: A Case Study in Industrial Analytics**

This chapter is an extension of the work originally presented in:

H. Dinh-Tuan, F. Beierle, and S. R. Garzon, “MAIA: a Microservices-based Architecture for Industrial Data Analytics,” in *2019 IEEE International Conference on Industrial Cyber Physical Systems (ICPS)*, IEEE, 2019, pp. 23–30.

## **Chapter 8: A Comparative Analysis of Implementation Paradigms**

This chapter is an extended and updated version of the comparative analysis first published in:

H. Dinh-Tuan, M. Mora-Martinez, F. Beierle, and S. R. Garzon, “Development frameworks for microservice-based applications: Evaluation and comparison,” in *Proceedings of the 2020 European Symposium on Software Engineering*, 2020, pp. 12–20.

## **Chapter 9: A Comparative Analysis of Deployment Paradigms**

The results presented in this chapter are an extension of the work accepted for publication in:

H. Dinh-Tuan, “Unikernels vs. Containers: A Runtime-Level Performance Comparison for Resource-Constrained Edge Workloads,” in *2025 8th Conference on Cloud and Internet of Things (CIoT)*, IEEE, 2025.

## **Chapter 10: Service Migrations in Cloud-Native Environments**

The foundational concepts for the MS2M framework presented in this chapter were first introduced in:

H. Dinh-Tuan and F. Beierle, “MS2M: A message-based approach for live stateful microservices migration,” in *2022 5th Conference on Cloud and Internet of Things (CIoT)*, IEEE, 2022, pp. 100–107.

## **Chapter 11: Optimizing Stateful Microservice Migration in Kubernetes**

This chapter builds upon the MS2M framework and presents an integration analysis with Kubernetes, as detailed in:

H. Dinh-Tuan and J. Jiang, “Optimizing Stateful Microservice Migration in Kubernetes with MS2M and Forensic Checkpointing,” in *2025 28th Conference on Innovation in Clouds, Internet and Networks (ICIN)*, IEEE, 2025, pp. 83–90.

## **Chapter 12: Service Affinity and Centralized Resource Management**

The SAGA framework and the concept of Service Affinity presented in this chapter were first published in:

H. Dinh-Tuan and F. F. Six, “Optimizing Cloud-Native Services with SAGA: A Service Affinity Graph-Based Approach,” in *2024 International Conference on Smart Applications, Communications and Networking (SmartNets)*, IEEE, 2024, pp. 1–6.

## **Chapter 13: Decentralized Resource Management across the Continuum**

The DREAMS framework presented in this chapter is based on a manuscript accepted for publication at:

H. Dinh-Tuan, T. H. Nguyen, and S. R. Pandey, “DREAMS: Decentralized Resource Allocation and Service Management across the Compute Continuum Using Service Affinity,” in *2025 12th International Symposium on Networks, Computers and Communications (ISNCC’25)*, IEEE, 2025.

## Other Related Publications

Other publications by the author produced during the doctoral research, but not included in this dissertation, are listed below:

- H. Dinh-Tuan, S. R. Garzon, and J. Fu, “Secure and Trustful Cross-Domain Communication with Decentralized Identifiers in 5G and Beyond,” in *2024 27th Conference on Innovation in Clouds, Internet and Networks (ICIN)*, IEEE, 2024, pp. 32–36.
- H. Dinh-Tuan, K. Katsarou, and P. Herbke, “Optimizing microservices with hyperparameter optimization,” in *2021 17th International Conference on Mobility, Sensing and Networking (MSN)*, IEEE, 2021, pp. 685–686.
- S. Rodriguez Garzon, H. Dinh-Tuan, M. Mora Martinez, A. Küpper, H. J. Einsiedler, and D. Schneider, “Beyond Certificates: 6G-ready Access Control for the Service-Based Architecture with Decentralized Identifiers and Verifiable Credentials,” in *2024 Joint European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit)*, IEEE, 2024, pp. 830–835.
- F. Beierle, H. Dinh-Tuan, and Y. Wu, “Demonstrator Game Showcasing Indoor Positioning via BLE Signal Strength,” in *2021 17th International Conference on Mobility, Sensing and Networking (MSN)*, IEEE, 2021, pp. 679–680.



# Contents

<b>Declaration of Authorship</b>	<b>iii</b>
<b>Abstract</b>	<b>v</b>
<b>Zusammenfassung</b>	<b>vii</b>
<b>Acknowledgements</b>	<b>ix</b>
<b>Glossary</b>	<b>xi</b>
<b>List of Figures</b>	<b>xiii</b>
<b>List of Tables</b>	<b>xvii</b>
<b>Bibliographic Notes</b>	<b>xix</b>
<b>I Foundations</b>	<b>1</b>
<b>1 Introduction</b>	<b>3</b>
1.1 Contributions of the Dissertation . . . . .	4
1.1.1 A Systematic Taxonomy of the Compute Continuum . . . . .	4
1.1.2 A Reference Architecture for Industrial Compute Continuum . . . . .	4
1.1.3 Comparative Analyses of Implementation and Deployment Paradigms . . . . .	4
1.1.4 A Framework for Live Stateful Service Migration . . . . .	5
1.1.5 Frameworks for Intelligent Resource Management . . . . .	5
1.2 Research Methodology . . . . .	6
1.3 Scope and Research Question Formulation . . . . .	7
1.3.1 Scope . . . . .	7
1.3.2 Research Question Formulation . . . . .	8
Main Research Question . . . . .	8
Sub-Questions . . . . .	8
1.4 Dissertation Structure . . . . .	9
1.5 Chapter Summary . . . . .	10
<b>2 Background</b>	<b>11</b>
2.1 The Evolution of Computing Models: From Cloud to the Continuum . . . . .	11
2.1.1 Cloud Computing . . . . .	11
2.1.2 Edge Computing and other post-Cloud Paradigms . . . . .	12
2.1.3 The Integration of Cloud and Edge . . . . .	14
2.2 The Industrial Revolutions: From Industry 1.0 to Industry 4.0 . . . . .	15

2.3 Microservices as a Key Enabler for Cloud-Native Systems . . . . .	16
2.4 Chapter Summary . . . . .	18
<b>II Mapping the Landscape: A Taxonomy of the Compute Continuum</b>	<b>19</b>
<b>3 Research Design</b>	<b>21</b>
3.1 Research Questions and Objectives . . . . .	21
3.2 Related Work . . . . .	22
3.3 Systematic Review Methodology . . . . .	25
3.3.1 Data Sources . . . . .	25
3.3.2 Search Queries . . . . .	25
3.3.3 Eligibility Criteria . . . . .	27
Inclusion Criteria . . . . .	27
Exclusion Criteria . . . . .	28
3.3.4 Selection Process . . . . .	28
3.3.5 Threats to Validity . . . . .	29
Internal Validity . . . . .	29
External Validity . . . . .	30
Construct Validity . . . . .	30
Conclusion Validity . . . . .	31
3.4 Chapter Summary . . . . .	31
<b>4 A Taxonomy of Research on Compute Continuum</b>	<b>33</b>
4.1 Industry-specific Use Cases and Applications . . . . .	34
4.1.1 Computing and Network Infrastructure . . . . .	34
4.1.2 Disease Control and Monitoring . . . . .	36
4.1.3 Healthcare and Medical Applications . . . . .	36
4.1.4 Industry 4.0 . . . . .	36
4.1.5 Smart Systems . . . . .	36
4.1.6 Sustainability . . . . .	37
4.1.7 Transportation and Logistics . . . . .	37
4.2 Engineering and Technical Solutions . . . . .	37
4.2.1 Data Management . . . . .	38
Data Pipelines Management . . . . .	38
Storage Management . . . . .	39
4.2.2 Development and Deployment . . . . .	40
Deployment Frameworks . . . . .	40
Deployment Strategies . . . . .	40
Development Technologies and Tools . . . . .	41
4.2.3 Enabling Technologies and Paradigms . . . . .	41
6G . . . . .	42
Cloud Native Technologies . . . . .	42
Digital Twin . . . . .	42
Quantum Computing . . . . .	43
4.2.4 Interoperability and Integration . . . . .	43
4.2.5 Modeling and Simulation . . . . .	44

Modeling Techniques . . . . .	44
Simulation Frameworks . . . . .	44
4.2.6 Networking . . . . .	45
4.2.7 Operational Management . . . . .	45
Service Life-cycle Management . . . . .	45
Resource Management . . . . .	46
Workflow Management . . . . .	50
4.2.8 Optimization . . . . .	51
Data Transmission and Bandwidth Optimization . . . . .	51
Multi-objective Optimization . . . . .	51
Orchestration . . . . .	51
Resource Allocation and Utilization Optimization . . . . .	52
Service Optimization . . . . .	52
Task Optimization . . . . .	53
Workflow Optimization . . . . .	54
4.2.9 Security and Privacy . . . . .	54
Data Security and Integrity . . . . .	54
Network and Communication Security . . . . .	55
Secure Execution and Orchestration . . . . .	55
4.3 System Architectures and Designs . . . . .	56
4.3.1 General-purpose Architectures . . . . .	56
4.3.2 Domain-specific Architectures . . . . .	56
Agriculture . . . . .	59
Emergency and Disaster Management . . . . .	59
Healthcare and Medical Applications . . . . .	59
Industry 4.0 . . . . .	60
AI and Machine Learning . . . . .	60
Internet of Things . . . . .	60
Recommender Systems . . . . .	61
Robotics . . . . .	61
Smart Systems . . . . .	61
Telecommunication . . . . .	61
XR Applications . . . . .	62
4.4 Chapter Summary . . . . .	62
<b>5 Discussion, Conclusions, and Future Directions</b>	<b>63</b>
5.1 Revisiting the Research Questions . . . . .	63
5.1.1 Mapping Research Directions and Activity Levels (RQ1) . . . . .	63
5.1.2 Clarifying Terminology and Formulating a Definition (RQ2) . . . . .	65
5.1.3 A Synthesis of Trends and Values that Shape the Compute Continuum (RQ3 & RQ4) . . . . .	68
Decentralization: Enabling Flexibility and Customization . . . . .	68
Cloud-Native Technologies: Key Enabler for Unified Scalability .	69
Low-latency Processing . . . . .	69
Security and Privacy in a Decentralized Continuum . . . . .	69
Interoperability and Standardization . . . . .	70
5.1.4 Identifying Research Gaps (RQ5) . . . . .	70

Heterogeneity Management . . . . .	70
Federated Coordination . . . . .	71
Stateful Service Management . . . . .	71
Security and Privacy . . . . .	71
Under-explored Domains . . . . .	71
5.2 Chapter Summary . . . . .	72
<b>III The Core Architecture: A Cloud-Native Reference Architecture for Industry 4.0</b>	<b>73</b>
<b>6 CRACI: A Cloud-Native Reference Architecture</b>	<b>75</b>
6.1 Introduction . . . . .	75
6.2 Background and Motivation . . . . .	75
6.2.1 Formal Industry Standards and Models . . . . .	76
The Traditional Hierarchy: ISA-95 . . . . .	76
Lifecycle Management: IEC 62890 . . . . .	77
RAMI 4.0: A Model for Industry 4.0 . . . . .	77
6.2.2 Industry-driven Operational Frameworks . . . . .	79
FIWARE Smart Industry Reference Architecture . . . . .	79
NAMUR Open Architecture (NOA) . . . . .	80
AIOTI's Computing Continuum Reference Architecture . . . . .	81
6.2.3 Academic Reference Architecture . . . . .	81
LASFA (LAsim Smart FActory) Architecture for Smart Manufacturing . . . . .	81
Stuttgart IT Architecture for Manufacturing (SITAM) . . . . .	83
The 8C Architecture for Industry 4.0 Smart Factories . . . . .	84
6.2.4 Software Architecture Patterns . . . . .	85
Cloud-Native Principles & The Twelve-Factor App . . . . .	85
Event-Driven Architecture and Publish/Subscribe Pattern . . . . .	86
6.2.5 Summary . . . . .	86
6.3 Design Process and Methodology . . . . .	87
6.3.1 Design Methodology . . . . .	87
6.3.2 Architectural Drivers . . . . .	87
Design Purpose and Objectives . . . . .	88
Quality Attribute Goals . . . . .	88
Primary Functional Requirements . . . . .	88
Architectural Concerns . . . . .	89
Constraints . . . . .	90
6.4 The Proposed Reference Architecture . . . . .	91
6.4.1 Architectural Overview . . . . .	91
The Four Foundational Pillars . . . . .	93
6.4.2 Functional Breakdown of Architectural Components . . . . .	94
Physical Assets (Brownfield & Greenfield) . . . . .	94
Adaptation Services . . . . .	94
Edge Platform Services . . . . .	95
Semantic Context Services . . . . .	95

Core Platform Services . . . . .	96
Asset Representation Services . . . . .	97
Operational Intelligence Services . . . . .	97
Orchestration Services . . . . .	98
Strategic Intelligence Services . . . . .	98
External and User-Facing Services . . . . .	99
<b>6.5 Architectural Design Rationale . . . . .</b>	<b>100</b>
6.5.1 Decoupling the ISA-95 Hierarchy . . . . .	100
6.5.2 Extending RAMI 4.0 with Operational Aspects . . . . .	101
6.5.3 Completing the NOA Concept . . . . .	101
6.5.4 Generalizing the FIWARE Architecture . . . . .	102
6.5.5 Implementing the AIOTI Functional Models . . . . .	102
6.5.6 Improving LASFA's Data Architecture . . . . .	103
6.5.7 Formalizing the Cross-Cutting Concerns of the SITAM . . . . .	104
6.5.8 Implementing the Integration Layers of 8C Architecture . . . . .	104
6.6 Chapter Summary . . . . .	105
<b>7 Validation of CRACI: A Case Study in Industrial Analytics . . . . .</b>	<b>107</b>
7.1 Introduction to the MAIA Use Case . . . . .	107
7.2 Mapping MAIA Architecture to the Reference Architecture . . . . .	108
7.3 Implementation . . . . .	110
7.3.1 Implementation of the Core Platform Services . . . . .	111
7.3.2 Implementation of the Main Logic Services . . . . .	112
7.4 Evaluation and Results . . . . .	113
7.4.1 Evaluation Setup . . . . .	113
7.4.2 Experimental Validation and Analysis . . . . .	113
Analyzing Microservice Overhead and Mitigation . . . . .	114
Evaluating System Scalability and Performance . . . . .	117
7.5 Evaluation Summary . . . . .	119
7.5.1 Validated Strengths of the Reference Architecture . . . . .	119
7.5.2 Identified Challenges and Areas for Improvement . . . . .	119
7.6 Chapter Summary . . . . .	120
<b>IV Implementing the Architecture: Comparative Analyses of Development and Deployment Paradigms . . . . .</b>	<b>121</b>
<b>8 A Comparative Analysis of Implementation Paradigms . . . . .</b>	<b>123</b>
8.1 Introduction . . . . .	123
8.2 Background and Related work . . . . .	123
8.3 The Selected Microservice Frameworks . . . . .	124
8.4 Theoretical Comparative Analysis . . . . .	125
8.4.1 Design Philosophy . . . . .	125
8.4.2 Developer Experience and Testing Philosophy . . . . .	125
8.4.3 Maturity, Governance, and Commercial Ecosystem . . . . .	129
8.4.4 Modularity and Extensibility . . . . .	129
8.4.5 Alignment with the CRACI Architecture's Core Platform Services	129
8.5 Empirical Comparative Analysis . . . . .	134

8.5.1	Use Case . . . . .	134
8.5.2	Experimental Setup . . . . .	136
8.5.3	The Impact of JVM Warm-up on Service Startup . . . . .	136
8.5.4	End-to-End Latency Performance . . . . .	136
8.5.5	Resource Consumption . . . . .	138
8.6	Chapter Summary . . . . .	140
8.6.1	Framework Choice is Context-Dependent . . . . .	140
8.6.2	Validation of the CRACI Core Platform Services . . . . .	140
8.6.3	Performance Governed by Bottlenecks . . . . .	141
8.6.4	The Impact of Resource Efficiency . . . . .	141
8.6.5	Implementation Details Can Outweigh Theoretical Advantages .	141
8.6.6	The Trade-offs between Coupling and Interoperability . . . . .	142
<b>9</b>	<b>A Comparative Analysis of Deployment Paradigms</b>	<b>143</b>
9.1	Introduction . . . . .	143
9.2	Background . . . . .	144
9.3	Evaluation Design . . . . .	145
9.4	Results and Analysis . . . . .	146
9.4.1	Static Properties: Footprint and Startup Time . . . . .	147
9.4.2	Performance Evaluation: I/O-Bound Workload . . . . .	147
	Resource-rich Scenario . . . . .	148
	Resource-constrained Scenario . . . . .	149
9.4.3	Performance Evaluation: CPU-Bound Workload . . . . .	151
	Resource-rich Scenario . . . . .	152
	Resource-constrained Scenario . . . . .	152
9.5	Discussion . . . . .	154
9.5.1	Static Properties: Unikernel Agility . . . . .	155
9.5.2	I/O Throughput: Kernel vs. Runtime Efficiency . . . . .	155
9.5.3	CPU-Bound Performance: Specialization Advantage . . . . .	156
9.5.4	Crossover Point: Stability in Constrained Environments . . . . .	156
9.6	Chapter Summary . . . . .	156
<b>V</b>	<b>Managing State: Live Migration of Stateful Services</b>	<b>159</b>
<b>10</b>	<b>Service Migration in Cloud-Native Environments</b>	<b>161</b>
10.1	Introduction . . . . .	161
10.2	Related Work . . . . .	162
10.3	The MS2M Framework: Concept and Design . . . . .	163
10.3.1	Actors and Components . . . . .	163
10.3.2	The Five-Phase Migration Process . . . . .	163
10.3.3	Empirical Evaluation . . . . .	164
	Experimental Setup . . . . .	165
	Results and Findings . . . . .	166
10.3.4	Chapter Summary . . . . .	167

<b>11 Optimizing Stateful Microservice Migration in Kubernetes</b>	<b>169</b>
11.1 Introduction . . . . .	169
11.2 Integrating MS2M with Kubernetes . . . . .	170
11.2.1 The Adapted Four-Actor Model . . . . .	170
11.2.2 The Integrated Pod Migration Procedure . . . . .	171
11.3 Performance Optimization: The Threshold-Based Cutoff Mechanism .	172
11.3.1 Problem Formulation: Modeling the Replay Queue . . . . .	173
11.3.2 The Cutoff Mechanism: A Deterministic Solution . . . . .	174
11.3.3 The Optimized Migration Procedure . . . . .	175
11.4 Extending Support for StatefulSets . . . . .	175
11.4.1 The Challenges of StatefulSets . . . . .	175
11.4.2 The Adapted Migration Process for StatefulSets . . . . .	177
11.4.3 Analysis of the Trade-off . . . . .	178
11.5 Evaluation and Analysis . . . . .	179
11.5.1 Experimental Setup . . . . .	179
11.5.2 Evaluation Results . . . . .	179
Stability evaluation . . . . .	180
Comparative Performance Analysis Against Baseline . . . . .	181
Analysis of Migration Bottlenecks Across Phases . . . . .	184
11.6 Chapter Summary . . . . .	186
<b>VI Managing Scale: Resource Coordination across the Continuum</b>	<b>189</b>
<b>12 Service Affinity and Centralized Resource Management</b>	<b>191</b>
12.1 Introduction . . . . .	191
12.2 Related Work . . . . .	192
12.3 The Service Affinity Concept . . . . .	194
12.3.1 Data Affinity ( $d$ ) . . . . .	194
12.3.2 Coupling Affinity ( $c$ ) . . . . .	195
12.3.3 Functional Affinity ( $f$ ) . . . . .	195
12.3.4 Operational Affinity ( $o$ ) . . . . .	195
12.3.5 Security & Privacy Affinity ( $p$ ) . . . . .	196
12.3.6 Weighted-graph representation of the microservices cluster . .	197
12.4 The Partitioning Algorithm . . . . .	197
12.5 The SAGA Framework . . . . .	197
12.5.1 Architecture Overview . . . . .	198
12.6 Evaluation and Discussion . . . . .	199
12.6.1 Algorithm Complexity Evaluation . . . . .	199
12.6.2 Algorithm Runtime Analysis . . . . .	199
12.6.3 Prototype Evaluation . . . . .	200
<b>13 Decentralized Resource Management across the Continuum</b>	<b>203</b>
13.1 Introduction . . . . .	203
13.2 Related Work . . . . .	204
13.3 Concept and Design . . . . .	205
13.3.1 Design Principles . . . . .	205

13.3.2 Modules . . . . .	206
13.3.3 Repositories . . . . .	209
13.4 Implementation . . . . .	209
13.4.1 Migration candidate selection . . . . .	210
13.4.2 Voting Procedure for Migration Approval . . . . .	211
13.5 Evaluation . . . . .	213
13.5.1 Qualitative evaluation . . . . .	213
13.5.2 Quantitative evaluation . . . . .	215
LDM registration . . . . .	216
Migration Voting . . . . .	217
13.6 Chapter Summary . . . . .	218
<b>VII Dissertation Conclusion</b>	<b>219</b>
<b>14 Contribution Summary</b>	<b>221</b>
14.1 Novel Architecture and Frameworks for the Industrial Compute Continuum . . . . .	221
14.2 Answers to Sub-Questions . . . . .	224
RQ1: Foundational Architectural Principles . . . . .	224
RQ2: Implementation and Deployment Trade-offs . . . . .	224
RQ3: Dynamic Management of Stateful Services . . . . .	226
<b>15 Discussion and Conclusion</b>	<b>227</b>
15.1 Key Insights . . . . .	227
15.1.1 The Complexity of Trade-off Decisions . . . . .	227
15.1.2 The Importance of Flexibility . . . . .	228
15.1.3 The Significance of Application-Aware Management . . . . .	228
15.1.4 The Hidden Overhead of Cloud-Native Architecture . . . . .	229
15.2 Limitations and Future Work . . . . .	230
<b>Bibliography</b>	<b>233</b>

# **Part I**

# **Foundations**



# 1 Introduction

The Fourth Industrial Revolution (Industry 4.0) is reshaping the manufacturing landscape by integrating digital intelligence into physical production processes. This new era, however, is defined by a set of challenging and often conflicting technical requirements. On the one hand, it demands real-time data analytics from Internet of Things (IoT) sensors or low-latency control of autonomous robotics. On the other hand, it requires the capacity for large-scale data processing for advanced predictive maintenance and enterprise-wide optimizations.

Traditional industrial architectures, many of which are based on the rigid ISA-95 automation pyramid, are inadequate for meeting these diverse needs. The model creates a strict separation between the real-time operational domain and the analytical enterprise domain, restricting the flexible data exchange necessary for agile operations. At the same time, the *Compute Continuum* (CC) has been introduced as the next evolution of computing models, following *Cloud Computing* and *Edge Computing*. This paradigm represents a unified spectrum of resources that seamlessly integrate, from powerful, centralized clouds to responsive, decentralized edge infrastructure. It allows computational workloads to be placed dynamically, whether in the cloud for large-scale analysis or at the edge for low-latency processing. By combining the strengths of both, the CC enables modern industrial systems to fully capture the benefits of Industry 4.0.

The strategic importance of this next industrial revolution is emphasized by its massive economic impact. Manufacturing remains a critical component of the global economy, and the transition to Industry 4.0 is expected to become a market worth \$884.84 billion by 2034 [1]. This growth is driven by the explosion of industrial data and the technologies designed around it, such as the Industrial Internet of Things (IIoT), which is expected to exceed \$4 trillion by 2032 [2], and Industrial Edge Computing, forecast to reach a \$44.73 billion market in the same period [3]. In Europe, manufacturing accounts for 14.67% of GDP and generates a trade surplus of €421 billion annually [4, 5]. Initiatives like the *Made in Europe* partnership have already attracted billions of Euros to strengthen Europe's leadership and competitiveness in sustainable, digital, and human-centric manufacturing. Moreover, recent disruptions in global supply chains and the increasing geopolitical competition among major powers have made the manufacturing a top priority for many economies.

This dissertation addresses this challenge by presenting a structured framework for the design, development, deployment, and optimization of modern industrial applications across the CC. Building such complex systems requires a systematic approach that spans six critical stages: motivation establishment (*Motivating*), landscape exploration (*Understanding*), architectural structuring (*Designing*), implementation and deployment evaluation (*Implementing*), system operation (*Operating*), and performance improvement (*Optimizing*).

## 1.1 Contributions of the Dissertation

This dissertation is structured according to the aforementioned framework, with each stage corresponding to a different contribution. These contributions are explained in detail in the following sections.

### 1.1.1 A Systematic Taxonomy of the Compute Continuum

The dissertation begins by addressing the fragmented CC research landscape. It presents a taxonomy derived from an extensive Systematic Literature Review (SLR) of 291 peer-reviewed publications from 2021 to mid-2024. This provides a comprehensive view of the state-of-the-art, systematically classifying scientific contributions into key areas including industry-specific applications, engineering solutions, and system architectures. By doing so, it clarifies inconsistent terminology and establishes a unified conceptual definition of the CC. Key directions are identified, and, more importantly, research gaps such as the absence of a unified Industry 4.0-specific architecture for CC are recognized. This taxonomy motivates and justifies the subsequent research presented in this work.

### 1.1.2 A Reference Architecture for Industrial Compute Continuum

In response to the identified architectural gap, this dissertation also proposes a novel *Cloud-native Reference Architecture for the Compute Continuum in Industry 4.0* (CRACI), which was designed specifically to address the limitations of legacy industrial models, particularly ISA-95. The architecture is built following decoupled, data-centric, cloud-native, and event-driven principles to eliminate data silos and enable flexible cross-layer communication. Its design is guided by the *Attribute-Driven Design (ADD 3.0)* methodology, so that critical quality attributes such as security, governance, observability, and lifecycle management are treated as foundational architectural elements. The architecture's feasibility is then validated through a smart manufacturing use case, demonstrating how its abstract principles can be deployed in a concrete industrial analytics system.

### 1.1.3 Comparative Analyses of Implementation and Deployment Paradigms

Following the CRACI design, the next step is to demonstrate that the proposed architecture can be implemented using existing technologies. This dissertation, therefore, provides a detailed evaluation of key implementation options in two crucial stages: service development and service deployment.

First, the development stage is addressed by analyzing several microservice frameworks used to implement the service's core logic. Rather than presenting only a theoretical comparative analysis of design philosophies and features, this work implements a representative data processing pipeline multiple times, with each version built using a different framework. By offering a perspective that connects both theoretical and practical aspects, this dissertation provides an in-depth analysis

of the trade-offs involved in selecting specific frameworks for service development, as well as how implementation details can influence the theoretical advantages of microservices.

Second, the deployment stage is addressed by analyzing the runtime environments, focusing specifically on the trade-offs between containers and unikernels. This analyzes how performance is affected not only by the deployment paradigm but also by the application's programming model, comparing an Ahead-of-Time (AOT) compiled language (Go) with a Just-in-Time (JIT) compiled one (Node.js). The benchmark identifies critical trade-offs in startup time, resource footprint, and performance under varying workloads and resource constraints. The findings present quantitative evidence to support informed decisions-making when selecting the optimal deployment paradigm, taking into account the specific workloads and constraints of the target industrial environment.

### 1.1.4 A Framework for Live Stateful Service Migration

To address the agility and flexibility required by Industry 4.0, this dissertation also proposes *Message-based Stateful Microservice Migration* (MS2M), a novel message-based migration framework. The key contribution of MS2M is its ability to reuse the system's existing asynchronous messaging infrastructure to indirectly reconstruct service state during migration, rather than relying on low-level memory synchronization. In addition, this dissertation includes the full architectural integration of the MS2M framework into the Kubernetes orchestration platform, making use of its optimized Checkpoint/Restore feature and introducing a *Threshold-Based Cutoff Mechanism* as a key performance optimization to ensure predictable migration times in high-load industrial scenarios.

### 1.1.5 Frameworks for Intelligent Resource Management

While the MS2M framework provides the mechanism for migrating a stateful service, it does not decide when a migration should occur. This final contribution addresses the intelligence needed to orchestrate this process. It first formalizes the inter-service relationships through the concept of *Service Affinity* (SA), which captures multi-dimensional dependencies among services. This model serves as the foundation to develop two distinct management frameworks:

The first is the *Service Affinity Graph-based Approach* (SAGA), a centralized, graph-based framework that applies SA to optimize service placement across the continuum using a heuristic method.

The second extends this concept into *DREAMS (Decentralized Resource Allocation and Service Management using Service Affinity)*, a fully decentralized, fault-tolerant, multi-agent framework. DREAMS adopts a consensus-driven voting protocol to allow autonomous domains to collaboratively negotiate and agree on service migrations, enabling scalable and robust management for large-scale, multi-stakeholder industrial systems.

## 1.2 Research Methodology

The research conducted in this dissertation follows the Design Science Research (DSR) methodology [6]. DSR is a problem-solving approach focused on the systematic creation and evaluation of novel artifacts, such as *constructs*, *models*, *methods*, or *instantiations*, designed to solve identified organizational or technical problems [6, 7]. Originating from engineering and information systems, DSR seeks to generate scientific knowledge through the act of building and evaluating artifacts, extending the boundaries of human and organizational capabilities [8, 9, 10].

This work follows the well-established DSR frameworks proposed by Hevner et al. [11] and Peffers et al. [7], providing a structured research process. As illustrated in the process model in Figure 1.1, the DSR lifecycle typically involves six stages: *problem identification*, *objective definition*, *design and development*, *demonstration*, *evaluation*, and *communication*. Through the application of DSR, the primary artifacts created in this work are:

- The Taxonomy of Compute Continuum Research (as a construct and model).
- The Cloud-native Reference Architecture for the Compute Continuum in Industry 4.0 (CRACI) (as a model and method).
- The MS2M Framework for stateful microservice migration (as a method).
- The SA concept as well as SAGA and DREAMS Frameworks for centralized and decentralized resource management (as a model and method).

Within the overall DSR methodology, this dissertation integrates several methodological components. Firstly, to ensure DSR principles of *Problem Relevance* and *Research Rigor*, this dissertation begins with a formal SLR in Part II. Following the PRISMA 2020 guidelines [12], the SLR systematically identifies, synthesizes, and maps the existing body of knowledge on the CC. This process serves two critical functions: (1) *Problem Identification and Motivation*: The SLR identifies the key research gaps that directly justify the need for the artifacts developed in this dissertation, and (2) *Positioning the Contribution*: It ensures that the proposed artifacts are novel and contribute meaningfully to the existing body of knowledge.

The proposed CRACI reference architecture (Part III) is designed using the ADD 3.0 methodology, demonstrating *Research Rigor* and *Design as an Artifact* principles. Its applicability and relevance are then demonstrated through an architectural analysis and a case study through the Microservices-based Architecture for Industrial Data Analytics (MAIA) use case. This mapping of the abstract architecture to a concrete implementation serves as a form of *Design Evaluation*.

The *Design Evaluation* principle is further demonstrated through the use of *quantitative experimental benchmarking*, a practice central to Quantitative DSR (QDSR). Part IV conducts extensive experiments to evaluate the performance of different development frameworks and deployment options. These experiments provide quantitative insights to evaluate the performance trade-offs of the proposed architectural designs.

The DSR guideline of *Design as a Search Process* is evident in the iterative nature of this research. The shortcomings identified during the evaluation of the centralized

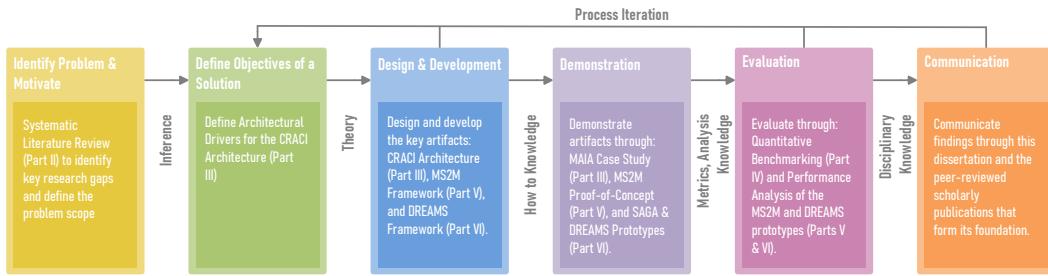


FIGURE 1.1: The DSR methodology process model adopted in this Dissertation [11].

SAGA framework directly motivated a research iteration, leading to the design of the decentralized DREAMS framework. Similarly, the initial proof-of-concept for the MS2M framework is later refined by integrating it into Kubernetes and introducing the Threshold-Based Cutoff Mechanism to address performance limitations discovered during the first evaluation cycle.

Finally, this work adheres to the *Communication of Research* principle through two primary aspects. The first is this dissertation itself, which presents the problem, the artifacts, their utility and evaluation. The dissertation as a whole loosely follows the *System Development Life Cycle* [13], thereby providing a structured engineering approach to applying the CC in Industry 4.0. The second aspect consists of the multiple peer-reviewed publications that form the basis of this work, ensuring that the research contributions have been examined by and disseminated to the scientific community.

## 1.3 Scope and Research Question Formulation

### 1.3.1 Scope

The scope of this dissertation is centered around engineering and developing cloud-native solutions spanning across the CC, with a specific focus on modern industrial application use cases. Specifically, this study is structured around three technical pillars that directly relate to the research questions:

1. *Architecture*: This dissertation proposes a novel cloud-native reference architecture designed to overcome the structural limitations of traditional industrial models. This includes defining the key components, communication patterns, and guiding principles for designing a decoupled, event-driven system.
2. *Implementation & Deployment*: The scope includes evaluating the technical trade-offs involved in implementing the proposed reference architecture. This involves both qualitative and quantitative comparisons of cloud-native development frameworks and modern deployment paradigms to evaluate their performance, resource efficiency, and scalability.
3. *Resource Management*: Also included in this dissertation is the design and validation of novel strategies for managing a dynamic and interconnected industrial system. This includes techniques for stateful service migration and decentralized resource management that use application-level semantics to maintain operational efficiency.

To maintain a clear focus, several related areas are intentionally considered out of the scope of this work:

1. *Security Analysis*: While this dissertation acknowledges the role of security in industrial systems, it does not explicitly provide a detailed security analysis.
2. *Economic and Business Models*: The primary focus of this work is on technical performance and architectural design, as well as service management techniques. Therefore, a comprehensive analysis of business models, return on investment (ROI), or market adoption strategies is out of scope.
3. *Hardware and Network Protocol Design*: This dissertation uses existing commodity hardware and standard network protocols. The design of new physical sensors, custom edge hardware, or low-level network communication protocols are not part of this work.

### 1.3.2 Research Question Formulation

Applying the Design Science practices mentioned by Wieringa [14], in this work, a main research question has been defined. This main research question leads to three sub-questions to cover the design, implementation, and operational aspects of the industrial compute continuum.

#### Main Research Question

**Main Research Question:** *How can a unified cloud-native reference architecture be designed and implemented within the Compute Continuum to meet the changing demands of Industry 4.0?*

This sub-question addresses the core technical challenge of applying cloud-native principles to integrate centralized, cloud-based intelligence with decentralized, edge-based operations, which is crucial for modern industrial systems.

#### Sub-Questions

**RQ1:** *What are the architectural principles and design patterns that form the foundation of a decoupled, and event-driven system capable of overcoming the structural limitations in traditional hierarchical models such as ISA-95?*

This question focuses on the foundational design aspects, involving understanding the theoretical and practical principles that will guide the design of the architecture.

**RQ2:** *What are trade-offs in performance, resource efficiency, and scalability between key implementation (microservices frameworks) and deployment paradigms (containers vs. unikernels) when realizing this architecture?*

This sub-question focuses on the practical aspects of the proposed architecture, providing an evidence-based perspective that supports decision-making in selecting appropriate related technical solutions.

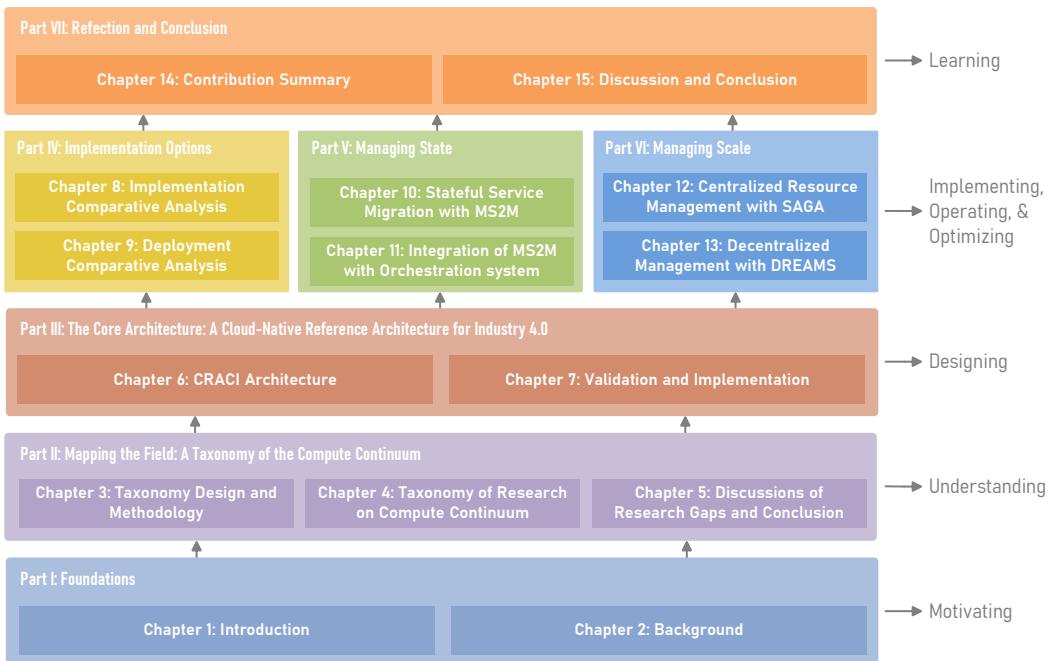


FIGURE 1.2: Dissertation Structure.

**RQ3:** *How can stateful services be dynamically managed and migrated across the compute continuum by utilizing application-level semantics and decentralized coordination to ensure both high availability and operational efficiency?*

This question examines two aspects of operational management. The first concerns how techniques provided by cloud-native principles can be applied in industrial contexts to improve operation efficiency. The second explores which new mechanisms must be developed to meet the specific requirements of Industry 4.0.

## 1.4 Dissertation Structure

As outlined in the six-stage approach (Motivating, Understanding, Designing, Implementing, Operating, and Optimizing) and reflected in the five main contributions of this dissertation, the structure of the dissertation follows the same logic. It is organized into seven parts, each building upon the previous one, moving from foundational understanding to architectural design and finally to implementation and management, as illustrated in Figure 1.2.

**Part I: Foundations** introduces the research context, outlines the dissertation's contributions, defines the core research questions, and details the research methodology. It also provides the necessary background on the evolution of computing models, industrial revolutions, and key technologies relevant to the presented research.

**Part II: Mapping the Field** presents a systematic literature review and a taxonomy of the CC. This part establishes the current state-of-the-art, identifies critical research gaps, and provides the motivation for the subsequent parts.

**Part III: The Core Architecture** details the primary artifact of this dissertation: the Cloud-native Reference Architecture for the Compute Continuum in Industry 4.0

(CRACI). This part presents the architectural design, its foundational pillars, and validates its design through the MAIA case study.

**Part IV: Realizing the Architecture** provides a comparative analysis of the key implementation and deployment paradigms required to realize the CRACI architecture. It quantifies the trade-offs between different microservice frameworks and between container and unikernel deployment models.

**Part V: Managing State** addresses the critical challenge of stateful service migration. It introduces the MS2M framework and details its integration and optimization with Kubernetes.

**Part VI: Managing Scale** addresses the challenge of decentralized coordination. It presents two advanced frameworks, SAGA and DREAMS, which use Service Affinity and consensus protocols for intelligent, scalable resource management across the continuum.

**Part VII: Conclusion** synthesizes the dissertation's findings. It summarizes the primary contributions, revisiting the research questions, and discusses the key insights and limitations of the work. It concludes by outlining directions for future research.

## 1.5 Chapter Summary

In this first chapter, a high-level overview of the motivations, research questions, and research methodology of this dissertation has been presented. The six-stage framework (Motivating, Understanding, Designing, Implementing, Operating, and Optimizing) forms the backbone of the entire dissertation. Both the overall structure and the main scientific contributions are organized around this framework, which also reflects the logical process adopted to address the challenges of applying the Compute Continuum to modern industrial systems.

In the following chapter, the dissertation presents the historical evolution of different computing models that have led to the concept of the Compute Continuum. In parallel, it presents the progression of the industrial revolutions that led to the convergence of Information Technology (IT) and Operational Technology (OT), a topic discussed in later chapters. It further introduces several cloud-native principles and technologies that are used to enable the integration of the CC into industrial systems.

# 2 Background

## 2.1 The Evolution of Computing Models: From Cloud to the Continuum

### 2.1.1 Cloud Computing

The history of *Cloud Computing* dates back to the 1960s, with the development of time-sharing concepts and the *Intergalactic Computer Network*, proposed by J.C.R. Licklider, who envisioned a way to democratize computing [15]. In his vision, multiple computers were connected, and users access computing resources remotely rather than owning dedicated machines. A similar idea was proposed in 1961 by John McCarthy at MIT, who stated that “computing can be sold as a utility, like water and electricity” [16]. The term *Cloud Computing* was first used in a 1996 internal document by Compaq [17].

In 1999, Salesforce began delivering enterprise applications over the Internet, marking the start of the Cloud computing boom [18]. Cloud-based services became widely available with offerings such as Amazon Web Services (AWS), which introduced storage and computational services in 2002, followed by the commercial launch of Amazon’s Elastic Compute Cloud (EC2) in 2006. Other providers followed, such as Google with Google App Engine in 2008 [19]. Other major companies, including Microsoft, Alibaba, IBM, Oracle, and HP, also launched their own Cloud services around this time.

*Cloud Computing* is officially defined by the National Institute of Standards and Technology (NIST) as “a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction” [20]. Based on the services offered, this computing model can be divided into four main categories.

- *Infrastructure as a Service (IaaS)*: As the name suggests, IaaS offers the essential components of computing, such as servers, virtual machines (VMs), storage, and networks. Through advancements in virtualization and network management, these services are provided on-demand by a Cloud provider [21]. Some researchers, such as Wang et al. [22], even propose subdividing IaaS into HaaS (Hardware as a Service) and DaaS (Data as a Service), but IaaS is generally viewed as a single service model.
- *Platform as a Service (PaaS)*: PaaS extends IaaS by providing a platform in the Cloud for the development, testing, delivery, and management of applications

without the direct infrastructure management. This allows developers to focus on application logic.

- *Software as a Service (SaaS)*: SaaS delivers fully functional software applications over the Internet on top of PaaS and IaaS infrastructures [23]. Users access applications via browsers or APIs without managing the underlying infrastructure or platform.
- *Serverless Computing*: This is a recent addition to Cloud service models and closely overlaps with PaaS [24]. In this paradigm, the Cloud provider manages the underlying infrastructure while users deploy and run their applications. Serverless computing dynamically allocates resources in response to incoming requests or events, and scales up or down automatically without manual intervention. A key feature of this model is also known as *scale-to-zero* capability, where no compute resources are consumed, and no charges are incurred when there are no incoming requests or events.

Together, these service models cover the entire spectrum from infrastructure management to full application delivery, enabling benefits such as scalability, flexibility, reliability, security, and integration capability [25]. Importantly, the primary business motivation for adopting Cloud computing is the potential to enhance operational efficiency and cost-effectiveness. Cloud computing follows a *pay-as-you-go* billing model, effectively shifting IT costs from *CAPEX (Capital Expenditure)* to *OPEX (Operational Expenditure)* [26]. This allows enterprises to adjust resources dynamically in response to demand and eliminates the need for significant upfront capital expenditure on infrastructure, economically benefiting businesses of all sizes. However, the centralized nature of Cloud computing introduces a fundamental limitation, which is latency [27, 28]. Since the servers are collocated in several large-scale datacenters, all data must be transmitted over the Internet for processing and then returned, creating delays that are unacceptable in many latency-sensitive applications, especially in industrial contexts.

### 2.1.2 Edge Computing and other post-Cloud Paradigms

The advancements in semiconductor technology have continuously reduced the cost of computation, enabling the placement of computing resources closer to data sources. This aims to address the high-latency, data privacy, and scalability issues inherent in the Cloud model, leading to the advent of a new paradigm: *Edge Computing*.

The origins of Edge computing can be traced back to the 1990s with the development of Content Delivery Networks (CDNs) by companies such as Akamai, which cached static content at the network edges to speed up web page loading times [29]. However, the proliferation of mobile and Internet of Things (IoT) devices, which requires real-time data processing, has transformed the Edge from a simple caching layer into a fully-fledged computational platform. The idea of moving computational resources from the centralized Cloud closer to the data source has appeared in several different proposals.

For example, in 2012, Cisco introduced the term *Fog Computing* for dispersed cloud infrastructures [30], describing a platform that extends Cloud capabilities to

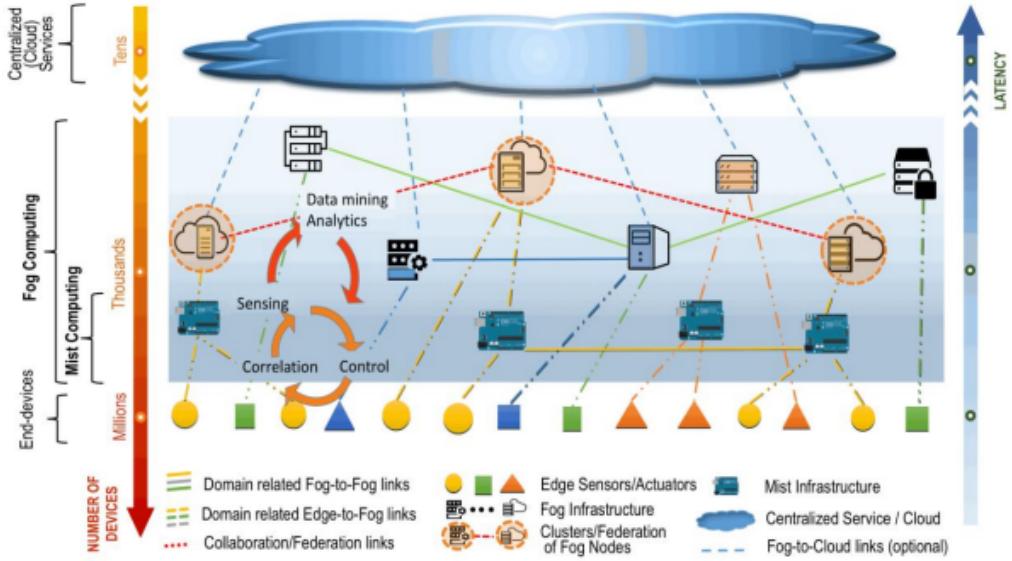


FIGURE 2.1: Fog computing as defined by the National Institute of Standards and Technology (NIST)[31].

the network edge and brings computing, storage, and networking closer to data sources. Fog computing is formally defined by NIST [31] (Figure 2.1) as: “Fog computing is a layered model for enabling ubiquitous access to a shared continuum of scalable computing resources. The model facilitates the deployment of distributed, latency-aware applications and services, and consists of fog nodes (physical or virtual), residing between smart end-devices and centralized (Cloud) services.”

In the context of telecommunication systems, there is a similar concept named *Mobile Edge Computing* [32]. It offers Cloud computing capabilities at the Edge of the Radio Access Network (RAN) as shown in Figure 2.2. The first real-world MEC platform was introduced by Nokia Networks in 2013 and later standardized by European Telecommunications Standards Institute (ETSI) Industry Specifications Group (ISG) [33]. Later, this was renamed to *Multi-access Edge Computing* to reflect the interests of both cellular and non-cellular operators across industries [34].

Another concept worth mentioning is *Mist Computing* [35]. In this, the processing and decision-making are pushed further down to the very edge of the network, directly on IoT sensors and actuators. This results in a higher level of system autonomy and reduced latency as well as bandwidth usage, lowering power consumption [36].

*Dew Computing* [37, 38, 39] is another related term, which is defined as “an on-premises computer software-hardware organization paradigm in the Cloud computing environment where the on-premises computer provides functionality that is independent of Cloud services and is also collaborative with Cloud service” [40]. The core idea is based on the assumption that Internet connectivity may not always be available, requiring on-premises devices to operate autonomously without Cloud dependence.

Many works use the terms *Edge Computing* and *Fog Computing* interchangeably [41, 42]. According to the OpenFog Consortium, Fog computing is often mistakenly referred to as Edge computing, although there are significant distinctions between

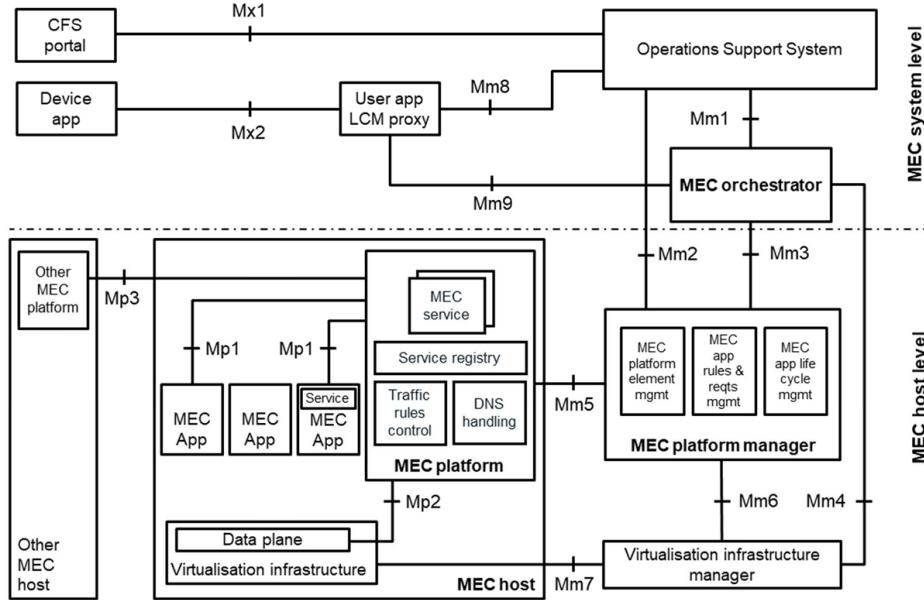


FIGURE 2.2: Multi-access Edge System Reference Architecture [33].

the two [43]. However, whether “Fog”, “Edge”, “MEC”, “Mist” or “Dew”, they share the same objective: to enable low-latency computation capabilities for time-sensitive tasks, which cannot tolerate the round-trip delay to the centralized Cloud.

In this dissertation, the term “Edge” refers to the computing resources provided at the edge of the network, while end-user devices are referred to as “Things.” Specifically, “Edge” refers to the local network infrastructure connecting sensors and IoT devices, which is a particularly common setup in industrial use cases. In other words, the Edge represents the immediate first hop from IoT devices, such as Wi-Fi access points or gateways, rather than the IoT nodes themselves.

### 2.1.3 The Integration of Cloud and Edge

With the emergence of Edge Computing, the idea of integrating Edge and Cloud Computing began to take shape. In 2009, Satyanarayanan et al. [44] introduced *cloudlets*, which address resource limitations of mobile devices by offloading computation to nearby resource-rich nodes. The concept was formalized several years later, when the authors introduced a two-tier architecture [45]. The first tier is known as the Cloud (high latency), and the second as cloudlets (lower latency). The latter are widely distributed Internet infrastructure components that provide compute cycles and storage resources to nearby mobile devices, maintaining only soft state such as cached copies of data.

By the 2010s, the integration of Cloud and Edge computing became more pronounced, especially with the rise of IoT [46, 47, 48, 49, 50]. A single Edge node cannot match the computational capacity of the Cloud. Consequently, cloud and edge are recognized as complementary rather than competing paradigms. The combination of Cloud’s vast storage and processing capabilities with Edge’s low-latency, real-time processing became essential for large-scale IoT applications. This integration allowed for a more efficient and scalable approach to handle the massive data generated by IoT devices [51, 52], so that latency-critical data are

processed locally at the Edge, while the Cloud is used for long-term analysis and storage, with appropriate data-privacy controls [53, 54, 55]. Several concepts have been proposed, such as *Osmotic Computing* [48, 56, 57], which describes a system with the capability to dynamically balance and distribute workloads across Cloud, Edge, and IoT environments to optimize performance and efficiency. *Fluid Computing* [58] is another approach that unifies computing capabilities across Cloud, Fog, and Edge into a single abstraction through provisioning of an end-to-end cooperative platform, which can be used for seamless computing, storage and networking. Thus, it allows application data to flow between components regardless of resource location.

Over the years, several synonymous terms have been proposed, such as *Cloud-Edge Continuum* [59, 60, 54, 61], *Cloud-to-Things Continuum* [62, 63, 64], *Cloud-to-Edge Continuum* [65, 66, 67], *IoT-Edge-Cloud Computing* [68, 69, 70, 71], *IoT-Fog-Cloud* [72], *Transcontinuum* [73], *Distributed Computing Continuum Systems* [53, 74, 75, 76]. They collectively represent a shift from siloed systems to unified and dynamic infrastructures. Designing and engineering systems that can fully utilize this powerful yet complex paradigm in industrial contexts represents the central challenge addressed in this dissertation.

## 2.2 The Industrial Revolutions: From Industry 1.0 to Industry 4.0

To date, three major industrial revolutions have fundamentally transformed both production methods and society. As early as the mid-18th century, the *First Industrial Revolution* began in Great Britain, crossed Europe's borders, and spread across the Atlantic to North America. During this time, the invention of the steam engine led to radical changes: the gradual adoption of machinery in production operations and the emergence of factory systems [77, 78]. The *Second Industrial Revolution* followed, lasting until the First World War, characterized by the adoption of electricity to enable mass production and the assembly line [79]. The result was a significant boost in productivity and reduction in costs. Many researchers agree that the contributions from these two revolutions played an important role in enabling the rapid transition to modern society by increasing the number of consumer goods in the market and improving living standards [80].

A few decades ago, the Internet and the advancement of digital technology paved the way for the *Third Industrial Revolution*, commonly known as the *Digital Revolution* [81, 82]. From the late 20th century until now, an increasing number of processes in factories have been automated through computers and programmable logic controllers (PLCs). This created the foundation for the next development stages across various industries, as information technology was increasingly used to extend manufacturing systems' capabilities.

The term *Industry 4.0* (or *Industrie 4.0* in the original German form) was first introduced at the Hannover Fair in Germany in 2011 [83] as the name for the planned *Fourth Industrial Revolution* in the *High-Tech Strategy 2020 Action Plan* of the German Federal Government [84, 85, 86]. Unlike previous revolutions, the foundation of *The Fourth Industrial Revolution* is the close integration of software and hardware. This

integration promises to be the next developmental stage in the organization and management of the entire manufacturing value chain. The transformation is driven by several key technological enablers:

- *Cyber-Physical Systems (CPS)*: These are systems where computational elements are tightly integrated with physical machinery. In a smart factory, every machine, robot, and conveyor belt becomes a CPS, capable of sensing its environment, processing data locally, and communicating with other systems [87, 88, 85, 89]. These systems enable a feedback loop spanning virtual and physical domains, through which computers control the physical processes and vice versa [90, 91, 92].
- The *Industrial Internet of Things (IIoT)*: The proliferation of low-cost sensors and ubiquitous connectivity allows for the collection of vast amounts of granular, real-time data from every stage of the production and is referred to as one of the main factors that could revolutionize the future of manufacturing [93].
- *Digital Twins*: An integrated multi-physics, multi-scale, probabilistic simulation representation of a physical asset or process [94, 95]. It is continuously updated with real-time data from its physical counterpart, enabling real-time monitoring, data-driven decisions, and simulation or optimization of various operating scenarios [96].
- *IT/OT Convergence*: Industry 4.0 breaks down the traditional division between Information Technology (IT) and Operational Technology (OT). Data from the factory floor is now directly used by enterprise-level analytics and business systems, and vice versa, offering opportunities to redesign industrial processes [97, 98].

Together, these technological enablers differentiate Industry 4.0 from previous revolutions: it exceeds incremental improvements, enabling a major shift toward fully connected, adaptive, and intelligent manufacturing processes.

## 2.3 Microservices as a Key Enabler for Cloud-Native Systems

The integration between OT and IT requires an alternative approach to traditional software design. The limitations of monolithic systems and the need for a dynamic distribution of workloads across heterogeneous environments, from the Cloud to the Edge, demand a new set of tools and principles.

Among various approaches, *Cloud-native technologies* provide an effective toolkit. Their primary goal is to build “loosely coupled systems that are resilient, manageable, and observable,” as defined by the Cloud Native Computing Foundation (CNCF) [99]. The core idea is to decompose applications into services and package them in lightweight containers to be deployed and orchestrated across a variety of servers [100]. As a result, Cloud Native Applications (CNAs) are designed to run on automated platforms, utilizing *softwarization* of infrastructure and networks. They emphasize interoperability across multi- and hybrid-cloud environments, ensuring portability

across different platforms [101]. By design, development practices, and deployment models, these applications provide horizontal scalability, adherence to service-level agreements (SLAs), and resilience to failures.

One of the most important cloud-native concepts used in this dissertation is the *Microservices Architectural Style*. According to Fowler and Lewis, a microservice-based application is a single application composed of a set of services [102]. These services are not only small but also highly decoupled, independently replaceable, upgradable, and deployable [103], and designed following the well-known UNIX principle “Do one thing and do it well”<sup>1</sup> [104, 105, 106]. Although these services are isolated from each other, i.e., each service has its own process and its own data management mechanism, there is extensive communication flow between them. These interactions are not performed using internal calls, but rather through clearly defined interfaces, for example, a hypertext transfer protocol (HTTP) resource API (Application Programming Interface) [102]. Clients interact with microservice-based systems via standardized protocols, such as REST (Representational State Transfer) over HTTP or SOAP (Simple Object Access Protocol) over HTTP.

Other researchers frame microservices in relation to *service-oriented architecture* [107], which is essentially a collection of services with well-defined interfaces and can be orchestrated to perform a task, to define this new concept. For instance, Fowler and Savchenko et al. describe microservices as “service-oriented architecture done right” [102, 108]. This definition is close to those of Sebastian Łaskawiec [109] and Cockcroft, A. [110], who have defined microservices as an architecture that is “mainly inspired by SOA services and the single responsibility principle” and as “a loosely coupled service-oriented architecture with bounded contexts” respectively.

Microservices are typically defined by a set of core characteristics. They are composed of *small, deployable, and independent services that interact via interfaces and protocols*, supporting rapid and continuous updates aligned with modern practices like CI/CD (Continuous Integration and Continuous Delivery/Deployment) [111].

Another defining aspect of microservices is their *focus on business capabilities*. Each microservice reflects a specific part of the business model, ensuring that the software architecture mirrors organizational needs. Teams are often structured around these services, taking ownership of complete technology stacks tailored to particular business areas [103].

Communication among microservices focuses on *message exchange* rather than complex functions like those in enterprise service buses [112], with business logic handled within each microservice [103].

Lastly, microservices employ *decentralization in governance and data management*, allowing developers to choose tools and manage services independently while using language-neutral APIs (gRPC/Protobuf, REST/JSON). Data management decentralization includes both conceptual model decentralization (services using different attributes of the same entities) and storage-backend decentralization (each service maintaining its own isolated database) [103].

---

<sup>1</sup>Doug McIlroy, the inventor of UNIX pipes and its design concept of software component, has summarized this philosophy as “Write programs that do one thing and do it well.”[104]

## **2.4 Chapter Summary**

In this second chapter, the dissertation presented an overview of the evolution of computing models, illustrating how they have gradually converged into a unified concept referred to as the Compute Continuum.

Parallel to this development, the industrial manufacturing sector has also undergone major transformations, reflected in the four industrial revolutions that have taken place to date. Over time, developments in computer science and manufacturing have increasingly intersected, creating a growing overlap between the two fields. Today, their progress is more closely connected than ever before. As productivity and efficiency remain central goals across industries, this convergence has become a key driver of modern industrial innovation.

This IT/OT convergence brings together diverse technologies to function as a unified system, reducing errors and improving workflows. It allows data from physical operations (OT) to be rapidly analyzed by IT systems, enabling more informed decision-making and autonomous operations that improve both accuracy and uptime.

To support the development of the Compute Continuum, many existing technologies originating in Cloud computing have been adopted, collectively referred to as cloud-native technologies. Among them, the most fundamental concept is the Microservices Architectural Style, representing a new approach to software design. The three core characteristics of microservices identified in this chapter, which are business capability alignment, communication-focused design, and decentralization, will be further discussed in the following chapters of this dissertation.

Before that, to survey the current research landscape of the Compute Continuum, a systematic study is conducted to organize and classify the existing body of research in this field. That literature review is presented and discussed in the next part of this dissertation. In Part II, Chapter 3 outlines the research methodology, while Chapter 4 presents the reviewed literature in detail. Finally, Chapter 5 discusses the key takeaways from the taxonomy development process.

## Part II

# Mapping the Landscape: A Taxonomy of the Compute Continuum



# 3 Research Design

The research on the CC and its components (Cloud, Fog, and Edge computing) has grown significantly in recent years. Various aspects have been discussed in the literature, such as architectures, distributed data management, and resource orchestration across heterogeneous environments, with applications ranging from healthcare to smart cities. However, this rapid development has also led to several issues, some of which were already discussed in the first part of this dissertation.

1. **Fragmented Research Landscape:** Similar to other growing domains in computer science, the development of the CC has led to a fragmented body of research. This fragmentation makes it difficult to build an overall understanding of the field.
2. **Inconsistent Terminologies and Definitions:** The lack of standardized terminologies and definitions within the CC poses a significant barrier to effective collaboration. Terms such as *cloud-edge continuum*, *edge-cloud continuum*, or *IoT-edge-cloud computing continuum* are often used interchangeably or inconsistently, leading to conceptual ambiguity in the research community.
3. **Unclear Research Trends:** While the field is evolving, systematic analyses of the main trends in CC research remain limited. Such analyses are essential to understand the current research dynamics and predict future developments.
4. **Limited Mapping of Research Gaps:** While individual studies identify specific challenges, there is still a need for more systematic mapping of major research gaps within the CC to better guide future research efforts.

Motivated by these limitations, this examines the CC research landscape by mapping existing studies, clarifying terminology, analyzing emerging trends, and highlighting research gaps. The resulting taxonomy is expected to serve as a foundational framework for organizing knowledge in the field, helping to navigate existing work and identify directions for future research.

## 3.1 Research Questions and Objectives

Based on the aforementioned motivations, the primary research question of this study is:

*What are the current research directions, emerging trends, and key terminologies associated with the Compute Continuum, and how active is each research area? What significant research gaps exist within this field?*

This main question can be decomposed into five research objectives, each focusing on a key aspect of the study.

1. **RO1: Map Research Directions and Activity Levels:** To identify the main research topics within the CC and measure their activity levels, therefore creating a clear map of the current research landscape.
2. **RO2: Clarify Terminologies and Formulate a Definition:** To examine various terms used to describe the CC and, from this analysis, synthesize a definition.
3. **RO3: Analyze Underlying Trends:** To identify and analyze the key underlying trends shaping the development and adoption of the CC.
4. **RO4: Identify Enabled Values:** To identify and describe the new technical, operational, and application-level values and opportunities enabled by the CC.
5. **RO5: Identify Research Gaps:** To systematically identify under-researched areas within the CC, providing directions for future research efforts.

By addressing these objectives, the taxonomy not only clarifies the conceptual understanding of the field but also provides a structured overview of current research on the Compute Continuum.

## 3.2 Related Work

Over the years, several studies have attempted to provide a more systematic view of the continuum, ranging from high-level surveys to detailed analyses of specific enabling technologies.

For example, exploring the support technologies for the CC, Bendechache et al. [113] focus on simulation tools and their role in evaluating resource management mechanisms. While widely used tools like CloudSim are effective for Cloud computing, the paper highlights the growing need for simulators tailored to Fog and Edge computing, such as iFogSim and FogNetSim++, which better simulate latency-sensitive applications and geographically distributed networks. This work presents a systematic review of 35 articles published between January 2015 and March 2019.

In a different direction, Moreschini et al. [114] seek to standardize the definition of the continuum by reviewing 36 studies (from 2016 to 2022) and synthesizing varied definitions across the literature. Their analysis reveals three main themes: (1) a resource-centric view extending from Edge to Cloud, (2) distributed processing across the continuum, and (3) multi-infrastructure perspectives incorporating IoT and micro-data centers. The study highlights challenges in dynamic orchestration, context-aware solutions, and security, emphasizing the need for abstraction models that support adaptable and resilient systems across the continuum.

There are also works that focus on the enabling technologies required to implement the CC. *Distributed Intelligence* is one such technology, as presented by Rosendo et al. [115]. Their work examines how Edge devices handle low-latency, privacy-sensitive tasks, while Cloud resources are used for large-scale data analytics. Their main contribution is a taxonomy of libraries, frameworks, and machine learning paradigms

enabling distributed intelligence across Edge–Cloud infrastructures. Reviewing 69 papers (2016–2021), the authors identify challenges in deployment on heterogeneous infrastructures, and system performance analysis across the continuum.

Among the various approaches to distributed intelligence, Federated Learning (FL) has gained traction due to its intrinsic privacy-preserving properties, suitable for privacy- and latency-sensitive applications. Mishra et al. [116] review 45 studies (2010–2023), addressing FL architectures across domains and highlighting challenges like data heterogeneity and Edge resource constraints. They propose scalable, context-aware mechanisms, hybrid Edge–Cloud training, and secure aggregation protocols to address these issues and maintain model accuracy. In another study, Prigent et al. [117] analyze 53 studies and 20 open-source FL tools (2018–2023), identifying challenges in applying FL across heterogeneous infrastructures. Key identified issues include statistical heterogeneity, system volatility, and security risks like model inversion and poisoning attacks. To mitigate these issues, the authors discuss strategies such as personalized FL, client selection, differential privacy, and multi-hop communication. The authors conclude that future FL systems must be adaptive and interoperable to function reliably across heterogeneous environments.

Another reviewed aspect is the service execution model. Oliveira et al. [118] systematically map Function-as-a-Service (FaaS) platforms (a subset of Serverless Computing discussed in Chapter 2) in the CC, analyzing 33 studies (2017–2022) to assess how FaaS addresses latency, resource constraints, and network overhead. They evaluate deployment, function placement, orchestration, and execution strategies, proposing a taxonomy of FaaS capabilities across Cloud, Edge, and IoT environments. The study identifies several research gaps, including the lack of adaptive, context-aware function allocation strategies suited for heterogeneous and resource-constrained environments. They recommend developing adaptive placement strategies, improved integration across layers of the continuum, and more real-world testing to enhance performance of FaaS deployments.

Another related work is the study by Gkonis et al. [119], which presents an extensive review of IoT, Edge, and Cloud (IEC) integration, highlighting challenges such as device heterogeneity, latency constraints, and distributed security. In the study, scalability is identified as a key concern due to the rapidly growing number of IoT devices, while resource management remains complex across multi-layer environments. The authors also discuss supporting technologies such as federated learning, serverless computing, and blockchain, which can help achieve secure and low-latency processing at scale. Based on 35 publications (2016–2023), the study concludes that IEC systems are foundational for real-time, resource-efficient applications.

S-Julián et al. [120] review self-\* capabilities (self-configuration, self-healing, self-awareness, self-scaling, and self-optimization), which are essential for autonomous operations across the CC. Analyzing 77 studies (2015–2023), the authors show that autonomy has become essential in IoT systems, where manual management is no longer feasible. The paper proposes a taxonomy categorizing these self-\* capabilities, emphasizing their role in dynamic adaptation, resilience, workload scaling, and performance optimization. Key challenges identified include managing heterogeneity across nodes and the need for unified frameworks to

enhance interoperability in autonomous distributed systems.

Last but not least, Gaglianese et al. [121] review green orchestration for containerized applications in the CC. Analyzing 36 studies (2015–2022), they categorize orchestration approaches into energy-aware, green multi-criteria, and energy-optimal methods, highlighting trade-offs between energy efficiency, performance, and adaptability. The review distinguishes between stateless and stateful orchestration and examines techniques ranging from planning algorithms to ML-based methods. Key challenges include the absence of comprehensive sustainability metrics and limited adaptability to changing cloud-edge conditions. The authors call for adaptive, QoS-aware orchestration, standardized benchmarks, and open-source tools to advance sustainable continuum management.

TABLE 3.1: Comparison of Related Surveys on Compute Continuum.

Publication	Year	Primary contribution	Publications reviewed	Time range
Bendechache et al. [113]	2020	Simulation tools in continuum	35	2015–2019
Moreschini et al. [114]	2022	Definition of the continuum	36	2016–2022
Rosendo et al. [115]	2022	Machine Learning and Data Analytics across CC	69	2016–2021
Oliveira et al. [118]	2023	Application of FaaS across CC	33	2017–2022
Gkonis et al. [119]	2023	Integration of computing models	35	2016–2023
S-Julián et al. [120]	2023	Autonomous operations of cloud-edge nodes	77	2015–2023
Gaglianese et al. [121]	2023	Sustainable orchestration of services	36	2015–2022
Mishra et al. [116]	2024	Federated Learning across CC	45	2010–2023
Prigent et al. [117]	2024	Integration of Federated Learning in the CC	53	2018–2023

The reviewed studies, summarized in Table 3.1, reinforce the need for a new taxonomy of the CC as previously motivated in this dissertation. The terminologies used across these studies vary significantly. For instance, Moreschini et al. [114] use the term *Cloud Continuum*, whereas Rosendo et al. [115] describe it as the *Edge-to-Cloud Continuum*. In addition, existing reviews often focus on narrow topics, such as specific technologies like Federated Learning [116]. Such narrow focus leaves

broader research trends underexplored and makes it difficult to gain an overall perspective.

### 3.3 Systematic Review Methodology

This research follows a Systematic Literature Review (SLR) methodology, conducted in line with the 2020 PRISMA (Preferred Reporting Items for Systematic Reviews and Meta-Analyses) guidelines [12]. The PRISMA framework provides a structured procedure that promotes transparency and reproducibility throughout the review process. A two-phase approach was used, illustrated in Figure 3.1.

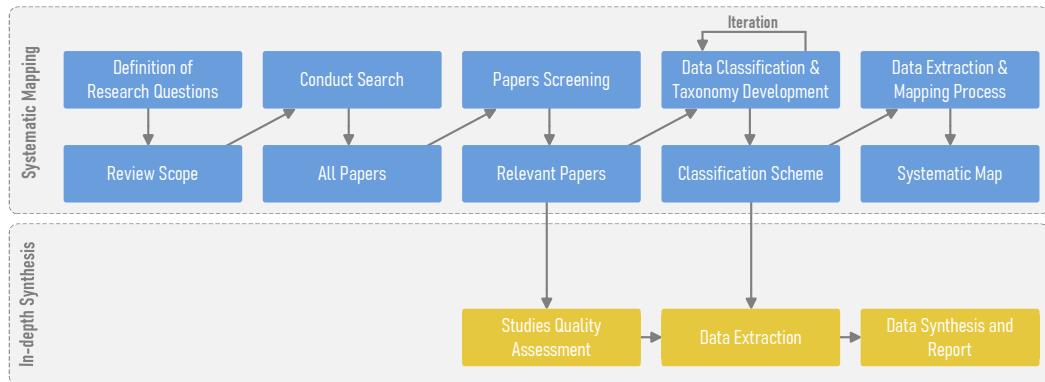


FIGURE 3.1: Two-phase approach used in this study.

#### 3.3.1 Data Sources

To ensure broad coverage of relevant publications, this study has used Scopus, one of the most extensive databases in computer science, which indexes approximately 75.86% of research articles [122, 123, 124, 125]. Furthermore, previous studies have shown that Scopus provides high-quality bibliographic records with better granularity compared to more general databases due to sophisticated filtering system [126]. To further broaden coverage and minimize the risk of missing relevant studies, this work also considered IEEE Xplore and the ACM Digital Library. This practice is consistent with previous SLRs; for example, one study shows that combining Scopus with the ACM Digital Library achieves a coverage rate of up to 96.5% of relevant papers [126].

#### 3.3.2 Search Queries

The search queries included seven key terms: *Cloud-to-Things*, *Edge-Cloud Continuum*, *Cloud-Edge Continuum*, *IoT-Edge-Cloud*, *IoT-Fog-Cloud*, *Compute Continuum*, and *Computing Continuum*. These terms were searched within the title, abstract, or keywords, as indexed by the databases. The search focused on papers published between January 1, 2021, and June 30, 2024, as these years reflect a significant increase in research activity as shown in Figure 3.2. The queries used for each library are documented in Table 3.3.

TABLE 3.3: Search Queries Used for the Systematic Literature Review.

Database	Query
<b>SCOPUS (576 entries found)</b>	(( TITLE-ABS-KEY ( "Cloud-to-Things" ) OR TITLE-ABS-KEY ( "Edge-Cloud continuum" ) OR TITLE-ABS-KEY ( "Cloud-Edge continuum" ) OR TITLE-ABS-KEY ( "IoT-Edge-Cloud" ) OR TITLE-ABS-KEY ( "compute continuum" ) OR TITLE-ABS-KEY ( "computing continuum" )) AND PUBYEAR > 2020 AND ( LIMIT-TO ( DOCTYPE , "ar" ) OR LIMIT-TO ( DOCTYPE , "cp" ) ) AND ( LIMIT-TO ( SUBJAREA , "COMP" )))
<b>IEEE (289 entries found)</b>	"Document Title": "Cloud-to-Things" OR "Document Title": "Edge-Cloud continuum" OR "Document Title": "Cloud-Edge continuum" OR "Document Title": "IoT-Edge-Cloud" OR "Document Title": "IoT-Fog-Cloud" OR "Document Title": "compute continuum" OR "Document Title": "computing continuum" OR "Abstract": "Cloud-to-Things" OR "Abstract": "Edge-Cloud continuum" OR "Abstract": "Cloud-Edge continuum" OR "Abstract": "IoT-Edge-Cloud" OR "Abstract": "IoT-Fog-Cloud" OR "Abstract": "compute continuum" OR "Abstract": "computing continuum" OR "Author Keywords": "Cloud-to-Things" OR "Author Keywords": "Edge-Cloud continuum" OR "Author Keywords": "Cloud-Edge continuum" OR "Author Keywords": "IoT-Edge-Cloud" OR "Author Keywords": "IoT-Fog-Cloud" OR "Author Keywords": "compute continuum" OR "Author Keywords": "computing continuum"
<b>ACM (24 entries found)</b>	[[Title: "cloud-to-things"] OR [Title: "edge-cloud continuum"] OR [Title: "cloud-edge continuum"] OR [Title: "iot-edge-cloud"] OR [Title: "iot-fog-cloud"] OR [Title: "compute continuum"] OR [Title: "computing continuum"]] AND [[Abstract: "cloud-to-things"] OR [Abstract: "edge-cloud continuum"] OR [Abstract: "cloud-edge continuum"] OR [Abstract: "iot-edge-cloud"] OR [Abstract: "iot-fog-cloud"] OR [Abstract: "compute continuum"] OR [Abstract: "computing continuum"]] AND [[Keywords: "cloud-to-things"] OR [Keywords: "edge-cloud continuum"] OR [Keywords: "cloud-edge continuum"] OR [Keywords: "iot-edge-cloud"] OR [Keywords: "iot-fog-cloud"] OR [Keywords: "compute continuum"] OR [Keywords: "computing continuum"]]] AND [E-Publication Date: (01/01/2021 TO 30/06/2024)]

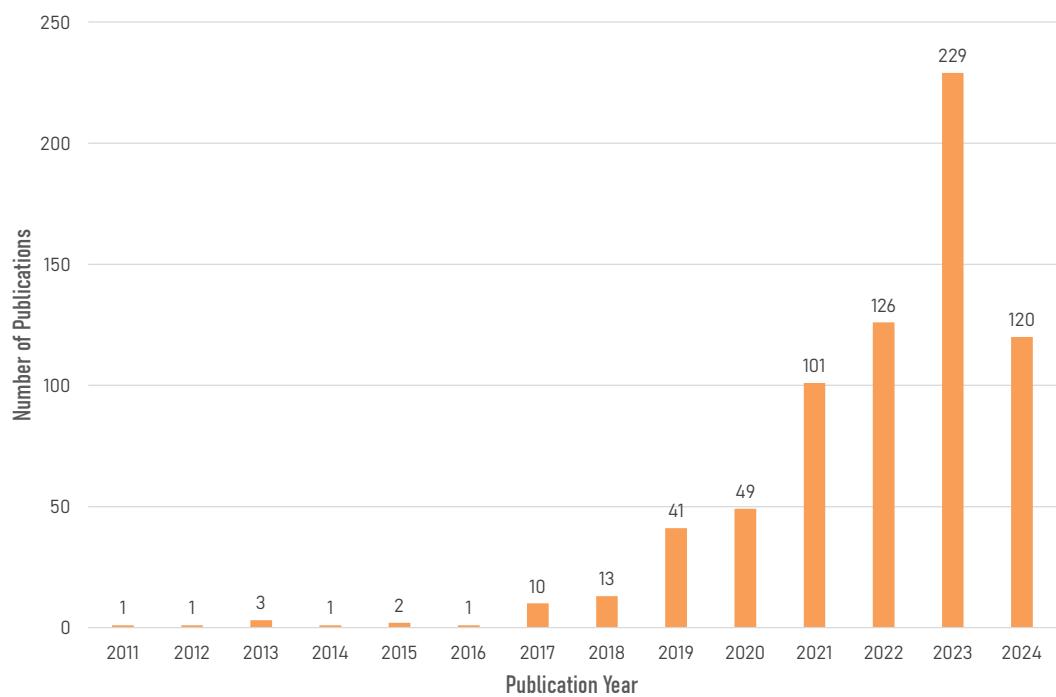


FIGURE 3.2: Number of Publications by Year, searched on Scopus using the query shown in Table 3.3.

### 3.3.3 Eligibility Criteria

#### Inclusion Criteria

Eligible publications included peer-reviewed journal articles, conference papers, or book chapters in the field of computer science. In addition, only studies written in English and published between January 2021 and June 2024 were considered.

### Exclusion Criteria

Papers were excluded if they met any of the following criteria:

- *Synthesized research*: Exclude literature reviews, meta-analyses, or other synthesized research studies, for example [127, 119].
- *Collections of studies*: Exclude items that are collections of papers, such as proceedings or books without original research contributions.
- *Short papers*: Exclude short papers (less than 6 pages), extended abstracts, or brief communications that do not provide detailed findings.
- *Demo papers*: Exclude demo papers that do not present complete research results.
- *Abstract-only papers*: Exclude papers where only the abstract is available without access to the full text.
- *Poster presentations*: Exclude poster presentations as they do not provide full research details.
- *Papers from unrelated fields*: Exclude research that is outside the scope of computer science or does not directly relate to the research focus.
- *Tangentially related studies*: Exclude papers that address general topics (e.g., general Cloud computing) without directly focusing on the CC or related aspects. Examples include works on Edge computing in traffic processing [128] and digital twins in high-performance Edge environments [129].
- *Low-quality research*: Exclude studies with unclear methodology, insufficient evidence, or incomplete analysis.
- *Unpopular terms*: Exclude studies that use unpopular terms for the CC. There are some related terms but not widely used, such as *Cloud-Fog-IoT* [130], *Cloud Continuum* [131], *Cloud Computing Continuum* [132], *Cloud-Fog-Continuum* [133], *Cloud-Fog-IoT* [130], *Device-Edge-Cloud* [134], *Edge-Fog-Cloud* [135], *Things-to-Cloud computing continuum* [136] or *Distributed Computing Continuum Systems*, a term used exclusively by Distributed Systems Group, Technical University Vienna, Austria (TU Wien) [53, 74, 75, 76], or *Cloud-Edge-IoT Continuum* used by the EUCloudEdgeIoT initiative [137]. Another example is *Transcontinuum*, which is mentioned solely by KerData, INRIA, France [115].
- *Non-English papers*: Exclude papers written in languages other than English.

By applying these exclusion criteria, this study selects only high-quality, accessible, and relevant research to be included in the review, maintaining the focus and the quality of the findings.

#### 3.3.4 Selection Process

The study selection followed the PRISMA 2020 guidelines as summarized in Figure 3.3), detailing the number of records identified, screened, excluded, and included, as well as the reasons for exclusion at each stage. After applying the eligibility criteria, a

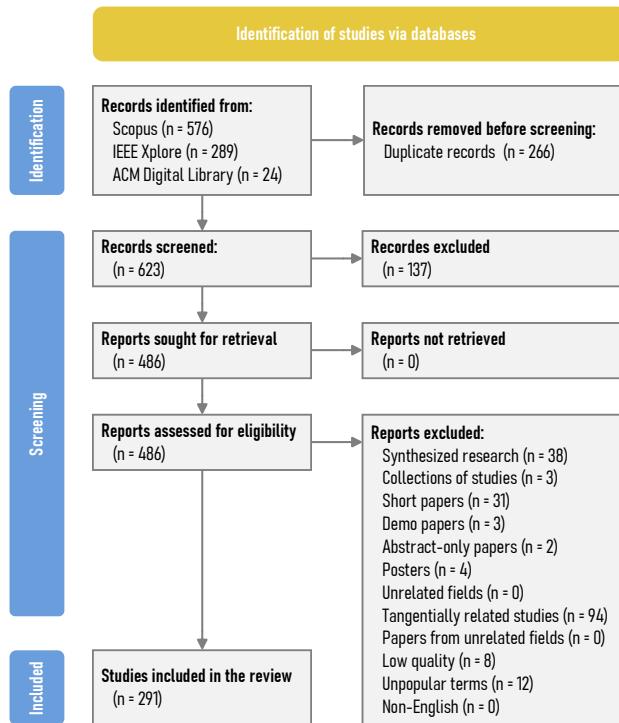


FIGURE 3.3: PRISMA Flow Diagram (created for this study based on [12]).

total of 291 papers remained in the final dataset. Specifically, 281 papers were collected from Scopus, 8 papers were sourced from IEEE Xplore, and 2 papers were selected from ACM Digital Library.

### 3.3.5 Threats to Validity

In this research, validity threats as outlined by Wohlin's guidelines [138] can influence various aspects of the taxonomy development process. This section discusses these threats and their corresponding mitigation strategies.

#### Internal Validity

Internal validity concerns whether the taxonomy accurately reflects the true structure and relationships within the reviewed literature. In this specific study, two main threats are *misinterpreting individual papers* and *overlooking key subdomains*.

*Threat 1: Misclassification of Publications:* Given the nature of the CC, studies often span multiple computational domains. A key risk is misclassifying a paper's core contribution, such as categorizing a paper on Edge-native orchestration under general Cloud computing. Another scenario is a paper that mentions the CC but primarily addresses Edge computing.

*Mitigation:* To address this risk, this study adopted an iterative classification process as illustrated in Figure 3.1. The initial round involved creating a draft classification system based on paper titles, abstracts, and keywords. This was then followed by a full-text review to ensure the final categorization correctly identifies the primary contributions.

*Threat 2: Overlooking key subdomains:* A fundamental issue with SLRs based on database searches is the risk of missing studies that use different terminologies. To some extent, this has already been demonstrated in the reviewed related work, with different terms such as “cloud continuum” [114] and “cloud-edge” [120]. This risk is significant, as it could result in an incomplete taxonomy that fails to represent the true research landscape.

*Mitigation:* To address this risk, this study adopted a multi-database approach. The inclusion and exclusion criteria, as well as the queries, were iteratively refined. The goal was to achieve a balance between fragmented and overly coarse classifications.

## External Validity

In this study, external validity refers to the extent to which the findings can be generalized beyond the specific context of CC. This study identifies three main limitations of the search process. Firstly, the dependency on specific keywords means that studies discussing relevant concepts without using the exact terms as the queries presented in Table 3.3 may have been missed. Secondly, although the combination of research databases improves coverage, relying solely on these online databases may have excluded relevant papers from other sources or grey literature (e.g., technical reports, non peer-reviewed literature). Last but not least, by restricting the search to the years from 2021 to 2024, the taxonomy may overlook foundational older works as well as more recent advancements that have just been published.

*Mitigation:* To address the limitations of keyword dependency, the search queries were refined iteratively, as illustrated in Figure 3.1. In each iteration, new relevant terms were added, and unrelated ones were removed. This process resulted in the seven key terms that offered the best balance of coverage and accuracy. Although the use of specific databases and a defined time frame may have excluded some studies, the transparent and structured nature of the process enhances the generalizability of the findings within the defined scope.

## Construct Validity

Construct validity concerns the extent to which the taxonomy accurately represents the theoretical structures it aims to classify. A potential threat arises when the categories do not clearly distinguish between different conceptual domains, leading to overlap or ambiguity.

*Mitigation:* This threat was mitigated by grounding the taxonomy in prior, well-established research. Rather than creating the classification framework from scratch, the well-known taxonomy for Cloud Computing proposed by Yang and Tate [139] was adopted as a baseline, given its conceptual similarity to the Compute Continuum. From this foundation, categories and subcategories were defined to maintain consistency and avoid overlap.

### Conclusion Validity

Conclusion validity concerns the soundness and justification of the inferences drawn from the classified data. This is closely related to the completeness and quality of the selected studies.

*Mitigation:* This threat was addressed by applying a systematic and iterative selection procedure based on clearly defined inclusion and exclusion criteria, following PRISMA 2020 guidelines. The structured selection process minimizes potential selection bias and strengthens the derived conclusions.

## 3.4 Chapter Summary

This chapter began by outlining the main motivations for the development of a new taxonomy for the CC. In response to these challenges, the study defined five key objectives (to map research directions and activity levels, to clarify terminologies and formulate a definition, to analyze underlying trends, to identify enabled values, and to identify research gaps). These motivations were further reinforced by reviewing existing systematic literature reviews relevant to the Compute Continuum.

Subsequently, the chapter presented details of the research design, including the data sources, search queries, and the inclusion and exclusion criteria applied. Several threats to validity are also discussed, along with the mitigation strategies used to address them.

By clearly outlining the research process in accordance with the PRISMA 2020 guidelines, this study aimed to develop a comprehensive, up-to-date, and reproducible taxonomy for the Compute Continuum. The taxonomy, which was based on 291 selected publications, together with its subcategories and detailed discussion of the reviewed publications, will be presented in the following chapter.



# 4 A Taxonomy of Research on Compute Continuum

Developing a taxonomy for CC research is challenging because the field combines several computing models, each with distinct characteristics. A naïve categorization based solely on computation location (Edge, Cloud, or IoT devices) would oversimplify the CC's core principle of seamless integration and interaction across layers. Similarly, focusing on individual components or service models (IaaS, PaaS, SaaS, Serverless) fails to capture its unified nature.

However, since the CC builds on established computing models, existing taxonomies, especially those from Cloud computing, could serve as a useful foundation. The classification system in this dissertation was based on the well-known framework by Yang and Tate [139], which categorizes Cloud computing research into *Technological Issues*, *Business Issues*, *Domains and Applications*, and *Conceptualizing Cloud Computing*. While this provides a solid foundation, further refinements are required to adapt it to the new characteristics of the CC.

First, while Yang and Tate's taxonomy captures high-level challenges in Cloud computing, it lacks the granularity needed for the CC, particularly in addressing real-time processing, resource management, and orchestration across Cloud, Edge, and IoT environments. The taxonomy presented in this dissertation addresses these complexities through detailed subcategories, such as Operational Management, Optimization, and Resource Management, to cover challenges like task scheduling and resource allocation.

Second, although Yang and Tate's emphasis on *Business Issues* was critical during the early adoption of Cloud computing, as mentioned previously in the research scope, this dissertation focuses primarily on technical challenges. Although business aspects remain important, they appeared less frequently in the reviewed studies, which were collected from technical publication libraries. The taxonomy, therefore, emphasizes technical challenges but can be extended in future to incorporate business aspects.

Third, while Yang and Tate's category for *Conceptualizing Cloud Computing* is still relevant, the interconnected environments of the CC require a more structured and technically focused taxonomy. Accordingly, the proposed taxonomy replaces this broad category with a targeted review of architectures and designs specific to the CC, covering both general-purpose architectures and domain-specific designs.

In summary, the proposed taxonomy organizes the collected publications into three main categories:

- **Industry-specific Use Cases and Applications:** This category addresses the real-world deployments and implementations of the CC. Building on the *Domains and Applications* category from Tate and Yang's work, it focuses

on studies that explore how the CC addresses domain-specific challenges. Subcategories include applications in various industries such as healthcare, smart cities, transportation, agriculture, and manufacturing.

- **Engineering and Technical Solutions:** This category extends Tate and Yang's *Technological Issues* category to cover engineering approaches, methodologies, and tools for solving technical challenges such as resource management, optimization, task scheduling, and orchestration. It also contains technical solutions aim to provide scalability, performance, and reliability across the continuum's layers.
- **System Architectures and Designs:** This category addresses the foundational architectural and design principles required to develop and deploy systems within the CC. Inspired by the *Conceptualizing Cloud Computing* category, it includes research on designing distributed, scalable, and efficient systems that integrate Cloud, Edge, and IoT resources. Subcategories include both general-purpose and domain-specific architectures.

Rather than developing a new classification system from scratch, this study builds on the established taxonomy of Tate and Yang, thereby ensuring construct validity as discussed previously. The following sections present each main category and its corresponding subcategories in detail, together with brief summaries of the collected publications.

## 4.1 Industry-specific Use Cases and Applications

As mentioned earlier, this category includes publications that focus on the applicability of the CC in specific use cases. The overall structure of this category is illustrated in Figure 4.1.

### 4.1.1 Computing and Network Infrastructure

The studies in this category can be further divided into three subcategories: Data Analysis, Graph Processing, and Machine Learning/AI use cases.

For data analytics, the CC relies on dynamic policies to optimize workflows across distributed resources. Belcastro et al. [140] demonstrate an application in urban mobility, where an Edge orchestrator places analytics tasks on either Edge or Cloud infrastructure based on network and computational load to improve efficiency. Further expanding the concept, Balouek-Thomert et al. [141] propose a generalized, policy-driven adaptation framework that allows analytics pipelines to react to runtime changes in data size, bandwidth, and CPU capacity. This approach enforces user-defined objectives such as hard deadlines for urgent, time-critical applications such as disaster response.

The challenge of massive graph processing has also been addressed within the CC through new architectures focused on both scalability and sustainability. Farahani et al. [142] propose a serverless framework to orchestrate graph processing workloads across the continuum's heterogeneous resources by abstracting the underlying

infrastructure complexity. Complementing this approach, Iosup et al. [143] introduce the "Graph Greenifier," a specialized digital twin designed to provide sustainability analysis by modeling the energy consumption and environmental impact of these workloads, enabling evidence-based decisions for more energy-efficient graph processing.

Similar patterns were observed in machine learning research. For training, Ahmad and Aral [144] propose a hierarchical federated learning framework that leverages Edge nodes to create personalized models for client clusters while using the Cloud to enable global knowledge sharing. For reference, applications such as the real-time sports-injury prediction proposed by Wang et al. [145] argue for low-latency, Edge-based computation. However, deploying these models presents significant trade-offs. For example, Bueno et al. [146] show that while FaaS provides dynamic scaling, their resource overhead limits deployments on resource-constrained Edge devices. This performance challenge is quantified by Mittone et al. [147], who demonstrate that distributing inference can yield significant speedups over centralized models but is highly dependent on network bandwidth.

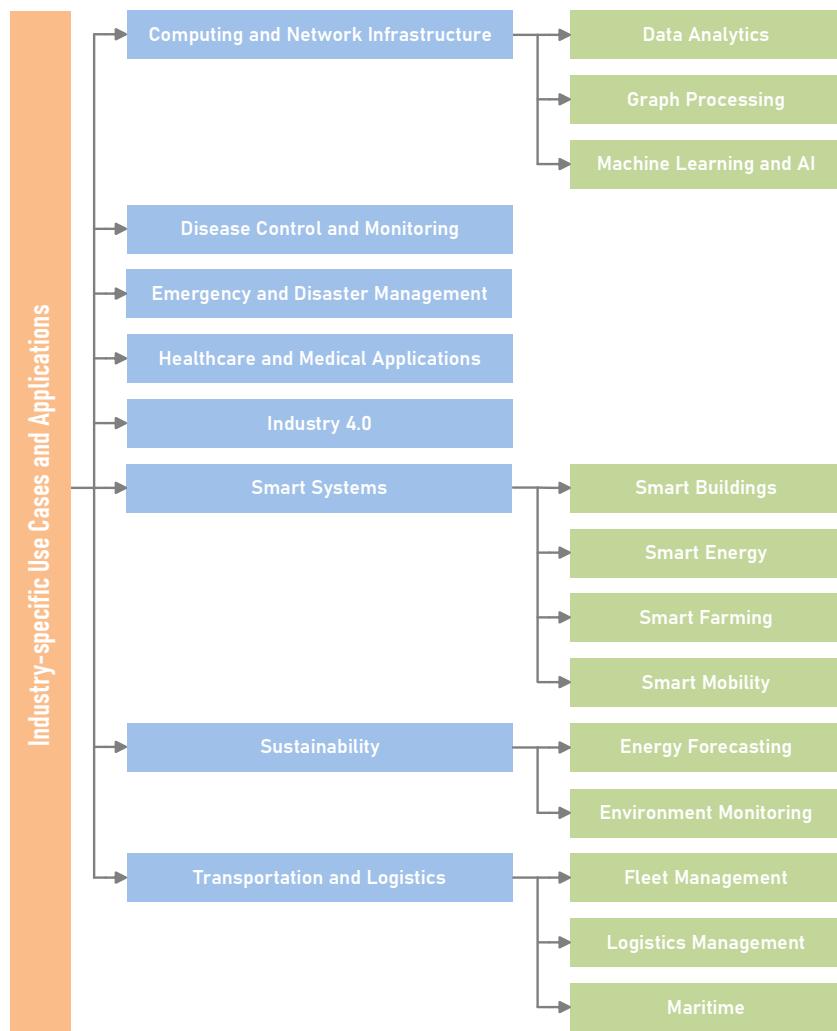


FIGURE 4.1: Industry-specific Use Cases and Applications.

#### 4.1.2 Disease Control and Monitoring

The CC can also play a key role in disease control and monitoring by enabling real-time and scalable systems. For instance, Alqahtani et al. [148] propose an AI-driven dengue control framework using Edge servers for rapid response and Cloud servers for comprehensive analysis. Similarly, Manoharan et al. [149] apply a Convolutional Neural Network - Time-aware Long Short-Term Memory (CNN-LSTM) model across the continuum to enable early diagnosis and prevent disease outbreaks.

#### 4.1.3 Healthcare and Medical Applications

The low-latency capabilities of the CC are also beneficial for healthcare scenarios such as real-time diagnostics, remote monitoring, and tele-rehabilitation. Pati et al.'s FRIEND framework [150] demonstrates this by reducing cardiac diagnosis latency via Edge processing while offloading heavy deep learning analytics to the Cloud. Similar Edge–Cloud integration enables tele-rehabilitation, as proposed by Ciraolo et al. [151]. Their work uses facial expression recognition to support personalized patient monitoring and reduce hospitalization. Wearable health applications also benefit from hybrid approaches, as shown by Xiaolin et al. [152], who integrate Edge-based anomaly detection with Cloud-based CNN analysis to minimize latency and energy use while maintaining accuracy. Beyond diagnostics, Kimovski et al. [153] demonstrate that decentralized electronic health records (EHRs) and distributed machine learning across the continuum can improve privacy and enable more adaptive, personalized care.

#### 4.1.4 Industry 4.0

In Industry 4.0, the CC decentralizes data processing by allowing IoT devices, gateways, and Edge servers to operate more autonomously, reducing dependence on centralized systems. The concept is illustrated by Lacalle et al. [154]. Their work demonstrates a self-\* framework that allows industrial nodes to handle self-awareness, self-orchestration, and self-healing, creating adaptive and efficient production lines. The authors argue that this is critical for *Lot-Size-One manufacturing*, where real-time adjustments allow factories to customize products to individual specifications without sacrificing the efficiency of mass production.

#### 4.1.5 Smart Systems

In smart agriculture, Sarantakos et al. [155] propose using Edge-Cloud computer vision combined with drones for early olive leaf disease detection, reducing manual inspections. Smart building management benefits from platforms such as COGITO [156], which regulates Heating, Ventilation, and Air Conditioning systems (HVAC) at the Edge while relying on the Cloud for long-term analysis. In the same context, Mishra et al. [157] highlight the continuum's potential for disaster response through a LiDAR-based system that enables rapid assessment of structural damage.

Energy systems also benefit from continuum integration. Arul et al. [158] optimize microgrid management in vehicular ad-hoc networks (VANETs), and Tzanis et

al. [159] enhance resilience and efficiency in smart grids through Edge–Cloud Phasor Data Concentrators (PDCs). In urban mobility, Belcastro et al. [160] present an Edge–Cloud architecture for traffic monitoring and route planning, balancing low-latency Edge tasks with Cloud-based predictions.

#### 4.1.6 Sustainability

The CC has also been increasingly applied to energy forecasting and environmental monitoring, utilizing hybrid Edge–Cloud intelligence for sustainability. Moussa et al. [161] develop a compute continuum framework for electrical energy forecasting, combining Edge-to-Cloud (E2C) centralized learning and Edge-to-Edge (E2E) federated learning to enhance smart grid efficiency and forecasting accuracy with LSTM and CNN-LSTM models. In renewable energy forecasting, Frincu et al. [162] propose a solar forecasting platform that executes rapid Edge adjustments while delegating complex analysis to the Cloud to optimize photovoltaic energy management. Similarly, Hong et al. [163] use a Bidirectional LSTM-based model in a distributed architecture to detect abnormal electricity consumption through Edge data collection with Cloud data processing. In another context, Fernández et al. [164] demonstrate a 26% latency reduction in indoor air quality prediction using an Edge–Cloud platform, while Wang et al. [165] apply a Fog–Cloud system for wildfire detection, coupling real-time Fog-based analysis with large-scale Cloud-based predictions.

#### 4.1.7 Transportation and Logistics

In transportation and logistics, the CC enables real-time data processing, low-latency communication, and scalability, which together improve overall operational efficiency. For example, Aljumah et al. [166] propose a Stochastic Game Network (SGN)-inspired framework using sensors and Fog computing for immediate decision-making and Cloud computing for long-term analysis. In maritime logistics, Charpentier et al. [167] demonstrate how 5G networks and Edge computing enhance vessel navigation and port operations by reducing latency and improving safety, as shown in the Port of Antwerp. Alsubai et al. [168] present a digital twin-inspired framework for electric vehicles (EVs) combining IoT, Fog, and Cloud computing to monitor battery health and charging efficiency. Data are processed at the Fog layer for immediate response, while the Cloud provides deeper analytics to optimize fleet management.

## 4.2 Engineering and Technical Solutions

Inspired by Yang and Tate’s *Technological Issues*, this second category includes publications that address technical aspects of the CC. However, the collected dataset demonstrates a much broader diversity of topics, ranging from optimization to data management. For that reason, this dissertation proposes a finer-grained classification system than the original framework by Yang and Tate. An overview of this classification is given in Figure 4.2, and the following sections discuss each subcategory in detail.

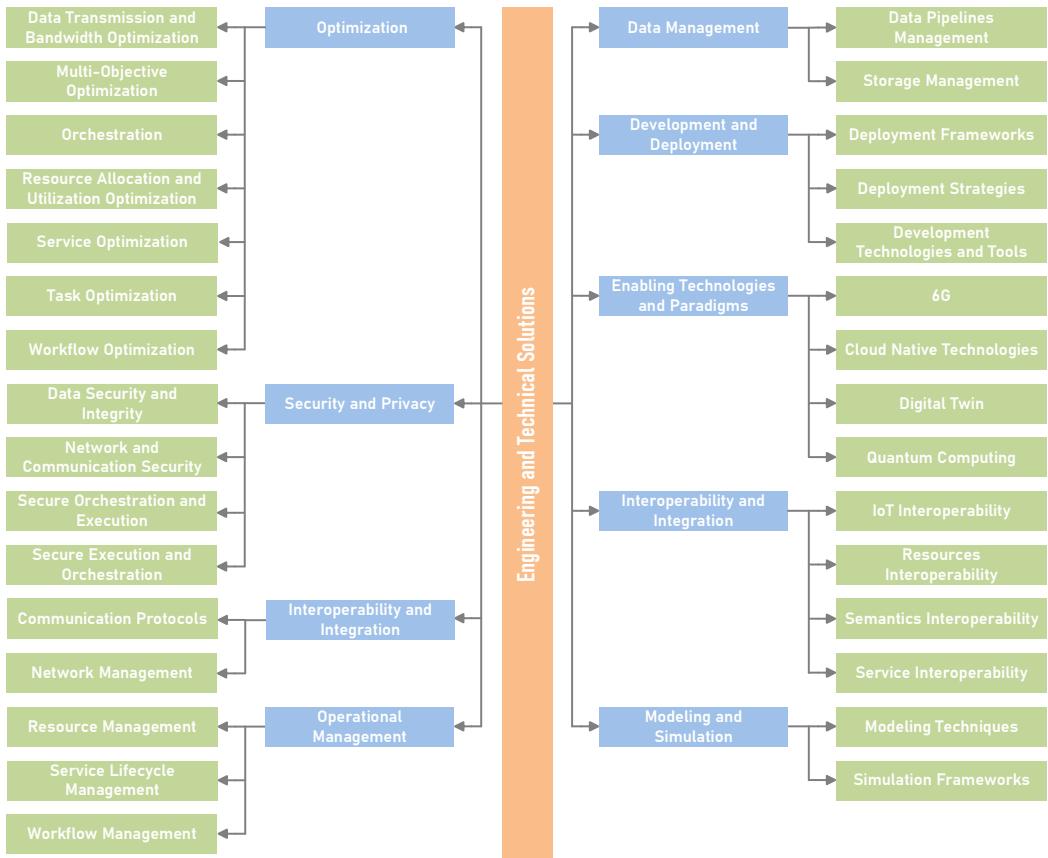


FIGURE 4.2: Engineering and Technical Solutions.

### 4.2.1 Data Management

The first subcategory of *Engineering and Technical Solutions* covers data-related topics. These studies can be further divided into two groups: *Data Pipeline Management*, which deals with dynamic data, and *Storage Management*, which focuses on static data.

#### Data Pipelines Management

Nikolov et al. [169] divide pipelines across Edge–Cloud layers using containerization to improve modularity and to secure data handling in constrained environments. Marcu and Bouvry [170] extend this idea by enabling direct data exchange between layers, which increases throughput in IoT and IIoT systems.

To optimize efficiency, Laso et al. [171] design an elastic framework that balances data quality, cost, and resource use for collaborative multi-stakeholder analytics. Roman et al. [172] address “dark data” close to the Edge to produce faster and more cost-effective insights, particularly in healthcare and industrial contexts.

Several approaches target specific workloads. For example, Afzal et al. [173] introduce the Multi-layered Pipeline Encoding and Compression Continuum (MPEC2) framework for video encoding, distributing compression tasks across Fog and Cloud layers. In telemedicine, a scheduling and adaptation framework [174] separates scheduling from execution to support real-time monitoring through adaptive resource provisioning. Similarly, Souza et al. [175] propose a distributed

analytics pipeline that dynamically allocates tasks across IoT, Edge, and Cloud layers to optimize latency, throughput, and resource utilization.

## Storage Management

**Data Caching** Zyrianoff et al. [176] introduce CACHE-IT, a distributed Edge-caching architecture that separates caching policies from system architecture. This separation allows the framework to support multiple deployment scenarios, including smart cities and structural-health-monitoring (SHM) applications. By predicting data needs, CACHE-IT minimizes redundant computation and data transfer, enabling near real-time operation with reduced dependence on Cloud resources. Extending this idea, Zyrianoff et al. [177] develop CACHUUM, a FL-based caching framework for privacy-sensitive Edge–Cloud environments. It processes data locally and transmits only model updates to the Cloud, preserving user privacy. CACHUUM integrates local, global, and federated caching strategies and, in simulation studies, achieves higher cache-hit ratios and better compliance with Age-of-Information (AoI) requirements.

**Data Placement** Sedlak et al. [178] define the term “data gravity” as the tendency of data to attract nearby applications, and “data friction” as the resistance encountered when moving data between layers. They propose using Markov SLO Configurations (MSCs) to dynamically adjust QoS levels and processing locality, achieving real-time responsiveness in IoT systems. Mellone et al. [179] develop a data-partitioning framework for large-scale environmental datasets deployed on Amazon Web Services and Microsoft Azure. Their approach allows selective data retrieval, which is critical for environmental monitoring and forecasting applications. For robotic and industrial scenarios, Bakhshi et al. [180] design a storage-placement strategy that distributes data across continuum layers using a Replicated Data Structure (RDS) and Storage Container (SC) governed by the Raft consensus protocol. This approach improves fault tolerance and provides consistent, low-latency access to time-sensitive information.

**Distributed Data Storage and Management** Martinez-Casanueva et al. [181] introduce CANDIL, a context-aware data fabric designed for network analytics. The system integrates a distributed Edge–Cloud architecture with a Semantic Web–based knowledge graph to ensure consistent data interpretation. Using the Next Generation Service Interfaces–Linked Data (NGSI-LD) model, CANDIL federates and normalizes data to provide unified, real-time access while reducing redundancy and latency. Plebani et al. [182] take a complementary approach with TEADAL, a project that implements “stretched data lakes” spanning Cloud and Edge infrastructures. TEADAL enables federated data sharing while preserving data sovereignty. Through optimized data placement and energy-aware transfer mechanisms, it helps minimize duplication and storage overhead, benefiting data-intensive domains such as healthcare and finance.

**Federated Data Storage and Management** In contrast to distributed storage, where data are spread across multiple physical location but remain centrally managed, federated data storage connects independently governed systems, maintaining local

control while preserving data sovereignty. In this context, Aznar Poveda et al. [183] propose SDKV, a distributed key–value store designed specifically for the continuum. The system adapts data placement based on client proximity, which helps to reduce latency and network load. It also supports several consistency models to balance real-time and eventual access requirements. Similarly, Carrizales-Espinoza et al. [184] propose StructMesh, a federated storage framework tailored for serverless environments operating on the continuum. StructMesh uses a mesh-based topology to support secure data transfer and dynamic resource allocation depends on the serverless workloads.

#### 4.2.2 Development and Deployment

Another major technical challenge in the CC concerns the development and deployment of services. As the CC has a much higher level of distribution compared to previous computing models, it requires new approaches, as discussed in the earlier chapters of this dissertation. The collected publications under this subcategory address these challenges and are discussed in the following sections.

##### Deployment Frameworks

Among the collected publications, deployment automation is an important topic. Jansen et al. [185] present Continuum, automating infrastructure deployment for benchmarking and resource orchestration across Cloud and Edge. Another automated deployment framework is Colony, proposed by Lordan et al. [186]. As a framework specifically designed for FaaS, it uses hierarchically organized “Agents” for parallel, low-latency task execution across the continuum. Hass and Spillner [187] propose Continuum Deployer, an interactive tool for deploying applications across Cloud, Fog, and Edge with algorithm-based resource mapping. Nastic [188] presents a self-provisioning serverless infrastructure for FaaS and backend services, using AI-driven optimizations for zero-touch deployment.

Addressing another cloud-native technology, Caron et al. [189] introduce the Nanoservices Framework, grouping microservices into single containers to reduce CPU, memory, and storage usage in edge deployments. Carnero et al. [190] and Torres et al. [191] present Kafka-ML frameworks for managing Distributed Deep Neural Network (DDNN) inference across the cloud-to-things continuum, using Apache Kafka and Kubernetes for low-latency, fault-tolerant AI deployments using BranchyNet for efficient early exits. Lastly, Judvaitis et al. [192] develop a DevOps framework for mobile IoT environments, modularizing software across IoT, Edge, and Cloud. This approach enables real-time control and resilience in intelligent transportation systems.

##### Deployment Strategies

The FREEDA framework by Vitali et al. [193] offers a DevOps-oriented and energy-aware deployment strategy for microservices across the continuum by dynamically adjusting to system health and energy use.

Cohen et al. [194] take a different approach with a distributed service-provisioning protocol based on asynchronous and decentralized orchestration, aiming to reduce bandwidth usage and overhead in highly dynamic, resource-constrained environments. Their follow-up work, the Distributed Asynchronous Placement Protocol for the Edge–Cloud Continuum (DAPP-ECC) [195], extends this design to support local decision-making for microservice migration, improving latency and operational efficiency.

For latency-sensitive AI applications, Basaras et al. [196] analyze deployment trade-offs in 5G-connected port environments, balancing latency reduction with energy and bandwidth costs. In healthcare, Song et al. [197] propose a model-based deployment strategy for IoT devices in smart healthcare networks, enabling automated, context-aware placement. Fu et al. [59, 61] present Nautilus, a reinforcement learning-based microservice mapper and resource manager that can maintain QoS by optimizing placement based on network and computational demands.

In contrast to other approaches, Montoya-Munoz et al. [198] address reliability by evaluating cost-reliability trade-offs through dedicated and shared redundancy schemes, providing an economically feasible, resilient deployment strategy for resource-limited environments.

### Development Technologies and Tools

In the collected dataset, several new development tools have been proposed. For example, Galante et al. [199] propose an adaptability-focused parallel programming framework which extends traditional patterns for dynamic resource allocation and fault tolerance. Aldinucci et al. [200] introduce a Continuum-aware Programming Model with the "Hugger" pattern for autonomous service distribution, monitoring, and migration.

For 6G-enabled IoT environments, Baldoni et al. [201] develop Zenoh-Flow for latency-sensitive, distributed data flow. Marcelino and Nastic [202] introduce CWASI, a WebAssembly runtime shim for low-latency inter-function communication in serverless deployments. Ménétréy et al. [203] propose a WebAssembly framework with Trusted Execution Environments (TEEs) for secure, lightweight, cross-platform application deployment. In a different context, Russo et al. [204] present Serverledge, a decentralized FaaS framework enabling vertical and horizontal offloading for latency-sensitive IoT applications, while Sicari et al. [205] develop OpenWolf, a serverless workflow engine that orchestrates complex workflows using Kubernetes and OpenFaaS across the continuum. This help reduce latency and optimizing resource use for IoT infrastructures.

#### 4.2.3 Enabling Technologies and Paradigms

Given the emerging nature of the CC, several cutting-edge technologies are discussed to evaluate their impact on its development. The following technologies have been mentioned in the collected dataset.

## 6G

The 6G technology is the sixth generation of wireless communication, promising faster speeds, lower latency, and other connectivity improvements [206]. Kumar et al. [207] describe 6G-enabled Continuum frameworks that integrate IoT, Edge, and Cloud resources to bring computation closer to data sources. Their approach can be useful in latency-sensitive use cases, such as industrial automation and remote healthcare. The applicability of 6G-enabled CC also expands to other domains. For example, Zheng [208] explores its role in adaptive e-learning. The proposed Intelligent Language Acquisition Model combines federated learning with deep reinforcement learning and uses Intelligent Software Agents (ISAs) to monitor student behavior and emotions. By utilizing 6G's ultra-low-latency communication, this system enables near real-time feedback and a high degree of personalization. Although this is still a concept, the initial evaluations highlight how 6G low-latency capability could enable responsive learning environments in the future.

## Cloud Native Technologies

Cloud-native technologies can provide adaptable and event-driven architectures that span the entire continuum [209]. Paraskevoulakou and Kyriazis [210] introduce a serverless FaaS architecture using Apache OpenWhisk to deploy ML pipelines across the continuum. This approach increases system flexibility and responsiveness in latency-sensitive applications such as fraud detection. Also using FaaS, Kousiouris and Kyriazis [211] propose a generalized architecture that enables dynamic application placement through middleware extensions, semantic service descriptions, and enhanced runtimes. Their solution addresses the challenge of orchestrating polyglot applications across heterogeneous resources while maintaining compliance with diverse Service Level Objectives (SLOs). Lastly, Chochliouros et al. [212] present the NEMO framework, a meta-operating system designed for modular AI and IoT services. NEMO emphasizes cybersecurity and automation by adopting an intent-based DevZeroOps approach that supports on-device intelligence, federated learning, and secure, low-latency service execution.

## Digital Twin

The Digital Twin (DT) is a virtual representation of physical systems, which can be used for real-time monitoring and predictive analysis [213]. Picone et al. [214] propose Fluid Digital Twins (FDT), which apply fluid computing to enable dynamic task migration and latency-aware resource optimization in applications like computer vision. In another use case, Toro-Gálvez et al. [215] introduce a Hierarchical Urban Digital Twin (UDT) Model distributing roles across Edge, Fog, and Cloud for real-time urban management. Their follow-up work [216] extends this approach using a model-driven framework based on Eclipse Ditto and UML to enhance routing flexibility.

Qu et al.'s blockchained dual-asynchronous federated learning (BAFL-DT) [217] integrates blockchain and FL in DTs to enable secure, decentralized AI across the continuum. In telecommunications, Raza et al. [218] present the Digital

Twin Network (DTN) for AI-driven predictive maintenance and flexible network management. Alsubai et al.'s Athlete Healthcare Digital Twin [219] proposes a framework using Bayesian and Multi-scaled Long Term Memory (MLSTM) analysis for real-time athlete health monitoring. Lastly, Rattanatamrong et al.'s Smart Building Digital Twin [220] uses IoT and Edge-Cloud integration for energy-efficient, secure building management.

### Quantum Computing

Quantum computing introduces a new computing paradigm based on quantum mechanics to address computationally complex problems. Cranganore et al. [221] explore hybrid quantum-classical workflows for molecular dynamics and combinatorial optimization, combining the processing speed of quantum systems with the reliability of classical computing. Their approach distributes computation across quantum and classical nodes to reduce the impact of noisy qubits and improve overall efficiency.

For simulation, Nguyen et al. [222] present iQuantum, a framework designed to model quantum environments, evaluating resource management strategies and hybrid task orchestration within a quantum-enabled continuum. Furutanpey et al. [223] propose an Edge-Cloud architecture that integrates mobile Quantum Processing Units (QPUs) using “warm-starting” and circuit-cutting techniques for quantum neural networks. Their results demonstrate that mobile QPUs can enhance latency-sensitive applications such as real-time augmented reality by reducing communication overhead and accelerating inference speed.

#### 4.2.4 Interoperability and Integration

Due to the heterogeneous nature of CC, achieving interoperability across layers has become a major technical challenge. To address this, Papathanail et al. [224, 225] propose VOStack, a multi-layered software stack that virtualizes IoT devices into Virtual Objects (VOs), enabling integration across heterogeneous devices using protocols like HTTP (Hypertext Transfer Protocol) and MQTT (Message Queuing Telemetry Transport). Similarly, Wasielewska-Michniewska et al. [226] apply semantic data processing for real-time, constraint-based resource selection in the CC.

Korontanis et al.'s ACCORDION framework [227] federates mini-clouds for flexible resource pooling while maintaining privacy for NextGen applications. In a different context, Vila et al. [228] present an ontology-driven semantic interoperability model for IoT-based systems, supporting rapid, context-aware data exchange in autonomous vehicles.

Galantino et al. [229] introduce the REsource Advertisement and Reservation (REAR) protocol for intent-based, carbon-aware, zero-trust resource allocation, validated in the FLUIDOS project. Lastly, Simion et al. [230] explore serverless computing orchestration across the continuum to dynamically offload workloads based on latency and load. The initial results demonstrate the feasibility of this approach in enabling flexible, scalable, and interoperable resource allocation.

### 4.2.5 Modeling and Simulation

As the CC's structure becomes increasingly complex, several studies focus on modeling and simulation, which provide a foundation for understanding, developing, and evaluating advanced approaches for the CC.

#### Modeling Techniques

Modeling approaches address complexity, interdependencies, and real-time constraints across the CC's heterogeneous environments. For example, Patki et al.'s Fluxion framework [231] introduces a graph-based resource model that addresses the scheduling challenges presented by high-performance computing (HPC) environments. For intelligent systems, Sedlak et al. [232] propose using Markov Blankets to model the interaction of Edge devices. By focusing on a subset of operational variables, the Markov Blanket model reduces computational overhead and enables each device to make localized decisions that align with global service-level objectives (SLOs). Using ML, Encinas et al. [233] present a data-driven model for reconfigurable multi-accelerator systems on Field Programmable Gate Array (FPGA) platforms to optimize energy and performance dynamically.

For the federated FaaS model, Bauer et al. [234] use the Globus Compute dataset, which provides data collected from numerous users and endpoints. This work provides valuable insights into workload patterns and the complexities of managing FaaS deployments over heterogeneous environments.

Other works include Kashansky et al. [235], who apply a Monte Carlo and XY-model approach over Barabási–Albert topologies to study workflows and bandwidth distribution under noise. Markus et al. [236, 237] model energy consumption in CC using DISSECT-CF-Fog, highlighting energy impacts in distributed environments. Balouek-Thomert et al. [238] propose the MDSC hierarchical modeling framework for distributed stream processing. Lastly, Bernabé et al. [239] introduce the Edge Cloud Ontology (ECO) to enhance interoperability and adaptive management in IoT systems spanning continuum layers, modeling resource availability and link quality under dynamic conditions.

#### Simulation Frameworks

In addition to modeling techniques, the dataset includes several simulation frameworks. For example, SimulateIoT [240] uses a model-driven development approach to simulate IoT-based task scheduling within the continuum. By modeling IoT systems with varying nodes and tasks, SimulateIoT enables users to design and evaluate workflows specific to industrial applications.

iFogSim [241] focuses on end-to-end energy consumption analysis for deployed applications. By modeling various application modules, iFogSim evaluates different deployment scenarios, comparing traditional Cloud-only setups with Edge-oriented and hybrid configurations.

For Osmotic Computing, which was briefly mentioned in the first part of this dissertation, IoTSim-Osmosis [242] supports the unified modeling and simulation of IoT applications. By integrating Software-Defined Networking (SDN), this framework

allows dynamic resource management and data routing. IoTSim-Osmosis addresses challenges like network heterogeneity and the complexity of service-oriented architectures, providing a platform for testing microservice-based IoT applications.

FaaS workloads can be evaluated using faas-sim [243], which uses trace-driven simulations. Another FaaS simulator is Matricardi et al.'s  $\lambda$ ContSim [244], which simulates metrics like energy use and service times while supporting node failures and migration scenarios.

Valdez et al.'s SIMaaS [245] unifies blockchain, IoT, Fog, and Cloud simulations, supporting AI-based resource optimization for smart applications. Lastly, Markus et al.'s DISSECT-CF-Fog [246] provides energy and mobility modeling for CC, supporting scenario-based evaluations for latency-sensitive applications.

#### 4.2.6 Networking

One of the main use cases of CC in general, and Edge computing in particular, is latency-sensitive applications. For such scenarios, several frameworks focused on network slicing and dynamic network resource management across the continuum have been found in the collected dataset.

For example, Habeeb et al. [247] propose a dynamic bandwidth slicing framework using SDN with multi-level queue traffic scheduling to prioritize latency-sensitive applications, reducing latency and improving energy efficiency. Maciel et al.'s NECOS Cloud-Network Slicing Management and Orchestration (NS-MANO) platform [248] enables end-to-end lifecycle management of programmable network and cloud slices for 5G IoT applications, maintaining QoS under dynamic demands.

Finocchio et al. [249] present the Multi-Transport Communication Library (MTCL), a C++ library unifying protocols such as MQTT, MPI, and TCP under one API for scalable, low-overhead communication across IoT, Edge, and High-Performance Computing (HPC) environments. Horvath et al.'s Mobile Edge Service Discovery using DNS (MESDD) framework [250] utilizes geofence-based discovery with Intermediate Discovery Codes (IDCs) for location-aware, low-latency service discovery in Edge-centric IoT applications.

#### 4.2.7 Operational Management

The Operational Management subcategory contains works that aim at optimizing the deployment, allocation, and maintenance of resources across the CC. It focuses on coordinating Cloud, Fog, and Edge infrastructures to meet workload requirements. This can be further divided into different subcategories, as discussed below.

##### Service Life-cycle Management

Robust life-cycle management within the CC is essential to achieve resilience, adaptability, and system efficiency. For example, Truong and Magoutis's RVE Accelerators model [251] supports programmable elasticity for end-to-end lifecycle integration and real-time adaptation in distributed workflows. In a concrete smart grid use case, Boloorchi and Azizi's IoT-Fog-Cloud framework [252] proposes using Fog nodes for real-time energy monitoring and dynamic pricing to balance demand.

Zanella et al.'s BarMan framework [253] utilizes the BarbequeRTRM Run-Time Resource Manager for adaptive resource allocation, while the BeeR extension enables dynamic task offloading, reducing processing latency in distributed applications. Volpert et al. [254] propose a holistic Cloud-to-Thing (C2T) lifecycle management framework using cloud-native practices, containerization, and lightweight Kubernetes (K3S) to integrate Cloud, Edge, and Single Board Computers (SBCs). Finally, Houmani et al. [255] present a microservices-based system for managing time-critical Deep Learning (DL) workflows across the continuum, dynamically balancing latency and accuracy. The evaluation shows a 54.4% makespan reduction in real-time applications compared with a cloud-only setup.

## Resource Management

Within the collected dataset, studies on resource management span strategies addressing performance and efficiency. An overview of these approaches is presented in Table 4.1. These works are discussed in the following sections.

**Load Balancing** Several load balancing strategies have been found in the literature. For example, Vijarania et al. [257] propose an energy-efficient load-balancing mechanism using Fog computing to offload workloads from Cloud servers. By dynamically deactivating underutilized Fog nodes, the mechanism reduces response times and conserves energy, suitable for smart IoT applications. A more advanced technique is proposed by Shamsa et al. [256]. Their work presents a distributed, prediction-based load balancing approach that segments environments into grid cells. Excess load is redistributed based on real-time evaluations to prevent deadline misses and improve scalability for large-scale deployments such as smart cities.

**Orchestration** Ullah et al.'s MiCADO-Edge [66] extends traditional Cloud orchestration capabilities to the Edge, enabling flexible microservice deployment and runtime management across multi-layered infrastructures. Similarly, Grigoropoulos and Lalis's Fractus [258] extends existing orchestration frameworks to manage applications spanning the Drone-Edge-Cloud continuum, specifically targeting mobile drone environments. Fractus manages the spatial distribution and connectivity of drones through mission-aware resource allocation, aligning operations with application requirements. For latency-sensitive and time-critical IoT use cases, the DECICE framework (Kunkel et al. [259]) offers an advanced AI-driven orchestration framework, dynamically allocating resources across Cloud, Edge, and high-performance computing (HPC) environments. At its core, DECICE uses a digital twin to continuously monitor real-time data on system states, such as resource availability, network conditions, and application demands, enabling its AI models to predict resource needs, balance workloads, and adjust scheduling to maintain optimal performance and energy efficiency. Another AI-based orchestrator is the Multi-Agent Reinforcement Learning (MARL) framework by Zafeiropoulos et al. [260]. In their proposal, multiple MARL agents are deployed to optimize autoscaling in complex, distributed applications. Each agent autonomously manages localized resources, monitoring metrics like CPU usage and latency to meet Service Level

TABLE 4.1: Summary of Resource Management Strategies

<b>Subdomain</b>	<b>Key Approaches</b>
<i>Load Balancing</i>	Prediction-based strategies [256] and energy-aware strategies [257].
<i>Orchestration</i>	Approaches range from policy-driven tools (e.g., MiCADO-Edge [66], Fractus [258]) to AI-based orchestration (e.g., DECICE [259], MARL [260]). Includes QoS-aware systems like FogArm [261], TOSCA-based orchestration (Alien4Cloud [262], Infrastructure Manager [263]), as well as Kubernetes/FaaS extensions such as KubeEdge-V [264], Kontinuum Controller [265], OpenWolf [266], Eclipse ioFog [267], ACOA [60], and AVEC [268].
<i>Resource Allocation</i>	Techniques use reinforcement learning [269], security-aware Mixed Integer Linear Programming (MILP) models [270], autonomic control [271], and dual-phase heuristics [272].
<i>Resource Management Framework</i>	Decentralized and autonomous frameworks reduce bottlenecks and support scalable operations. Examples include Serverledge [273], CODECO [274], MLSysOps [275], AIF-based models [276], intent-driven [277], VoI-aware systems [278], and the ELASTIC architecture for real-time, energy-efficient coordination [279].
<i>Service Placement</i>	Optimizes placement using user preferences [280], carbon footprint awareness [281], and SLA-compliant models [282].
<i>Task Allocation</i>	Methods such as DISSECT-CF-Fog [283] balance workload under constraints.
<i>Task Offloading</i>	Includes MIMO-based offloading [284], AI/ILP-based models [285, 286], adaptive queuing [287], incentive models [288], and ML workload migration [289].
<i>Task Scheduling</i>	Approaches include Kubernetes-based schedulers [290], ant-colony methods [291], ML-enhanced techniques [292], and priority-aware schedulers [293, 294, 295, 296, 297].

Agreements (SLAs) while minimizing resource consumption. FogArm by Bisicchia et al. [261] introduces QoS-aware orchestration for multi-service applications. It utilizes FogBrainX, a continuous reasoning engine, and FogMon, a lightweight monitoring tool, to dynamically manage and monitor services with minimal overhead. Similarly, Alien4Cloud by Spătaru et al. [262] uses Topology and Orchestration Specification for Cloud Applications (TOSCA)-based topologies to maintain modular, consistent deployments across Kubernetes clusters. Another TOSCA-based orchestrator, Infrastructure Manager (IM) by Caballer et al. [263], extends orchestration capabilities to include cloud VMs and FaaS platforms. IM integrates Infrastructure as Code (IaC) with DevOps tools like Ansible to streamline complex multi-cloud deployments. RISC-V is an open-standard instruction set architecture (ISA) based on Reduced Instruction Set Computing (RISC) principles, offering a fully open-source alternative to proprietary ISAs like ARM or x86. Its open-source nature and energy efficiency make it a good choice for low-power edge computing [127]. KubeEdge-V, presented by Lumpp et al. [264], introduces a Kubernetes-based orchestration platform tailored specifically for RISC-V, extending Kubernetes functionality to support containerized applications on edge devices. In industrial IoT and high-intensity computing applications, maintaining configuration consistency is a major challenge. Hass and Spillner's work on the Kontinuum Controller addresses this with Kubernetes-based workload configuration management across extensive edge-cloud networks, integrating CI/CD pipelines to minimize deployment oscillations [265]. Similarly, OpenWolf (Sicari et al. [266]) enhances real-time responsiveness in IoT with its serverless workflow engine. Using modular function chaining and event-driven orchestration, OpenWolf ensures low latency and rapid responsiveness, adapting application behavior based on environmental changes and QoS requirements. Čilić et al. [267] present a service orchestrator, implemented using Eclipse ioFog to demonstrate and evaluate the service orchestration strategies in an emulated network, and to explore the applicability of ioFog for real-world next-generation IoT solutions. The Application-Centric Orchestration Architecture (ACOA) by Orive et al. [298] introduces a QoS-driven scheduler tailored for high-demand applications like railway systems. ACOA uses Kubernetes to dynamically balance latency and reliability across components, accommodating diverse application needs in variable node environments. Lastly, AVEC presents a GPU-focused container orchestration framework optimized for deep learning and other intensive computing tasks across the continuum's networks [268].

**Resource Allocation** Filippini et al.'s FIGARO [269] applies reinforcement learning for Resource Selection and Component Placement (RS-CP), dynamically refining policies for AI-driven task distribution and ensuring QoS. Security considerations are considered by Soumplis et al. [270], who design a resource allocation model for multi-tier Edge–Cloud systems using Mixed Integer Linear Programming (MILP), and a Multi-Agent Rollout heuristic to balance security, latency, and resource efficiency in microservice orchestration. For resilience in critical scenarios, Balouek and Couillon [271] propose an autonomic Edge–Cloud architecture with Monitor-Analyze-Plan-Execute over a shared Knowledge (MAPE-K) loops for adaptive response in disaster management and early warning systems. Manogaran

and Rawal [272] propose a dual-phase framework with Profitable Resource Allocation (PRA) and Optimal Node Placement (ONP) to reduce communication delays, optimize Fog-node placement, and improve response times for high-density IoT applications.

**Resource Management Framework** Decentralization in resource management within the CC reduces bottlenecks, latency, and single points of failure. For example, Russo et al.'s Serverledge [273] distributes scheduling and task offloading for serverless applications, enabling seamless scaling and low-latency task migration. Sedlak et al. [276] apply Active Inference (AIF) with Bayesian causal models for autonomous, low-latency adjustments in Edge environments to maintain SLOs. Sofia et al.'s CODECO [274] introduces cognitive, AI-driven, decentralized orchestration for mobile edge-cloud infrastructures, while Loaiza et al.'s MLSysOps [275] uses Multi-Agent Systems (MAS) to autonomously manage Device-Edge-Cloud (DEC) resource allocation and application orchestration using MAPE loops. Zafeiropoulos et al. [277] propose an intent-driven orchestration model for hierarchical, goal-oriented resource management across layers. Zaccarini et al.'s VOICE platform [278] uses Value of Information (VoI) strategies for prioritizing critical data and adaptive resource allocation. Lastly, Sousa et al.'s ELASTIC architecture [279] coordinates Edge-Cloud resources to meet real-time and energy efficiency requirements for time-sensitive applications.

**Service Placement** Service placement within the CC optimizes QoS, latency, and resource efficiency using varied criteria. Mutichiro and Kim's QoS-Aware Service Placement Model [280] uses user preferences as the primary driver, relying on mixed-integer linear programming and a similarity-based index to group users with similar demands, reducing response times and costs in Edge environments. Kimovski et al.'s Carinthian Computing Continuum (C3) testbed [281] deploys services across layers by analyzing application needs and carbon footprint to reduce latency and energy usage. Samani et al.'s PROS (Proactive SLA-Aware Placement) Model [282] focuses on SLA-compliance and predictive placement, using a sigmoid-based scoring and betweenness centrality to dynamically monitor and adapt placements, maintaining QoS adherence and timely processing.

**Task Allocation** Markus et al.'s DISSECT-CF-Fog framework [283] explores Distance-Based and Load-Balanced methods prioritizing proximity and distribution, respectively, while privacy-focused Hold-Down keeps tasks local to minimize exposure, and Push-Up rapidly offloads compute-intensive tasks to higher-layer nodes. The advanced Pliant-based algorithm applies fuzzy logic for dynamic, real-time task allocation in resource-limited, multi-layered networks. Simulations demonstrate these strategies' effectiveness across diverse IoT workloads, optimizing resource utilization, reducing latency, and maintaining QoS.

**Task Offloading** Wang et al. [284] propose a multi-tier model utilizing massive Multiple-Input Multiple-Output (MIMO) and Intelligent Reflecting Surfaces (IRS) for optimized, low-latency task offloading. In contrast, Chahed et al.'s AIDA [285] uses AI and reinforcement learning for adaptive, real-time offloading, enhancing

QoS and localized processing. Miloradovic et al. [286] introduce a multi-criteria optimization model using Integer Linear Programming (ILP) and Constraint Programming (CP) to balance latency, energy efficiency, and green computing goals across heterogeneous layers. Al-Tuhafi and Al-Hemairy [287] propose Adaptive Threshold Task Offloading with dynamic queue-based thresholds for efficient resource utilization and reduced response times. Diamanti et al. [288] present a two-stage incentivization mechanism using Contract Theory for delay-tolerant offloading, enhancing cooperative energy-efficient resource use. Dynamically offloading ML workloads based on network and processing conditions, Vicenzi et al.'s DECOS [289] reduces latency by 29% and energy usage by 1.9 times compared to IoT-only execution.

**Task Scheduling** Saito et al. [290] use MEC and optimized Kubernetes scheduling for low-latency, proximity-based processing in a three-tier architecture. Pusztai et al.'s Vela framework [293] enables globally distributed scheduling with a three-phase workflow, enhancing node utilization and reducing latency. Panda et al. [294] apply First Come, First Served (FCFS), Earliest Deadline First–Shortest Job First (EDF-SJF), and FCFS-SJF algorithms for prioritizing real-time IoT tasks, while Mutichiro et al.'s STaSA [291] applies Service-Time-Aware Scheduling in KubeEdge with ant colony optimization to reduce QoS violations. De Palma et al. [295] introduce TAPP for topology-aware, declarative scheduling in serverless environments, aligning resource allocation with geographic constraints. Kumar et al.'s Delta [292] uses ML for dynamic function placement in FaaS deployments, while Matrouk et al.'s MISSION [296] uses multi-phase mobility-aware and energy-aware scheduling for latency-sensitive IoT workloads. Lastly, Platter et al.'s PASIoT [297] introduces a preemptive online scheduler for real-time, priority-based IoT task management, reducing execution time and improving energy efficiency.

## Workflow Management

The final subcategory of Operational Management is Workflow Management. Taghinezhad-Niar and Taheri [299] propose RCSECH (Reliability-Constrained, Security, Energy- and Cost-aware Heuristic) and RSECH (Reliability, Security, Energy and Cost-aware Heuristic) algorithms for scheduling. Their approach optimizes workflow performance across Mist, Edge, Fog, and Cloud, with RCSECH prioritizing strict reliability and RSECH focusing on adaptable, energy-efficient execution. Rosendo et al.'s ProvLight [300], integrated with the E2Clab framework, provides lightweight, efficient provenance tracking for workflows in IoT/Edge environments using compressed data models and MQTT-SN (MQTT for Sensor Networks), achieving up to 37 times faster capture with minimal resource use. Lastly, Dazzi et al. [301] present a federated workflow management framework adhering to FAIR (Findable, Accessible, Interoperable, Reusable) and FACT (Fair, Accurate, Confidential, Transparent) principles, enabling seamless integration of Edge and Cloud resources for data science workflows, supporting ethical, low-latency, and scalable execution.

### 4.2.8 Optimization

Besides Operational Management, Optimization techniques have also received significant research attention. Many of the reviewed studies falls within this group, focusing on enhancing the overall efficiency of the CC. This taxonomy categorizes these works according to their specific optimization objectives, which are detailed in the following sections.

#### Data Transmission and Bandwidth Optimization

Efficient data transmission in the continuum is crucial for processing IoT-generated data at scale. Yi et al. [302] propose a learning-driven transmission algorithm using Fountain codes with Gaussian Elimination (GE) on Field Programmable Gate Arrays (FPGA), achieving low-latency, resilient data transfer under network instability for real-time distributed applications. Similarly, Sowinski et al.'s JELLY framework [303] optimizes data streaming with Protocol Buffers for Resource Description Framework (RDF) compression, using gRPC and Kafka for low-latency, scalable, bandwidth-efficient processing across IoT deployments. Lastly, Nwogbaga et al. [304] introduce IoTCP\_DL for compressing data pre-offload and the Rank Accuracy Estimation Model (RAEM) for compression fidelity prediction, reducing network load. This approach also maintains data accuracy and enhances real-time responsiveness, evaluated in smart city and telemedicine applications.

#### Multi-objective Optimization

In the CC, optimization is often approached as a multi-objective problem targeting metrics such as energy consumption, latency, and QoS. For example, Jafari and Rezvani [305] apply the Non-Dominated Sorting Genetic Algorithm II (NSGA-II) for task offloading, minimizing energy and latency across Fog and Cloud layers for real-time applications like Augmented Reality (AR) and smart cities. Extending NSGA-II for semantic-based resource allocation in Enterprise Resource Planning (ERP) systems, Reffad and Alti [306] minimize energy use and optimize service delivery. By integrating semantic information with a cooperative multi-agent strategy, this system dynamically selects and composes services to match user requirements. Zhang et al. [307] combine compressed sensing with genetic scheduling in MEC to reduce energy consumption and transmission delay, enhancing resource efficiency for bandwidth-constrained environments. Angelelli et al.'s FOA (Function Orchestration Algorithm) [308] optimizes makespan and data transfer costs for serverless functions across the continuum, reducing cold-start delays for latency-sensitive tasks such as machine learning at the Edge.

#### Orchestration

In the CC, orchestration for optimization focuses on dynamic resource allocation to reduce latency, minimize energy consumption, and maximize resource utilization. There are several works found in the collected dataset, each with a different technique. For example, Filinis et al.'s intent-driven framework [309] applies reinforcement learning to link SLOs to orchestration actions, reducing QoS violations and energy

costs. Cai et al.'s SMO (Self-Managed Orchestration) [310] uses Multi-Agent Deep Deterministic Policy Gradient (MADDPG) to optimize microservice placement, reducing latency and maintaining SLA adherence. Decentralized models like Apollo [311] or Rodis and Papadimitriou's Service Function Chain (SFC) approach [312] bring orchestration closer to data sources, reducing bandwidth and computational overhead via distributed agents and unsupervised neural networks. The SERRANO framework [313] utilizes GPUs and FPGAs for hardware-optimized orchestration, reducing processing times for high-performance workloads. Finally, Srichandan et al.'s AOA-PS (Arithmetic Optimization Algorithm with Pattern Search) [314] uses a multi-level feedback queue to orchestrate tasks across IoT-fog-cloud layers, reducing makespan and energy consumption.

### **Resource Allocation and Utilization Optimization**

Yakubu and Murali introduce a two-stage resource allocation model using a Bayesian classifier with Crayfish Optimization Algorithm (COA) for delay-sensitive task handling [315] and a Modified Harris Hawk Optimization (MHHO) framework for dynamic load balancing and cost reduction [316]. Soumplis et al. [317] present a multi-agent rollout framework using MILP and reinforcement learning for cost-efficient, low-latency workload placement.

In the context of microservices, Puliafito et al. [318] address state allocation in microservices within a Cloud-Edge environment by contrasting PaaS with FaaS. Their study demonstrates how dynamically alternating between these service modes can minimize operational costs and enhance resource efficiency. By allowing stateful containers in PaaS to handle persistent states locally and using stateless containers in FaaS, this model ensures low-latency processing and efficient utilization of Edge resources, proving especially beneficial in applications with fluctuating demand, such as IoT and vehicular networks. Also targeting microservices, Wang et al. [319] use linear programming for cross-server microservice deployment under strict SLOs.

Sartzidakis et al. [320] utilize Integer Linear Programming (ILP) for resource allocation in distributed ML, balancing cost and computational performance. Predictive models such as Bauer et al.'s UtilML [321] apply Long Short-Term Memory (LSTM) neural networks for accurate CPU and memory forecasting, while Cui et al. [70] integrate Deep Q-Network (DQN)-based deep reinforcement learning for latency-sensitive IoT-edge-cloud resource management. Gabi et al. [322] propose a Fruit Fly-based Simulated Annealing Optimization Scheme (FSAOS), achieving efficient resource utilization and cost reduction in large-scale, resource-intensive workloads.

### **Service Optimization**

A variety of optimization strategies have been proposed to enhance task and service placement across the Compute Continuum, including heuristics, metaheuristics, machine learning, and mathematical models. Heuristic-based approaches include Almurshed et al.'s Enhanced Greedy Nominator Heuristic (EO-GNH) [323], which optimizes IoT deployments across Edge and Cloud using asynchronous MapReduce and parallel metaheuristics. Sedghani et al. apply randomized greedy algorithms

for AI placement [324, 325]. Similarly, Raith et al.'s pressure-based heuristic supports serverless functions under mobility [326, 327], while Rac and Brorsson propose Kubernetes-based cost-aware scheduling [328]. Almeida et al. with RIC-O [329] enhance latency-sensitive RAN Intelligent Controller deployments. Evolutionary and stochastic methods are explored by Guerrero et al., who introduce a Distributed Genetic Algorithm (DGA) for in-situ optimization [330], and Filippini et al.'s SPACE4AI-R [331], which employs Random and Stochastic Local Search for AI deployment. Approaches using Machine learning like Learning based task placement [332], hierarchical multi-agent rollout [333], QoS-aware heterogeneous FaaS [334], and AttentionFunc [335] optimize latency and efficiency across the continuum. Mathematical optimization models, including CASMaT using Binary Linear Programming (BLP) [336], declarative heuristics for sustainable Virtual Network Function (VNF) placement [337], Service Function Chain Deployment (SFCD) with Edge-First Chained Next Fit (ECNF) and Online Chained Next Fit (OCNF) algorithms [338], and CaMP-INC using ILP for microservice placement [339], further refine deployment efficiency. Markov Decision Process (MDP)-based mMAPO [340] addresses mobility-aware IoT placement, and FaaSDeliver [341] uses Bayesian optimization for cost-efficient FaaS management.

### Task Optimization

Machine learning-based approaches such as Reinforcement Learning (RL) and Deep Reinforcement Learning (DRL) enable adaptive task offloading and scheduling under dynamic network conditions. For example, Abdi et al.'s Q-learning model improves deadline adherence [342], while Heidari et al. use double Q-learning with blockchain-integrated Post-Decision State for green computing and security [343]. Naseri et al. combine a continuous-time Markov decision process with RL for power-performance optimization [344], and Nieto et al. apply DRL with distributed Actor-Critic models in 5G edge-cloud contexts for Quality of Experience (QoE) optimization [345]. Robles-Enciso and Skarmeta's Collaborative Reinforcement Learning system enables multi-layer collaborative learning for efficient Edge task allocation [346], while Cheng et al. utilize federated deep reinforcement learning with UAVs for energy-efficient computing in remote regions [347]. Proietti Mattia and Beraldì use RL for real-time scheduling within the continuum, reducing latency [348].

Metaheuristic methods like Dankolo et al.'s Flower Pollination based Improved Shuffled Frog Leaping algorithm [349], Salehnia et al.'s Multi-Objective Moth-Flame algorithm [350], Singh and Singh's Energy-efficient task offloading strategy (EETOS) using Levy Flight Moth-Flame optimization [351], Abbasi et al.'s multi-objective genetic algorithm [352], and Maashi et al.'s Metaheuristic Mountain Gazelle Optimization Algorithm based task scheduling approach (MMGOA-TSA) [353] balance latency, energy efficiency, and cost under fluctuating workloads.

Heuristic methods are represented by Syrigos et al.'s Energy Efficient and Latency Aware Schedulin (EELAS) for ML inference across the continuum [354] and Rac and Brorsson's cost-effective Kubernetes-based scheduling [355]. Game theory and Lyapunov optimization are also used, with Yu et al.'s intelligent offloading categorizing data and CPU-intensive tasks for improved resource utilization [356] and Wu et al.'s

EEDTO using Lyapunov optimization for blockchain-enabled, energy-efficient task offloading [357].

### Workflow Optimization

Abdi et al. propose a cost-aware offloading framework for workflow execution in the CC, using a nonlinear model and a tailored genetic algorithm to minimize execution costs. Their approach satisfies QoS constraints such as deadlines, security, and data dependencies across layers [358]. Complementing this, Mehran et al.'s C3-MATCH algorithm applies matching theory for asynchronous workflow scheduling, reducing queuing and transmission times in federated environments and enhancing latency-sensitive applications such as sentiment analysis [359]. Rosendo et al. optimize application configurations within E2Clab, tuning energy, latency, and cost for workflows like Pl@ntNet under fluctuating workloads, enabling reproducible, scalable performance adjustments across continuum-based systems [360]. For ML workflows, Syrigos et al.'s MLOps framework decomposes pipelines for stage-specific resource allocation across Cloud, Edge, and IoT, reducing latency and energy consumption while improving execution time through Kubeflow and Kubernetes orchestration [361]. Additionally, Stavrinides and Karatza present a multi-layered resource allocation framework that applies heuristic methods based on the “power of two choices” principle to reduce workload contention and response times in IoT-Fog-Cloud workflows, demonstrating adaptability under diverse workload conditions [362].

### 4.2.9 Security and Privacy

The final subcategory within *Engineering and Technical Solutions* is *Security and Privacy*. Because the CC spans multiple computing domains and interconnects numerous services and devices, security and privacy have become critical research topics. Studies within this subcategory can be further grouped into several main directions, which are discussed in detail in the following sections.

#### Data Security and Integrity

Several studies have proposed encryption, trust-management, and secure-storage mechanisms across the continuum. For example, Lukaj et al. introduce a Transparent Data Encryption (TDE) approach [363] for NoSQL databases, encrypting data at the Edge before storing in the Cloud, enabling secure, schema-free querying without decryption. This is beneficial for privacy-critical domains like healthcare and smart cities. Complementing this, Catalfamo et al. use Homomorphic Encryption to perform computations directly on encrypted data in the Cloud, thus preserving privacy during processing, which is essential for multi-jurisdictional healthcare applications [364]. Ayed et al.'s FogProtect framework integrates adaptive service management with secure containers and compliance management, providing end-to-end data protection and General Data Protection Regulation (GDPR) compliance across distributed systems [365]. Xu et al. propose BTT (Blockchain-based Truthful Data Trading), enabling decentralized, privacy-preserving data trading in vehicular

crowdsensing using blockchain and truth discovery algorithms, enhancing data integrity while reducing transaction costs [366]. Murphy et al. advance trust in distributed ML by introducing a Trust-Based Data Weighting model, assigning metadata-based trust scores to enhance model accuracy by filtering unreliable data, which is critical for healthcare and finance applications. [367]. Finally, Chaudhary et al. develop a Secure Authentication and Cloud Storage model using mutual authentication and Erasure Coding (EC) to maintain data availability and resilience against fragment loss in IoT-Edge-Cloud systems, validated against attacks for high-assurance deployments in smart cities and industrial IoT [368].

### Network and Communication Security

Mocanu et al. present NextEDR, an agent-based EDR (Endpoint Detection and Response) platform utilizing Cloud-Edge integration and machine learning to combat phishing and smishing attacks in mobile environments, using threat intelligence services for real-time protection [369]. In anomaly detection, Li et al. introduce a collaborative model using Marching Squares at the Edge and Kriging interpolation in the Cloud to refine anomaly boundaries with minimal energy and bandwidth consumption, providing low-latency detection for industrial applications [370]. Similarly, Manimurugan et al. utilize Principal Component Analysis (PCA)-enhanced Improved Naïve Bayes (INB) classifiers on Fog nodes for cyber-attack detection in smart city IoT networks, reducing transmission loads while maintaining detection accuracy [371].

For secure authentication and key management, Barbareschi et al. introduce S-PHEMAP, a lightweight PUF (Physical Unclonable Functions)-based protocol that provides low-overhead, end-to-end security in resource-constrained IoT devices across the continuum [372]. Loffi et al. propose a multi-factor mutual authentication scheme using elliptic curve cryptography and nonce-based challenges to protect against impersonation and man-in-the-middle attacks in latency-sensitive environments such as autonomous vehicles and healthcare [373]. Aleisa et al. advance security with a blockchain-based certificate model integrating attribute-based encryption for fine-grained access control within the continuum, supporting authorized communication and secure local data processing before Cloud transmission [374]. Seifelnasr et al.'s provably-secure Mutual Authentication Privacy-preserving protocol with Forward Secrecy (MAPFS) offers decentralized, privacy-preserving mutual authentication with forward secrecy using zero-knowledge proofs and elliptic curve cryptography, protecting IoT-Edge interactions without requiring a central authority [375]. Finally, Babitha and Siddappa utilize Chronological Jellyfish Optimization (CJFO) for scalable, low-overhead key management in IoT-Fog-Cloud systems, optimizing secure multi-phase key generation for large-scale IoT deployments [376].

### Secure Execution and Orchestration

Secure execution and orchestration focus on protecting workflows and function executions in serverless and FaaS architectures across the continuum. For example, Lordan et al. [377] integrate the COMPSs programming model with SCONE to

provide secure in-memory data handling for ML workflows executed within trusted environments (e.g., Intel SGX). The OpenWolf framework by Morabito et al. [378, 379] uses Software Defined Membrane (SDMem) for data transport and OAuth2 with JSON Web Tokens (JWT) to support authenticated serverless orchestration. Bocci et al. [380] propose a declarative FaaS placement enforcing information-flow policies, and Bacchiani et al.'s SEAWALL [381] combines the Web of Things (WoT) with dynamic workload orchestration for low-latency anomaly detection in Industry 4.0. Lastly, Dhanapala et al. [382] present a Zero Trust Architecture (ZTA) with a blockchain-based performance reputation system for adaptive, secure resource management in urban IoT.

## 4.3 System Architectures and Designs

The third and final category in this taxonomy focuses on publications proposing new architectures for the CC. In this dissertation, these works are divided into two main groups. The first group includes *general-purpose architectures*, which can be implemented in a wide range of use cases. These architectures typically follow a *top-down design approach*, in which new models and conceptual frameworks are applied. The second group consists of *domain-specific architectures*, designed for concrete scenarios. These are usually developed using a *bottom-up approach*, in which practical requirements directly influence the architectural decisions. An overview of this category is given in Figure 4.3.

### 4.3.1 General-purpose Architectures

Among the proposed generic architectures for CC, the UNITE framework [383] integrates computing, networking, and storage layers with AI-driven dynamic resource allocation. In contrast, Enoki [384] addresses stateful FaaS execution with a replicated key-value store across Edge-Cloud nodes for low-latency and consistent data access. The Two-Phase Split Computing Framework [385] reduces DNN inference latency using vertical and horizontal splits, and Baiardi et al. [386] demonstrate ECC infrastructures in small-scale research with container orchestration and dynamic provisioning. The SPEC-RG Reference Architecture [387] unifies Edge, Fog, and Cloud for flexible task offloading. Similarly, Self-Distributing Systems (SDS) [388] enables dynamic stateful component relocation for elastic scaling. Additionally, Pujol et al. [389] use Markov Blankets and the Free Energy Principle for adaptive system management. Ferrer et al.'s Cognitive Compute Continuum (CCC) [390] enables decentralized, self-managed swarms. Spillner et al. [391] introduce liquid functions within CoRFu, enabling mobile, pinned, and embedded function execution across the continuum for dynamic offloading and optimized resource utilization.

### 4.3.2 Domain-specific Architectures

Besides generic designs, many domain-specific architectures have been included in the collected dataset. An overview of these architectures is presented in Table 4.3.<sup>1</sup> The

---

<sup>1</sup>Note: Row formatting (alternating background color) assisted by an AI model (Google Gemini 2.5 Pro).

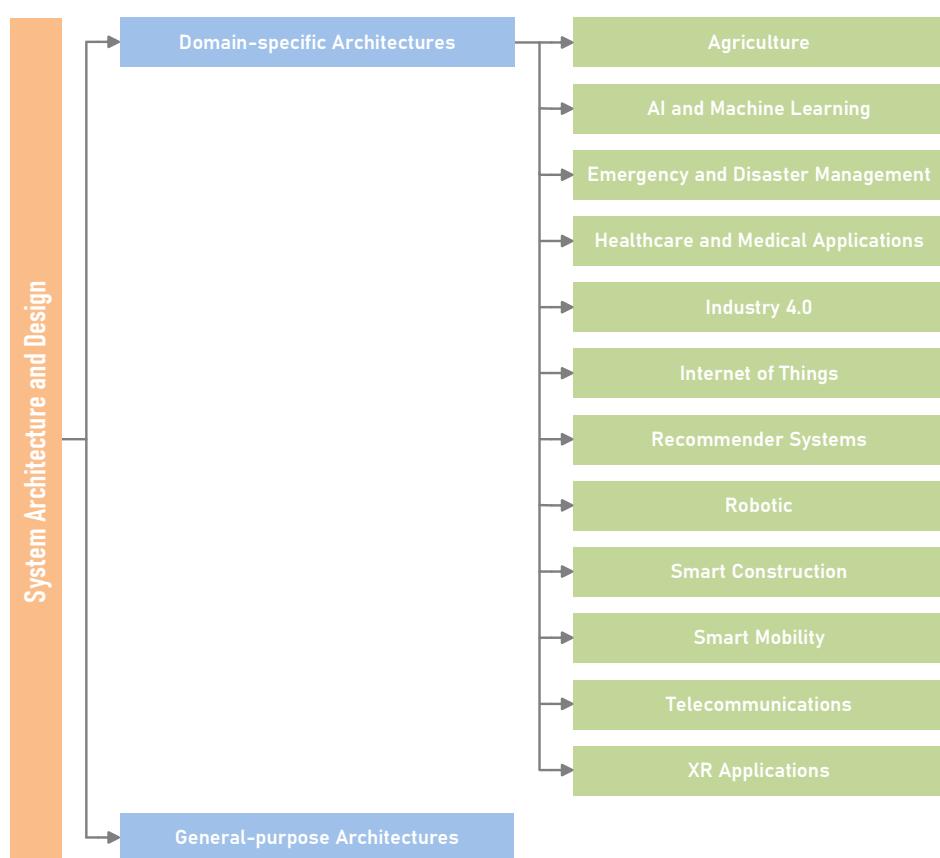


FIGURE 4.3: System Architectures and Designs.

TABLE 4.3: Comparative Analysis of Domain-Specific Architectures.

Domain	Reference	Low-Latency	Interoperability	Scalability	Reliability	Privacy & Security	Bandwidth Control	Cloud-native	Architecture Model
Agriculture	Nam and Choi [392]	✓	✓	✓	✓	✓	✓		Hybrid
	Goleva et al. [393]	✓	✓	✓			✓		Hybrid
Emergency Management	Carnevale et al. [394]	✓	✓	✓	✓	✓	✓	✓	Hybrid
	Ahanger et al. [395]	✓		✓	✓		✓		Hybrid
	Alqahtani et al. [396]	✓	✓	✓	✓	✓	✓		Hybrid
Healthcare	Singh et al. [397]	✓	✓				✓		Hybrid
	Gigli et al. [398]	✓	✓	✓	✓	✓	✓	✓	Hybrid
	Haque et al. [399]	✓		✓	✓	✓	✓	✓	Hybrid
	AlQahtani [400]	✓	✓	✓			✓		Hybrid
	Bhatia & Kumari [401]	✓	✓			✓	✓		Hybrid
	Aral et al. [402]	✓	✓	✓	✓	✓	✓		Hybrid
	Kallel et al. [403]	✓	✓	✓	✓	✓	✓	✓	Hybrid
	Silva et al. [404]	✓	✓	✓			✓		Hybrid
Industry 4.0	Borghesi et al. [405]	✓	✓	✓	✓	✓	✓	✓	Hybrid
AI & ML	Panda et al. [406]	✓	✓	✓		✓	✓		Hybrid
	Townend et al. [407]	✓	✓	✓	✓	✓	✓	✓	Dist.
	Moussa et al. [408]	✓		✓	✓	✓	✓		Dec.
	Trakadas et al. [409]	✓	✓	✓	✓	✓	✓	✓	Hybrid
	Patros et al. [410]	✓	✓		✓	✓	✓	✓	Dist.
	Cui et al. [411]	✓		✓		✓	✓		Hybrid
	Si Salem et al. [412]	✓	✓						Hybrid
	Marozzo et al. [413]	✓	✓	✓		✓	✓		Hybrid
	Peltonen et al. [414]	✓	✓	✓	✓	✓	✓		Hybrid
	Cicceri et al. [415]	✓	✓		✓				Hybrid
IoT	Bacciu et al. [416]	✓	✓	✓	✓	✓	✓	✓	Hybrid
	Zhuang et al. [417]	✓		✓		✓			Hybrid
	Holzer et al. [418]	✓	✓	✓			✓	✓	Hybrid
Recommender Systems	Milani et al. [419]	✓	✓	✓	✓	✓	✓		Hybrid
	Ahn et al. [420]	✓	✓		✓	✓	✓		Hybrid
Robotics	Mahajan & Kaur [421]	✓	✓				✓		Hybrid
Smart Systems	Toffetti et al. [422]	✓	✓	✓	✓	✓	✓	✓	Dist.
Telecommunications	Legierski et al. [423]	✓				✓	✓		Hybrid
	Mehran & Prodan [424]	✓	✓		✓		✓	✓	Dist.
XR Applications	Oztoprak et al. [425]	✓	✓	✓		✓	✓	✓	Hybrid
	Lee & Yoo [426]	✓	✓				✓		Hybrid

✓: indicates a goal explicitly addressed.

**Architecture Model:** Dec.: Decentralized with no central authority, Dist.: Distributed services with centralized orchestration, Hybrid: Edge for latency-sensitive tasks, cloud for compute-intensive tasks.

table summarizes the reviewed publications' key design goals, such as low-latency, scalability, and security, and identifies the architectural model (e.g., Hybrid, Distributed) for each. The detailed discussions of those architectures are given in the next sections.

## Agriculture

Nam and Choi [392] propose a smart farm architecture that integrates IoT Edge computing with blockchain to address bandwidth constraints, limited resources, and the need for real-time processing. Using MQTT for sensor data, Apache Kafka for message exchange, and blockchain for tamper-proof event recording, the system implements trustworthy food supply chains. This design also supports energy-efficient, low-latency environmental monitoring. Complementarily, Goleva et al. [393] present a multi-layer Edge–Dew–Fog–Cloud architecture that uses Long-range wireless communication (LoRa)-based IoT networks and machine learning predictions to monitor soil moisture, temperature, and water usage across large areas, enabling efficient water management, disaster prevention, and sustainable farming.

## Emergency and Disaster Management

Disaster management benefits from CC architectures in resource-constrained scenarios that require low latency. For instance, Carnevale et al. [394] propose TEMA, distributing intelligence across Cloud, Edge, and Deep-Edge layers with federated learning across IoT sensors and drones to support real-time situational awareness. In smart buildings, Ahanger et al. [395] introduce a continuum system that applies support vector machine (SVM)-based event detection and spatio-temporal analysis at the Edge, while the Cloud computes adaptive evacuation routes with an A\* algorithm, balancing energy efficiency and rapid response. Similarly, Alqahtani et al. [396] present a three-layer IoT–Edge–Cloud framework for industrial evacuations, combining Edge-level monitoring with Cloud-based Markov Decision Process modeling to generate reliable, dynamic evacuation strategies.

## Healthcare and Medical Applications

Similarly, the CC's low-latency properties have been explored in several healthcare studies. For example, Singh et al. [397] propose a framework for real-time student stress monitoring using ML models (VGG16, Bi-LSTM, MNB), enabling early interventions via Fog-level analysis. Gigli et al. [398] introduce MAC4PRO, a modular IoT-Edge-Cloud structural health monitoring (SHM) architecture reducing Edge-Cloud data transfers by 90% while detecting micro-changes in structures. Haque et al. [399] present a decentralized Internet of Medical Things - Multi-access Edge Computing (IoMT-MEC) system for secure, resilient patient monitoring. AlQahtani [400] evaluates a 5G-enabled IoT-Fog-Cloud architecture with SDN/NFV for low-latency eHealth applications. Aral et al. [402] highlight federated learning in Cloud-Edge eHealth systems for privacy-preserving, responsive data processing. Similarly, Kallel et al. [403] propose an IoT-Fog-Cloud BPMN-enhanced system for

monitoring autistic children and COVID-19 patients with low-latency processing. Bhatia and Kumari [401] utilize a Tri-logical IoT-Fog-Cloud (TIFC) architecture with Temporal-Recurrent Neural Network (T-RNN) and Self-Organized Mapping (SOM) for real-time Acute Encephalitis Syndrome (AES) outbreak prediction. Lastly, Silva et al. [404] evaluate an Internet of Healthcare Things (IoHT) M/M/c/K queuing architecture, analyzing bottlenecks in high-frequency healthcare monitoring.

### **Industry 4.0**

The IoTwins project [405] presents a digital twin platform for Industry 4.0, integrating IoT, Edge, and Cloud layers for real-time and batch processing across industrial environments. Its hierarchical architecture (IoT twins for local control, Edge twins for mid-tier management, Cloud twins for predictive modeling) enables efficient anomaly detection, predictive maintenance, and system monitoring. Emphasizing openness, modularity, and interoperability, IoTwins improves operational productivity by supporting descriptive, predictive, and prescriptive models and providing flexibility for industrial applications.

### **AI and Machine Learning**

In the collected dataset, several studies demonstrate how integrating AI across IoT, Edge, and Cloud can democratize AI, improve scalability, and enhance responsiveness in data-intensive applications. For instance, Panda et al. [406] and COGNIT [407] emphasize Cloud-Edge integration and serverless orchestration for accessible, adaptive AI. Federated learning approaches ([408], RAMOS [409], Rural AI [410], UDA-HFL [411]) enhance privacy, reduce latency, and enable decentralized model training. Architectures like IDNs [412], Peltonen et al. [414], TEACHING [416], FedUReID [417], and DILoCC [415] address latency-sensitive AI tasks in autonomous driving, healthcare, and surveillance, enabling both data privacy and high performance. Marozzo et al. [413] highlight scalable ML task distribution across Edge-Cloud to balance real-time analytics and complex model training.

### **Internet of Things**

SMT-as-a-Service [418] enables distributed Satisfiability Modulo Theories (SMT) workloads across IoT devices, fog nodes, and cloud resources using a modular architecture and reinforcement learning (Q-learning, deep Q-learning) for context-aware offloading, supporting CPS verification and low-latency, low-energy processes. Another example is Edge2LoRa [419], which enhances LoRaWAN for IoT with edge computing for secure, low-latency data pre-processing at gateways, reducing latency by 10 times and bandwidth by 90%, while remaining compatible with LoRaWAN v1.0.4/1.1. Lastly, Ahn et al. [420] propose an IoT-Edge-Cloud system for real-time smart space control, integrating existing technologies such as DNS Name Autoconfiguration (DNSNA), Indoor Positioning (SmartPDR, PF-IPS), and Constraint Application Protocol (CoAP) for efficient communication, improving localization accuracy and reducing latency in large IoT environments.

## Recommender Systems

CC is also being applied to recommender systems. For example, Mahajan and Kaur [421] propose a three-tier IoT-Edge-Cloud architecture (3T-IEC) to enhance real-time event recommendations in event-based social networks (EBSNs) like Meetup. The system processes IoT data (e.g., user location, weather, traffic) for context-aware recommendations, with sensor data collected on user devices, pre-processed at the Edge to reduce latency, and final recommendations computed in the Cloud using multiple criteria decision-making (MCDM) techniques. Events are ranked based on group influence, category, and economic factors. To address the cold-start problem, social network data is integrated. Experiments show 3T-IEC improves recommendation quality over several baseline methods such as VSM (260%), SVD (144%), CAER (335%), Skyline (100%), and SoCast\* (16%).

## Robotics

For robotics, Toffetti et al. [422] explore application orchestration using the Robot Operating System (ROS) across the continuum. They address challenges like adaptive component placement, stateful ROS node management, and integration of ROS communication with cloud-native microservices. The authors review difference orchestration models: Robot-first (e.g., FogROS2), Edge-first (e.g., KubeROS), and Cloud-first (e.g., Rapyuta), and present adaptive strategies like ElasticROS, which dynamically adapts the computation location based on resource availability. It also highlights the networking “impedance mismatch” between ROS and cloud systems, proposing solutions such as VPNs, Data Distribution Services (DDS) routers, and Zenoh to support distributed ROS deployments.

## Smart Systems

For smart systems, Legierski et al. [423] propose a continuum architecture for real-time Personal Protective Equipment detection (helmets, vests) on construction sites, supporting Edge-only (low latency, privacy) and Edge-Cloud (AWS Rekognition) setups. The system combines YOLOX for detection, DeepSORT for tracking, and MQTT for real-time alerts, achieving high precision and scalability through modular design. Similarly, Mehran and Prodan [424] present a CNN-based Edge–Fog–Cloud architecture for road sign inspection using high-resolution vehicle video. Lightweight CNNs run at the Edge, with high-accuracy models in the Cloud, coordinated by the CODA algorithm [133], which reduces completion time by up to 60% and lowers both CO<sub>2</sub> emissions and data transfer.

## Telecommunication

Oztoprak et al. [425] examine the telecommunications transformation toward the continuum to meet IoT, 5G, and 6G demands, outlining the shift from hardware-defined to SDN (Software-Defined Networking) and NFV (Network Functions Virtualization) for scalable, automated service delivery. The study details decomposing monolithic architectures into microservices, enabling zero-touch management and flexible resource allocation to support low-latency,

high-throughput applications like autonomous vehicles and real-time IoT analytics. Additionally, it explores blockchain integration for secure, decentralized management within MEC, providing tamper-proof data exchange.

### **XR Applications**

Lee and Yoo [426] propose a web-based extended reality (XR) collaboration system supporting seamless VR-AR interoperability for remote collaboration. To address existing XR limitations such as view instability and scalability, the system implements interaction device webization, unified XR representation, and a unified coordinate system. This approach allows VR users to guide other users with synchronized real and virtual poses. Tested among submarine crews, the system improved collaborative maintenance tasks, with visual instructions outperforming verbal guidance.

## **4.4 Chapter Summary**

This chapter has presented the taxonomy developed as part of this dissertation. Building on the foundation of Yang and Tate's framework, it introduced a new classification system, tailored to the characteristics of the Compute Continuum. In total, 291 peer-reviewed publications have been collected and organized into three main categories: *Industry-specific Use Cases and Applications*, *Engineering and Technical Solutions*, and *System Architectures and Designs*. Compared to the original taxonomy, the body of reviewed CC research demonstrates a higher level of diversity in approaches, techniques, and topics. Therefore, the taxonomy proposed in this chapter adopts a higher level of granularity. Several trends and patterns can be identified through the detailed analysis of the reviewed publications in this chapter. A more structured discussion follows in the next chapter, where the research questions and objectives are revisited, and the conclusion are drawn.

# 5 Discussion, Conclusions, and Future Directions

## 5.1 Revisiting the Research Questions

As mentioned in Chapter 3, the research question motivating the taxonomy in Chapter 4 can be decomposed into specific research objectives. This chapter begins by revisiting these objectives.

### 5.1.1 Mapping Research Directions and Activity Levels (R01)

The analysis of the 291 reviewed papers shows a clear pattern: the research landscape is heavily dominated by the *Engineering and Technical Solutions* category, which accounts for 74.23% of the collected literature (as illustrated in Figure 5.1 and detailed in Table 5.1). Even when examining the distribution across years, this category continues to consistently account for the largest share of the collected publications, as illustrated in Figure 5.2. The findings suggest that the field is currently in a strong technical development phase, with most efforts directed toward developing the algorithms, frameworks, and protocols that enable the implementation of the continuum. Given the CC's complexity, establishing a solid technological foundation is a necessary step before developing domain-specific applications.

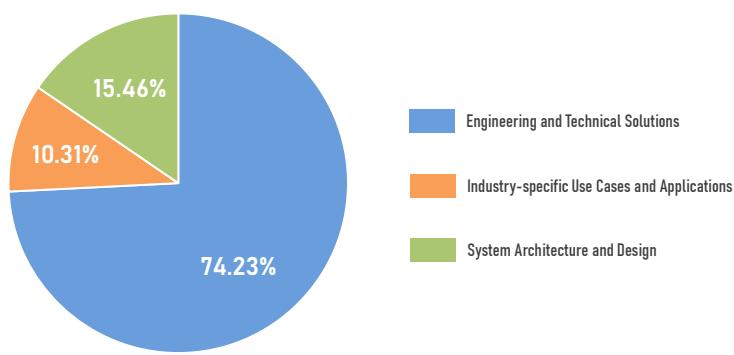


FIGURE 5.1: Classification of Research in the Collected Publications.

Within this category, three subcategories stand out as the most active research directions: *Optimization* (29.17%), *Operational Management* (24.07%), and *Development and Deployment* (10.65%). Together, these topics account for approximately two-thirds of all engineering contributions. This strong focus can be attributed to the fundamental challenges posed by the distributed nature of the CC. With a scale much larger than Cloud or Edge Computing alone, optimization in

TABLE 5.1: Distribution of Research Topics in Engineering and Technical Solutions category by Year.

Research Topic	2021	2022	2023	2024
Data Management	0	4	7	5
Development and Deployment	6	5	9	3
Enabling Technologies and Paradigms	2	1	6	6
Interoperability and Integration	1	1	3	2
Modeling and Simulation	6	1	6	3
Networking	1	1	1	1
Operational Management	10	10	21	11
Optimization	10	14	21	18
Security and Privacy	3	3	10	4

the CC is a highly complex challenge. The collected dataset shows that optimization within the CC has many goals, from bandwidth optimization to resource allocation and utilization optimization. Among others, the majority of selected papers focus on Service Optimization (primarily addressing *placement*) and Task Optimization (primarily addressing *execution strategy*). This strong focus on optimization highlights the academic community's increasing attention to the real-world economic and performance trade-offs within the CC. It signals a shift in the field's maturity, "from a largely conceptual focus toward a system constrained by deployment requirements. Based on the analyzed dataset, several trends can be identified that currently shape the research on optimization:

- *AI-Driven Optimization*: A significant number of studies use ML-based techniques, especially Reinforcement Learning. In service placement, techniques such as learning-based task placement algorithm (LBTP) [332] and AttentionFunc [335] can proactively predict and respond to resource needs. Similarly, models such as Q-learning [342] and Actor-Critic [345] are used to make dynamic offloading and scheduling decisions.
- *Multi-Objective Optimization*: From metaheuristic algorithms such as the Distributed Genetic Algorithm (DGA) [330] and Multi-Objective Moth-Flame Optimization (MOMFO) [350] to mathematical models, including Mixed Integer Linear Programming (MILP) [337] and Binary Linear Programming (BLP) [336], their goal is to optimize several parameters simultaneously, such as latency, cost, energy consumption, and reliability.
- *Method Diversity*: Many techniques have been proposed, each with its own characteristics. Heuristics and metaheuristics (greedy methods [324], genetic algorithms [330]) are valued for flexibility and scalability, while mathematical optimization (Integer Programming [339, 338] and Lyapunov optimization [357]) methods are used when an optimal solution is required. This reflects a trade-off between solution optimality and computational cost in selecting appropriate tools.

Similarly, Operational Management plays a crucial role in CC-based systems' reliability, with a focus on lifecycle automation and orchestration. Development and

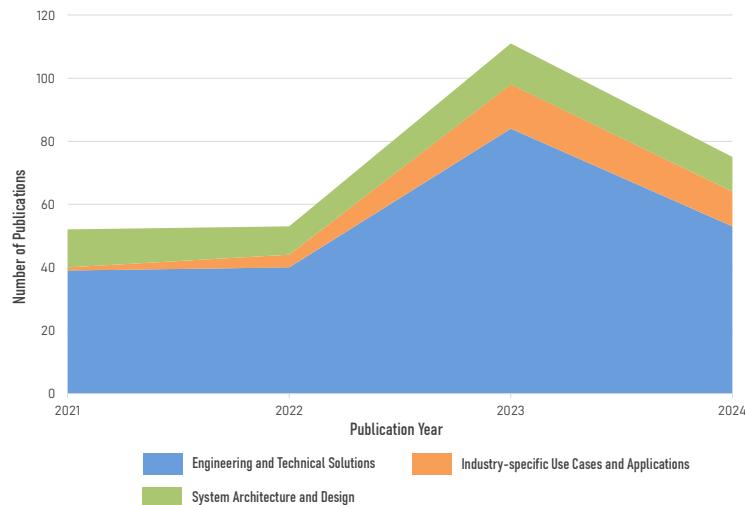


FIGURE 5.2: Publication Trends over the Years.

Deployment also receives considerable attention due to the increasing application of new principles such as cloud-native in the CC environment, requiring an entirely new approach to service development and deployment.

In contrast, the categories of *System Architectures and Designs* (accounting for 15.46% of the collected publications) and *Industry-specific Use Cases and Applications* (10.31%) demonstrate noticeably lower research activity levels. This distribution suggests that, while foundational architectural models and proofs-of-concept are being developed, most current efforts remain centered on addressing immediate technical challenges. These findings highlight a promising direction for future work, which is to develop specialized architectures and explore the CC's potential across a variety of domains.

### 5.1.2 Clarifying Terminology and Formulating a Definition (R02)

The collected literature demonstrates the fragmented use of terms to describe the concept of integrating computing models spanning clouds, edges, and IoT devices. Based on the seven keywords used during the search phase, these terms can be organized into four main groups as shown in Figure 5.3.

- *Cloud-Edge Continuum* or *Edge-Cloud Continuum* were used in 115 out of 291 papers (39.52%), making them the most frequently used terms in the collected dataset. However, these terms are only suitable for architectures focused on Cloud-to-Edge integration but do not take into account IoT or other endpoint devices.
- *Compute Continuum* or *Computing Continuum* were used in 90 papers (30.93%). These two terms provide a broader, more abstract description of the concept.
- *IoT-Edge-Cloud* or *IoT-Fog-Cloud* were used in 66 papers (22.68%). These terms explicitly suggest that IoT is the main data source. They are well-suited for describing hierarchical systems designed for low-latency IoT data processing, where intermediate layers like Fog or Edge play a crucial role.

- *Cloud-to-Things* was used in 20 papers (6.87%). This term emphasizes the complete, end-to-end connectivity from the powerful centralized cloud to constrained endpoint devices.

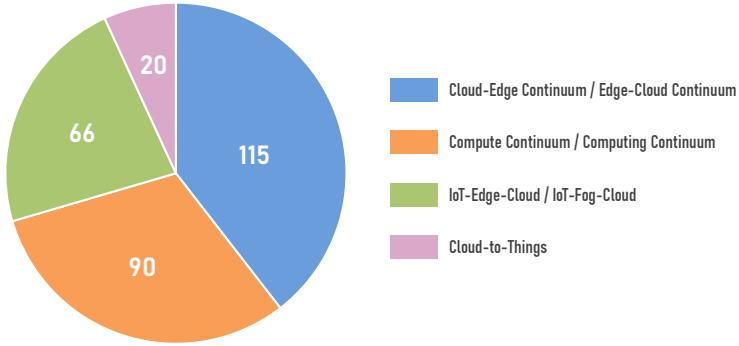


FIGURE 5.3: Distribution of Terminologies used in the collected Publications.

While *Cloud-Edge Continuum* or *Edge-Cloud Continuum* are the most frequently used terms, they imply that the continuum extends only up to the Edge. This dissertation argues that, in a specific use case such as Industry 4.0, they are rather limited and misleading terms. The *Edge* implies a boundary, suggesting the continuum stops at the network edge. However, there are many critical, low-latency operations execute on devices past the Edge, directly on the shop floor, involving actuators, programmable logic controllers (PLCs), and embedded sensors. Additionally, as discussed in the related work section, concepts that equip physical assets with computing capabilities such as *Mist Computing* or *Dew Computing* do not fit within these terms.

From another perspective, *IoT-Fog-Cloud* and *IoT-Edge-Cloud* are specific to IoT. While the IIoT is an important building block of Industry 4.0, these terms are too narrow. An advanced manufacturing environment is a complex ecosystem that includes not only IIoT sensors but also high-performance edge clusters for machine vision, legacy SCADA (Supervisory Control and Data Acquisition) systems, autonomous mobile robots (AMRs), etc. An architecture for Industry 4.0 must therefore integrate this entire heterogeneous collection of systems. Tying the paradigm's name solely to *IoT* or *IIoT* fails to represent this rich and diverse environment.

Finally, the term *Cloud-to-Things* also does not fit well into Industry 4.0 use cases. This term implies a hierarchical data flow primarily from the Cloud down to devices and fails to capture the distributed nature of modern industrial systems. In Industry 4.0, communication is not just top-down from the Cloud. It is multi-directional: peer-to-peer between machines on the production line, and bi-directional for forming closed-loop control. This forms an important argument of the dissertation and will be revisited in greater detail in the following section.

Therefore, the terms *Compute Continuum* and *Computing Continuum* are considered the most appropriate in this dissertation for industrial use cases, as they avoid the limitations associated with others. In fact, “they already appear frequently in the dataset, with almost one-third of all papers using these terms. While *computing* is a gerund (a verb used as a noun) emphasizing the process and activity aspects in

computation, as in *Cloud Computing* or *Edge Computing*; *compute*, on the other hand, is a noun that treats compute as a tangible resource, similar to *storage* or *network*. This aligns perfectly with the focus of this dissertation, which is to engineer a concrete infrastructure where computational resources are provisioned and managed across a seamless spectrum. The term *Compute Continuum* therefore better reflects the infrastructure- and engineering-centric contributions of this work.

Having established *Compute Continuum* as the most suitable term for industrial use case, a precise definition can now be formulated by synthesizing existing perspectives from the literature. Filippini et al. define it as “the natural intersection between Edge Computing, characterized by resource-constrained devices guaranteeing low latencies thanks to the proximity to data-generating sensors, and Cloud Computing, which makes accessible an ideally-unlimited computational power at the price of high network communications overheads” [331]. Similarly, Rac et al. describe it as “the aggregation of computing resources located anywhere between the network edge and servers in traditional datacenters” [328], while Carrizales et al. characterize it as “a paradigm in which diverse and distributed infrastructures are combined to support the deployment and management of data lifecycle stages” [184]. Al et al. expand on this perspective, describing the compute continuum as “the paradigm that integrates, organizes, and considers the computing and network resources across different infrastructures (endpoint devices, edge nodes, cloud data centers, and the networks connecting them) for deploying workloads, instead of relying on a specific infrastructure” [402]. Kimovski et al., on the other hand, emphasize the role of the cloud, defining the continuum as “the extension of cloud with energy-efficient and low-latency devices closer to the data sources located at the network edge” [281].

Other definitions highlight specific aspects of the continuum. For example, Barriga et al. focus on “the coordination of services and resources between the different computing layers” [240], while Legierski et al. note that “data is partially processed on the edge and partially in the cloud” [423].

Synthesizing these perspectives, the *Compute Continuum* concept is defined in this dissertation as follows:

*The Compute Continuum is an integrated, distributed computing model that represents the seamless integration of a full spectrum of resources, spanning from centralized cloud data centers to decentralized edge devices, including IoT sensors and end-user devices. It goes beyond a simple collection of siloed components, establishing a unified computational platform, allowing data and services to be dynamically and intelligently orchestrated to the most logical and efficient location. This placement is driven by the unique requirements of each workload, such as minimizing latency, minimizing network bandwidth, preserving data privacy, or maintaining operational resilience, therefore enabling optimized computing, storage, and communication at any point along the continuum.*

### 5.1.3 A Synthesis of Trends and Values that Shape the Compute Continuum (R03 & R04)

The development of the CC represents not only a technical advancement but also a response to changing demands and strategic objectives. It is driven by several *Underlying Trends* as illustrated in Figure 5.4 to extend the computational power of the cloud to the latency-sensitive edge, creating a unified computational fabric. The analyses conducted in this work indicate this evolution follows a clear trajectory: a foundational paradigm shift (Decentralization) is made possible by a universal technological enabler (Cloud-Native Technologies), all in pursuit of several goals (Low-latency Processing, Secure and Privacy-Preserving Computation, Interoperability). The following sections examine these trends (**R03.x**) and the corresponding values (**R04.x**) that the CC delivers.

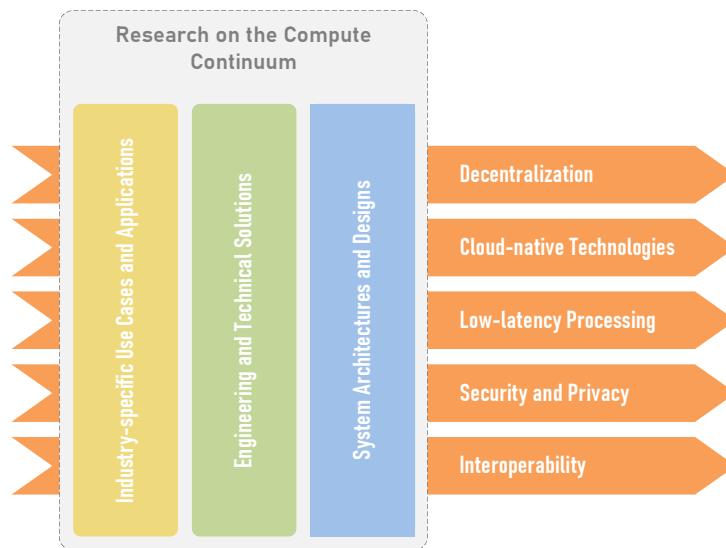


FIGURE 5.4: The Driving Trends of Compute Continuum Research.

#### Decentralization: Enabling Flexibility and Customization

Through the taxonomy, *decentralization* (**RO3.1**) appears as a key theme in the collected papers, addressing fundamental limitations such as high latency, scalability constraints, and single points of failure. This is evident across domains such as data analytics [175], caching [176, 177], service monitoring [200], task offloading [204, 273], and FL [144, 161]. Decentralized orchestration mechanisms [276, 311] also enhance scalability by managing resources and tasks without a central entity, while cloud-native technologies like microservices and serverless architectures inherently support this shift by enabling distributed computing.

This decentralization trend directly enables the CC's most powerful value: *Flexibility in Task and Resource Allocation* (**RO4.1**). By combining edge responsiveness with cloud capacity, architectures can dynamically distribute workloads [160, 150, 156, 162]. This allows for seamless, real-time adaptation and forms a scalable foundation for responsive systems [143, 167, 383, 388, 391]. This flexibility also enables *Domain-Specific Customization* (**RO4.2**). The balance between edge and cloud

can be tailored for the unique needs of diverse industries, from smart farming [392] and healthcare [150] to Industry 4.0 [405] and disaster management [394] and many other industries, each with a specific focus as presented in Table 4.3.

### **Cloud-Native Technologies: Key Enabler for Unified Scalability**

Another key trend in the evolution of the CC found in the literature is the *adoption of cloud-native technologies (R03.2)*. These technologies, initially developed for Cloud-based systems, are increasingly applied to Edge computing and IoT devices. Their maturity and robustness enable seamless integration across heterogeneous environments, supporting greater modularity, scalability, and fault tolerance. This results in *unified scalability (R04.3)* across layers, which is made possible by taking advantage of microservices [189, 255, 422, 154] and serverless functions [188, 205, 210, 211]. These technologies are also the building blocks for framework proposals such as SPEC-RG Reference Architecture [387], UNITE [383] and IoT Twins [405].

### **Low-latency Processing**

With a decentralized architecture powered by cloud-native tools, another primary objective of the CC is realized: achieving *Low-Latency and Real-Time Responsiveness* (both as trend **R03.3** and enabled value **R04.4**). As summarized in Table 4.3, all reviewed CC-based designs explicitly identify low latency as a core design priority. By enabling computation near data sources [140, 152], the CC transforms applications from passive data-gathering tools into active, responsive control platforms. This capability is critical for autonomous systems [147], real-time medical diagnostics [150], and industrial control loops where low-latency is a critical requirement.

Efforts to meet this demand address various latency sources, including transmission latency [302, 307], task offloading [305, 342], and cold-start mitigation [308]. In addition, advancements in orchestration [309, 303, 311], resource allocation [315, 70], and task scheduling [353, 318, 310] further strengthen the continuum's responsiveness. Emerging 5G and 6G technologies will further enhance low-latency performance by delivering new capabilities to the continuum.

### **Security and Privacy in a Decentralized Continuum**

The CC significantly *enhances privacy (R04.5)* by processing sensitive data locally, for example, in federated learning [153, 145, 417, 408, 144]. In these models, raw data (e.g., patient health data or user biometrics) never leaves the Edge device. Only anonymized, aggregated model updates are shared.

However, it also expands the attack surface. *Securing this distributed environment (R03.4)* is a complex trend in itself, with active research in providing secure communication, data integrity, and system resilience. Research efforts span privacy-preserving methods (e.g., secure FL [410, 407, 409]), secure orchestration [377, 378], and domain-specific implementations in healthcare [153, 152], smart environments [164], and logistics [168]. From the collected dataset, the proposed methods can be grouped into three main categories:

- **Data Security and Integrity:** approaches focusing on encryption [363, 364], authentication [368], and data integrity [366].
- **Network and Communication Security:** methods including anomaly detection [370], key management [372, 376], and secure communication [374, 375].
- **Secure Execution and Orchestration:** frameworks and mechanisms for secure orchestration and execution [377, 378, 382, 381].

Emerging technologies such as quantum computing [221, 222, 223] are expected to require new security measures. At the same time, maintaining lightweight yet effective security mechanisms for resource-constrained Edge devices continues to be a challenging research area.

### Interoperability and Standardization

*Interoperability and standardization (R03.5)* are essential for the CC. A true continuum cannot be a collection of siloed systems. Its components must communicate seamlessly but the lack of universal standards remains a major barrier. Overcoming these problems is a key trend identified in the dataset, with researchers developing semantic frameworks [226, 228], abstraction layers like VOSTack [224, 225], and orchestration platforms [227, 230], streamlining the communication across the continuum's layers. Cloud-native technologies (containerization, microservices, service mesh) further enhance interoperability, promoting scalable and resilient deployments [210]. Achieving true interoperability is the foundational prerequisite for unlocking the full potential of the Compute Continuum.

#### 5.1.4 Identifying Research Gaps (R05)

Along with trends and enabled values, several research gaps have been identified in the collected dataset. They will be discussed and presented in the following sections.

### Heterogeneity Management

The core principle of the continuum, integrating diverse resources from IoT devices to cloud servers, also represents its central architectural challenge. The review finds a significant gap in *Heterogeneity Management (R05.1)*. While frameworks like VOSTack offer high-level abstraction [224, 225], the field still needs further interoperable communication protocols (e.g., HTTP, MQTT), semantics, and data models. A unique feature of the Compute Continuum compared to traditional Cloud computing is the inclusion of resource-constrained Edge devices, which creates the need for lightweight protocols tailored to such environments. In summary, without a common language, true interoperability remains difficult, preventing seamless orchestration across the continuum.

## Federated Coordination

This heterogeneity naturally leads to the next challenge: coordination. The taxonomy highlights a strong emphasis on Orchestration and Resource Management [66, 259, 269, 270]. However, many mature solutions rely on a centralized approach inherited from the era of Cloud computing. Tools like MiCADO-Edge [66] and Kubernetes-based frameworks [60, 328], as well as Sedghani et al.'s placement strategy [324], rely on a single decision-making control plane to optimize global QoS and cost. While effective in static setups, this centralization limits scalability and adds latency in distributed, multi-domain environments like the CC. The critical research gap, therefore, is the development of truly *Federated Coordination* (**RO5.2**) mechanisms built on peer-to-peer communication and decentralized orchestration.

## Stateful Service Management

The taxonomy shows a strong focus on optimizing stateless computations, particularly through Task Offloading [342, 343]. This overlooks the fact that many of the most valuable continuum applications, from industrial digital twins to real-time control systems, are inherently stateful. The *management of these stateful services* (**RO5.3**) in a dynamic continuum presents a significant, under-addressed challenge. Existing migration techniques are often adapted from VM migration techniques, which treat a service's state as an opaque block of memory, leading to significant service downtime, which is unacceptable in latency-sensitive applications.

## Security and Privacy

The distributed and resource-constrained nature of devices across the CC creates critical security and privacy challenges. The analysis reveals that while solutions like homomorphic encryption [364], authentication [372], and key management [376] exist, they often target resource-rich cloud layers, leaving IoT devices under-protected [427]. There is a clear gap in integrated, *end-to-end security frameworks* (**RO5.4**) that can protect data and computational operations across the continuum. This is further complicated by the rise of quantum computing, which threatens current cryptography and highlights the urgent need for quantum-resistant methods tailored for constrained devices [428].

## Under-explored Domains

These technical gaps prevented the full implementation of the CC in several demanding and mission-critical sectors (**RO5.5**). While the literature shows broad applicability of the CC, most research focuses on use cases like urban mobility [160], smart buildings [156], energy systems [158, 159], and healthcare [150, 151, 148]. In contrast, domains like Industry 4.0, finance, and scientific computing are significantly under-explored. Although pioneering work exists in industrial frameworks [154] and digital twins [405], key challenges in industrial-grade standardization, security, and dynamic resource management remain largely unaddressed. Addressing these gaps is essential to exploit the continuum's potential in the domains where it could bring the most transformative value.

## 5.2 Chapter Summary

In order to build the foundation for this dissertation, this part has presented a large-scale systematic literature review of the CC. A comprehensive taxonomy has been developed by classifying 291 peer-reviewed publications into three main categories: *Engineering and Technical Solutions*, *System Architectures and Designs*, and *Industry-specific Use Cases and Applications*. This not only provides a structured overview of the field but also helps identify its key trends and critical research gaps.

The results show the field focuses deeply on technical problem-solving. A substantial 74.23% of the reviewed research falls within the Engineering and Technical Solutions category, with a strong focus on Optimization and Operational Management. This suggests that much of the current effort is devoted to addressing the challenges of resource management and orchestration in distributed, heterogeneous systems. Although several works have been proposed addressing architectural design and domain-specific applications, these areas still account for a comparatively smaller portion of the overall research landscape.

In summary, the evolution of the CC is shaped by a set of core *technological trends* and the corresponding *value propositions* they enable. At its foundation, decentralization addresses latency and scalability by distributing decision-making across layers, enabling flexibility in task and resource allocation while enabling domain-specific customization. This paradigm shift is supported through the adoption of cloud-native technologies, which offer the modularity and fault-tolerant capabilities required for unified scalability. Together, these trends support the CC's goal of real-time responsiveness, as evidenced by the prioritization of low-latency design across applications in healthcare, automation, and smart systems. However, decentralization also introduces new privacy and security challenges, motivating the development of local data processing and federated learning methods that preserve privacy. Achieving seamless end-to-end integration further requires resolving interoperability and standardization gaps, as they form the foundation of the CC's ability to deliver scalability, adaptability, and resilience across heterogeneous environments.

Looking ahead, research must progress from optimizing isolated functions toward building a robust, interoperable, and secure fabric capable of realizing the full potential of the continuum. Addressing these gaps is the prerequisite to expand the continuum's applicability in mission-critical domains like Industry 4.0, which remain largely under-explored.

The challenges and research gaps identified in this SLR form the motivation for the architectural and resource management frameworks developed in this dissertation. Specifically, Part III addresses the architectural challenges by proposing a novel architecture for the Industrial Compute Continuum, while Part V focuses on the management of stateful services. Part VI then examines decentralized resource management across the continuum.

Overall, the taxonomy presented in this part provides an extensive, evidence-based understanding of the field, positioning the contributions of this dissertation as a direct response to the most important open questions within the Compute Continuum's research landscape.

## **Part III**

# **The Core Architecture: A Cloud-Native Reference Architecture for Industry 4.0**



# 6 CRACI: A Cloud-Native Reference Architecture

## 6.1 Introduction

The preceding part presented a taxonomy of research in the Compute Continuum, concluding that the lack of a tailored architecture for Industry 4.0 makes it difficult to develop scalable, interoperable, and maintainable industrial systems. In the context of Industry 4.0, where physical assets integrate with digital intelligence, supporting low-latency analytics, autonomous control, and cross-layer integration is critical. Existing models like ISA-95 and RAMI 4.0, while structured, are hierarchical and ill-suited to decentralized and highly flexible systems.

To address this gap, this chapter introduces a cloud-native reference architecture designed to enable flexible service deployment and coordination across heterogeneous infrastructures. The architecture emphasizes modularity, interoperability, and scalability. Following the principles of Attribute-Driven Design (ADD 3.0), the design process prioritized key quality attributes and relied on iterative decision-making. The resulting architecture was developed through the use of standards, industry practices, academic proposals, and state-of-the-art software architecture practices to provide a maintainable and interoperable foundation.

The detailed architectural design is presented in this chapter, part of which was recently published [429]. Chapter 7 then validates its feasibility through MAIA (Microservices-based Architecture for Industrial Analytics) as a concrete use case, demonstrating the design's applicability and the implementation of its principles.

The remainder of this chapter is structured as follows: Section 6.2 overviews related models and architectures. After identifying gaps in these models, Section 6.3 outlines the systematic design process and methodology. This leads to Section 6.4, which presents the core contribution: the proposed reference architecture (CRACI). Section 6.5 compares CRACI with existing models to explain and analyze the design rationale. Section 6.6 closes the chapter by summarizing the key outcomes.

## 6.2 Background and Motivation

In designing the reference architecture, practicality and applicability in real industrial environments were treated as primary requirements. In addition to the scientific works analyzed in the taxonomy, the design process incorporated insights from relevant industrial standards and practices. As a result, the architecture was developed through consultation of four reference layers: industrial standards,

industrial practices, academic proposals, and modern software architecture patterns. These are reviewed and discussed in the following sections.

### 6.2.1 Formal Industry Standards and Models

The first category of related work reviewed in this chapter consists of industry standards and models. These provide the formal structural foundation with which the proposed architecture must remain compatible.

#### The Traditional Hierarchy: ISA-95

The ISA-95 standard [430] is a foundational model for enterprise-control system integration. Based on the Purdue Enterprise Reference Architecture (PERA) and first published in 2000, it defines a hierarchical structure for industrial systems, later adopted globally via IEC standardization as IEC 62264 [431]. ISA-95 organizes manufacturing functions into five levels, often depicted as the automation pyramid (Figure 6.1):

- *Level 0: Production Process:* Sensors, actuators, and machinery.
- *Level 1: Sensing and Manipulating:* Basic control via Programmable Logic Controllers (PLCs) and field devices.
- *Level 2: Monitoring and Supervising:* Supervisory Control and Data Acquisition (SCADA) systems and Human-Machine Interfaces (HMIs) for real-time oversight.
- *Level 3: Manufacturing Operations Management:* Manufacturing Execution System (MES) for scheduling, quality control, and workflow coordination.
- *Level 4: Business Planning and Logistics:* Enterprise Resource Planning (ERP) systems managing logistics, procurement, and finance.



FIGURE 6.1: The ISA-95 Automation Pyramid showing the hierarchical structure from physical processes to enterprise systems [432].

Alongside ISA-88 [433], which governs procedural control, ISA-95 forms the backbone of traditional industrial IT/OT integration. It enables standardized interfaces, reduces integration complexity, and promotes interoperability across

manufacturing environments. While ISA-95 remains relevant as a conceptual model, particularly for legacy system integration, its rigid hierarchical structure presents significant limitations in the context of Industry 4.0. Key limitations include:

- *Rigid Hierarchy*: ISA-95 dictates a vertical, top-down architecture. This limits real-time responsiveness, service mobility, and adaptability in distributed, event-driven environments.
- *Limited Distributed Intelligence*: The model centralizes decision-making at higher levels, conflicting with Industry 4.0 practices where smart sensors, edge nodes, and autonomous devices require local computation and autonomy.
- *No Native Cloud or Cross-Domain Integration*: ISA-95 predates cloud computing and does not address cross-organizational workflows, cloud-edge collaboration, or the Business-to-Business (B2B) and Application-to-Application (A2A) integration needs of digital supply chains.
- *Data Silos*: Security zones and vertical data flows restrict real-time data access needed for advanced analytics and AI.
- *Integration and Analytics Gaps*: ISA-95, especially at higher levels, is designed for Online Transaction Processing (OLTP) and is not optimized for high-volume, Online Analytical Processing (OLAP).

In summary, while ISA-95 remains one of the most important models for industrial system integration, its hierarchical structure is not fully compatible with the principles promoted in modern industrial practices.

### **Lifecycle Management: IEC 62890**

IEC 62890 (Life-cycle management for systems and components) [434] provides a conceptual model for structuring asset management across the system lifecycle by introducing an explicit separation between *Type* and *Instance*.

- *Type*: The design-time blueprint of an asset, defining its standard properties, capabilities, and interfaces (e.g., Pump Model ABC-123 with pressure and rpm attributes).
- *Instance*: The runtime, physical manifestation of a type, with specific identity and operational data (e.g., Pump-SN-987 on Line B showing current pressure).

This separation supports modular and maintainable system architectures. It decouples design-time engineering from runtime management, allowing for reusable models and easier system management.

### **RAMI 4.0: A Model for Industry 4.0**

The Reference Architecture Model for Industry 4.0 (RAMI 4.0) [435], proposed by Germany's *Plattform Industrie 4.0*, is a three-dimensional framework that standardizes the modeling, integration, and classification of Industry 4.0 components. This model aligns IT, OT, and industrial assets across three axes as shown in Figure 6.2.

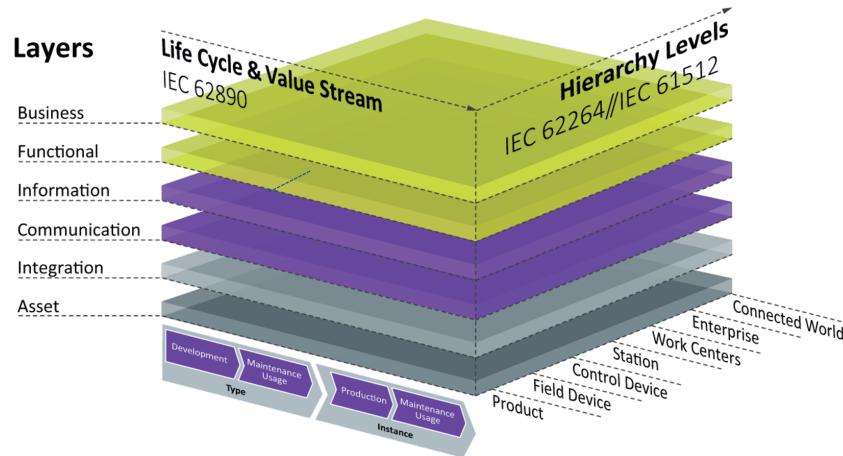


FIGURE 6.2: Reference Architecture Model Industrie 4.0 (RAMI 4.0) [435].

- *X-Axis – Hierarchy Levels:* Based on IEC 62264 [431] and IEC 61512 [436], this axis extends the ISA-95 automation pyramid with six hierarchy levels: *Field Device*, *Control Device*, *Station*, *Work Unit*, *Enterprise*, and *Connected World*.
- *Y-Axis – Life Cycle & Value Stream:* Based on IEC 62890, as explained above, this axis separates the lifecycle into a *Type Phase* and an *Instance Phase*.
- *Z-Axis – Architecture Layers:* Consists of six layers, which are *Asset* (physical devices), *Integration* (bridging physical and digital), *Communication* (protocols and networking), *Information* (data structuring), *Functional* (application logic), and *Business* (processes and goals).

While RAMI 4.0 provides a valuable map for structuring assets and functions, its hierarchical structure struggles to accommodate the dynamic, distributed, and software-defined nature of modern manufacturing environments. There are several limitations of RAMI 4.0 that motivate the need for a new design:

- *A Conceptual, not an Operational Model:* RAMI 4.0 offers a clear framework for classifying where functions belong (e.g., control or enterprise level) but does not guide how to deploy, manage, or migrate services across the CC.
- *Implicit regarding Cross-Cutting Concerns:* While RAMI includes a business layer, it lacks a framework for defining and enforcing cross-cutting concerns such as governance policies or trust management.
- *Hierarchical Assumptions:* Its structural foundation on the automation pyramid retains the strong hierarchical assumptions of top-down data flows. This conflicts with modern, event-driven, bidirectional architectures where edge and business services often communicate directly.
- *Limited External Ecosystem Integration:* Although its “Connected World” hierarchy level explicitly recognizes the need for inter-enterprise collaboration [437], RAMI 4.0 lacks native mechanisms for federated collaboration, for example data sharing via International Data Spaces (IDS) frameworks [438].

In summary, the RAMI 4.0 model retains the main issue of the ISA-95 framework, which is its rigid hierarchical structure, and remains highly abstract.

### 6.2.2 Industry-driven Operational Frameworks

In addition to the abstract models and standards defined by formal standardization bodies, which often remain highly conceptual, industrial consortia tend to adopt a more practical approach. They propose more concrete, implementation-oriented designs that are easier to apply in real industrial settings.

#### FIWARE Smart Industry Reference Architecture

FIWARE Smart Industry Reference Architecture [439, 440] provides an open-source, Next Generation Service Interface (NGSI)-based platform for context-driven industrial applications. This architecture defines several components such as context brokers, smart data models, adapters for industrial protocols, stream processing, etc. The design also supports integration with digital twins and federated data spaces. An overview of the architecture is given in Figure 6.3.

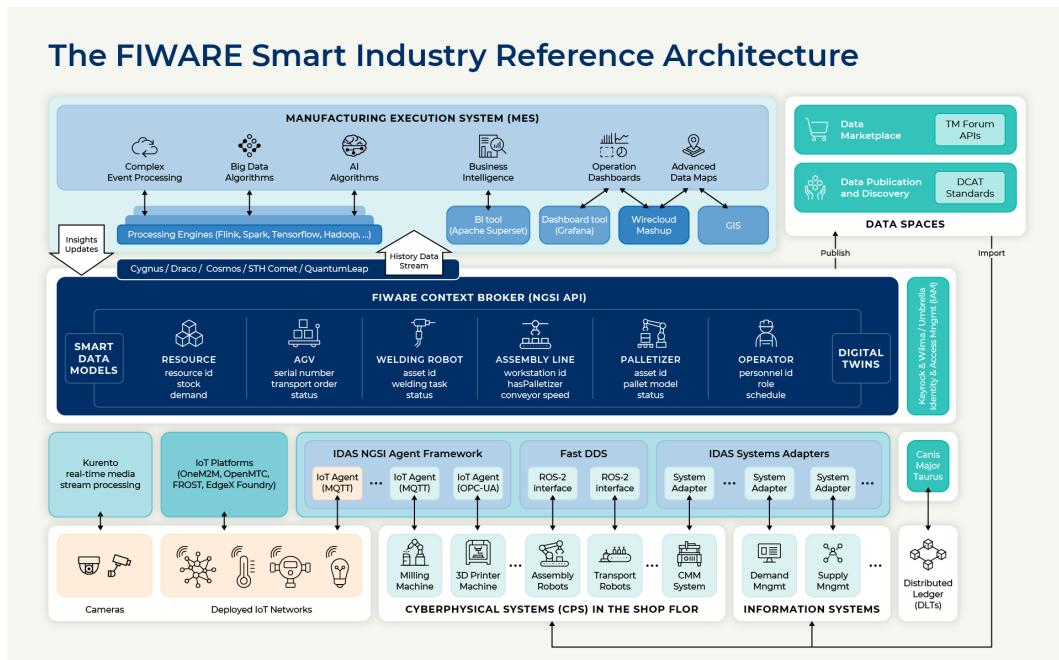


FIGURE 6.3: FIWARE Smart Industry Reference Architecture [439, 440].

Despite its advantages, several limitations remain:

- *Complexity*: FIWARE requires deep expertise in its ecosystem (NGSI-LD, Generic Enablers like Orion<sup>1</sup>, Cygnus<sup>2</sup>, Keyrock<sup>3</sup>) and IoT Agent configurations, resulting in a steeper learning curve than generic cloud-native approaches.

<sup>1</sup><https://github.com/telefonicaid/fiware-orion>

<sup>2</sup><https://github.com/telefonicaid/fiware-cygnus>

<sup>3</sup><https://github.com/ging/fiware-idm>

- *Limited Cross-Cutting Support:* Trust, observability, and lifecycle automation are acknowledged but handled externally, with little guidance on DevOps, monitoring, or governance.
- *No Runtime Orchestration:* FIWARE models data flows but lacks native support for microservice orchestration, dynamic deployment, or adaptive service placement

In summary, the FIWARE Reference Architecture offers a concrete and implementation-oriented framework for Industry 4.0; however, it is also tied to a specific technical stack.

### NAMUR Open Architecture (NOA)

The NAMUR Open Architecture (NOA), developed by the German industrial consortium NAMUR [441, 442], is a reference model that aims to incrementally introduce IT capabilities into the traditionally closed world of process automation. Unlike more complex frameworks such as RAMI 4.0 or ISA-95, NOA does not propose a top-down restructuring. Instead, it introduces a minimally invasive approach, making it particularly suitable for *brownfield environments* where replacing legacy automation systems is cost- and risk-prohibitive. Figure 6.4 provides an overview of the architecture.

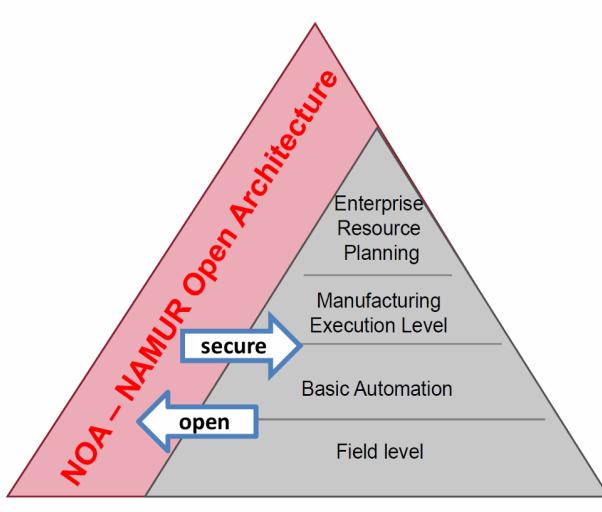


FIGURE 6.4: NAMUR Open Architecture [441, 442].

NOA divides automation systems into two domains: the *Core Process Control (OT domain)*, representing physical controllers such as PLCs, which handle safe, deterministic control and remain unchanged; and the *Monitoring and Optimization (M+O) domain*, a parallel IT domain for diagnostics, predictive maintenance, and analytics, which securely extracts data through a unidirectional gateway without affecting core operations. This is a low-risk path toward IT/OT convergence by preserving the integrity of the control domain. However, there are still some issues that have not been addressed by NOA:

- *Abstract IT Modeling:* NOA treats the M+O domain as a conceptual block, without detailing its internal structure. Therefore, it does not address runtime

management, DevOps processes, observability, or governance models for those services, leaving implementation details to be defined externally.

- *Transitional model, not a full-fledged architecture:* NOA is a structural pattern intended to augment existing systems, but it lacks an architectural model required for building complete, compute-continuum-based industrial systems.

In summary, NOA focuses on integrating IT solutions into brownfield industrial systems. However, as a model developed by an industrial consortium, it remains highly abstract with respect to the IT architecture.

### AIOTI's Computing Continuum Reference Architecture

The EUCloudEdgeIoT.eu initiative and Open Continuum project have proposed a detailed functional model for the CC [443]. This model organizes key capabilities into nine domains: security and privacy, trust and reputation, data management, resource management, orchestration, networking, monitoring and observability, distributed AI/ML, and lifecycle management.

Despite its broad scope, the current model remains largely abstract, providing limited technical details on interoperability, semantic integration across domains, or runtime orchestration. It also lacks concrete implementation patterns or explicit alignment with cloud-native and edge-native technologies. In addition, the model assumes ideal operating conditions across the continuum, overlooking the heterogeneity and latency challenges typical of real-world edge environments. Nevertheless, as a European Commission-backed initiative, and derived from the combined efforts of more than 30 collaborative EU projects, the AIOTI functional model remains an important and credible reference point.

### 6.2.3 Academic Reference Architecture

In addition to industry standards and implementation-oriented models, architectures proposed in academia were also considered. This inclusion completes the overview of the recent developments in CC architectures within the industrial context.

#### LASFA (LAsim Smart FActory) Architecture for Smart Manufacturing

Resman et al. [444] propose the LASFA model as a more user-oriented and pragmatic alternative to the complex RAMI 4.0. The LASFA architecture addresses RAMI 4.0's shortcomings by introducing a simplified, two-dimensional representation of digital twins, agents, and data flows.

In contrast to the traditional automation pyramid, LASFA adopts a decentralized structure in which each production process functions as an autonomous unit with local control, storage, and intelligence. The architecture centers on nested Digital Twins that offer real-time virtual representations of the factory, supported by Digital Agents that process data, optimize operations, and feed results back to physical systems. The architecture also uses both local and global cloud infrastructures for data storage, exchange, and remote access. It can be organized into four main layers as illustrated in Figure 6.5 and explained below:

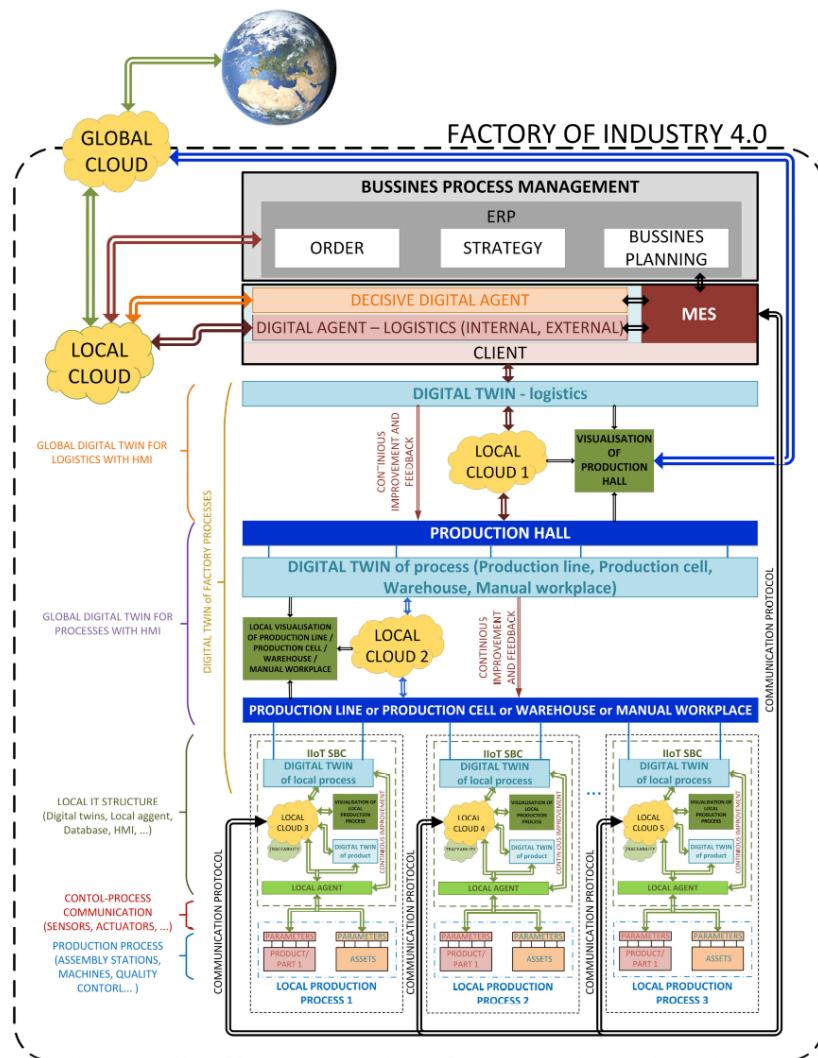


FIGURE 6.5: LASFA (LAsim Smart FFactory) Architecture for Smart Manufacturing [444].

- *Business Process Management*: The highest level, integrating enterprise systems such as ERP and MES for high-level tasks including order management and strategic planning.
- *Factory / Production Hall*: Features a global digital twin of the entire production hall to manage logistics and workflows. This layer is supported by a factory-level cloud and a dedicated logistics agent for optimization.
- *Production Line / Cell*: Contains dedicated digital twins for each production line or cell, modeling their specific operations and material flows. Data is aggregated in a local cloud for line-level visualization and control.
- *Local Production Process*: The most granular level, where each machine or station is a self-contained unit. Each includes a local control unit, digital twins for both process and product, a local AI agent for optimization, and its own micro-cloud for data management.

In summary, the LASFA architecture has partially addressed the limitations found in RAMI 4.0 and ISA-95. However, it lacks mechanisms for brownfield integration and does not support a federated data-sharing model.

### Stuttgart IT Architecture for Manufacturing (SITAM)

The *Stuttgart IT Architecture for Manufacturing (SITAM)* [445] is a conceptual IT architecture designed to enable data-driven factories. The authors highlight the limitations of the traditional information pyramid, including rigid integration, siloed data flows, and limited shop-floor accessibility, which restrict the effective use of industrial data.

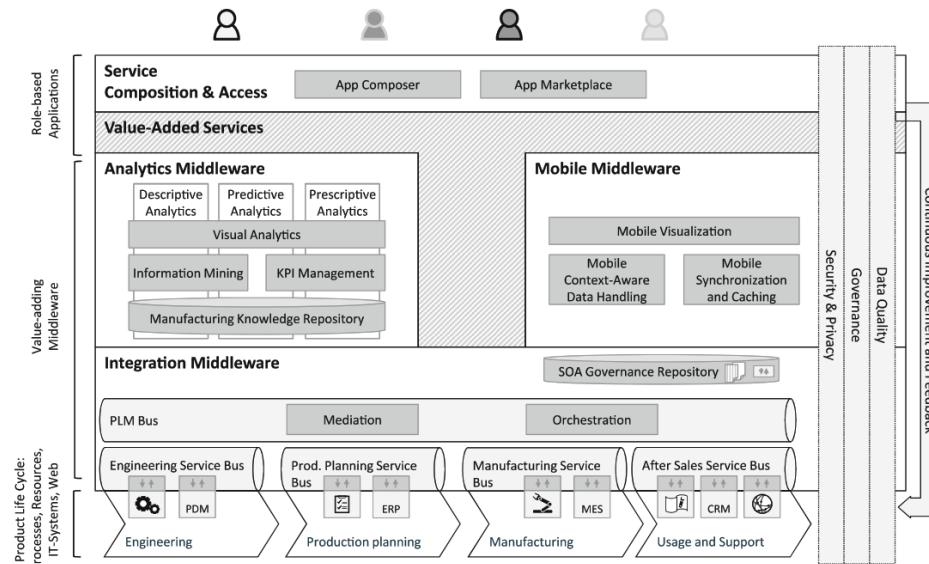


FIGURE 6.6: Overview of the Stuttgart IT Architecture for Manufacturing [445].

To overcome these limitations, the authors propose a *data-driven factory* characterized by adaptability, continuous learning, and a human-centric design. This concept requires an architecture that enables flexible integration of heterogeneous

IT systems, establishes a unified data basis for holistic analytics across the product lifecycle, and supports ubiquitous mobile information provisioning to empower employees at all levels. Motivated by that, SITAM is a service-oriented architecture (SOA) that aims to fill the “granularity gap” between highly abstract reference models (like RAMI 4.0) and specific case-based implementations by providing both a full reference architecture and concrete technological recommendations. Its key components are illustrated in Figure 6.6 and described below:

- *Integration Middleware*: The SOA-based foundational layer. It utilizes a hierarchical structure of Enterprise Service Buses (ESBs), with phase-specific buses (e.g., Engineering, Manufacturing) connected via a central Product-Lifecycle-Management (PLM) Bus. This design flexibly integrates and decouples IT systems across the entire product lifecycle.
- *Analytics Middleware*: The “intelligence” layer, centered on a *Manufacturing Knowledge Repository* that consolidates structured and unstructured data from diverse sources. It enables advanced analytics including descriptive, predictive, and prescriptive methods to generate actionable insights and support a continuously learning factory.
- *Mobile Middleware*: It provides tools to develop and deploy manufacturing-specific mobile applications, offering functionalities such as context-aware data handling, offline synchronization and caching, and tailored visualizations for shop-floor operations.
- *Value-Added Services*: The application layer where functionalities from the middleware are combined. It includes an App Composer that enables even non-technical users to create role-based applications through a drag-and-drop interface, with distribution via an *App Marketplace*.
- *Cross-Architectural Topics*: Overarching principles that apply to all layers. These include *SOA Governance*, managed by a central repository to control service complexity, a comprehensive *Data Quality* framework to ensure the reliability of data and analytics outcomes, and *Security and Privacy*.

Although the SITAM architecture does not explicitly apply cloud-native principles in its design, it successfully overcomes several limitations of ISA-95 and the FIWARE Architecture by proposing a design that maintains a balance between abstraction and granularity.

### The 8C Architecture for Industry 4.0 Smart Factories

Jiang’s 8C architecture [446] extends the 5C architecture (Connection, Conversion, Cyber, Cognition, Configuration) [447] by introducing three additional facets, namely *Coalition*, *Customer*, and *Content*, to address horizontal integration and external collaboration. The addition of these facets creates a more comprehensive framework that structures internal factory hierarchies while explicitly incorporating interactions with external supply chains, end customers, and lifecycle data streams. It positions itself as a model better suited to the demands of mass customization and

interconnected, multi-stakeholder industrial ecosystems. These facets are detailed as below:

- *Coalition*: Addresses horizontal integration across the value and supply chains by enabling collaboration among suppliers, logistics partners, and manufacturers. This enables dynamic formation and reconstruction of supply chains, co-scheduling of production lines and resources across organizational boundaries, and promotes a more flexible manufacturing ecosystem.
- *Customer*: Places the customer at the center of the product lifecycle, shifting from *mass production* to *mass customization*. It supports direct customer involvement in design, real-time tracking of production progress, and mid-production modifications of product specifications. This is enabled by a “product-centric” manufacturing concept, where factories can spontaneously and automatically adapt to individual orders.
- *Content*: Focuses on the extraction, storage, and management of all product-related data to enable “product whole lifecycle service.” It mandates complete traceability records, including raw material sources, production parameters, environmental conditions, after-sales service details, and customer feedback. Such holistic data aggregation supports traceability, advanced analytics, process improvement, and market trend prediction.

Despite these conceptual advances, the 8C architecture remains largely abstract, presenting strategic and technical challenges for implementation.

#### 6.2.4 Software Architecture Patterns

While domain-specific models such as RAMI 4.0 and NOA provide valuable industrial context, a robust reference architecture must also be built on solid software engineering foundations. The proposed architecture implements two such principles: *Cloud-Native Architecture*, inspired by the Twelve-Factor App methodology, and *Event-Driven Architecture (EDA)*.

##### Cloud-Native Principles & The Twelve-Factor App

Cloud-native architecture refers to the design and implementation of applications that fully utilize the Cloud computing paradigm. It focuses not only on the deployment location, but also on how applications are architected, packaged, and managed. This approach favors loosely coupled microservices, packaged as lightweight containers, and orchestrated to enable automated deployment, elastic scaling, and recovery. [448].

The Twelve-Factor App methodology [449] further guides this design, offering best practices for building scalable and maintainable systems. Core principles which are relevant to the design process include:

- *Portability and Decoupling*: Achieving workload mobility by strictly externalizing configuration from code (Factor III), isolating dependencies within containers (Factor II), and treating all backing resources from physical PLCs to cloud databases as swappable attached services (Factor IV). This makes a single service deployable across the entire CC without modification.

- *Resilience and Scalability:* Building for high availability and elasticity by executing services as stateless processes (Factor VI) that are designed for disposability (Factor IX). This "cattle, not pets" philosophy [450] allows an orchestrator to automatically recover from failures and scale services horizontally (Factor VIII) to meet dynamic workloads.
- *Continuous Delivery and Observability:* Establishing a foundation for robust management by treating logs as event streams (Factor XI) for centralized, system-wide monitoring, and strictly separating the build, release, and run stages (Factor V) to enable a reliable, automated CI/CD (Continuous Integration and Continuous Delivery/Deployment) pipeline.

This dissertation applies cloud-native principles and the Twelve-Factor App philosophy to the design of architectures for industrial scenarios. These approaches provide a highly available, flexible, and robust foundation for building modern industrial applications.

### **Event-Driven Architecture and Publish/Subscribe Pattern**

A foundational pattern for building modern distributed systems is the Event-Driven Architecture (EDA) [451]. In an EDA, components interact asynchronously by producing and consuming events, which represent a change in system state such as Pump\_123\_Temperature\_Exceeded OR ProductionOrder\_Created. These events are exchanged through a communication infrastructure like a message broker, allowing producers and consumers to remain fully decoupled through a publish/subscribe (Pub/Sub) model. By replacing rigid, point-to-point communication with asynchronous event streams, EDA enables an environment where new analytics, monitoring, or control services can be added without disrupting existing operations.

However, while EDA provides the blueprint for decoupling, it does not, by itself, offer a concrete architectural model for implementation in a complex industrial setting. Several questions remain unanswered: How should events be structured for semantic interoperability between OT and IT systems? How is event-based governance and security managed across a multi-stakeholder ecosystem? How should the event distributing infrastructure itself be deployed and managed across the heterogeneous layers of the Edge, Fog, and Cloud? EDA defines the pattern, but a formal architecture is required to apply it effectively in practice. In the proposed architecture, EDA and Pub/Sub serve as core principles to replace rigid communication models, enabling flexible and dynamic interactions among system components.

#### **6.2.5 Summary**

The previous review of several related works, including standards, industry practices, academic proposals and related principles confirms that while each offers valuable perspectives, a significant gap remains. Formal standards like ISA-95 and RAMI 4.0 provide structural clarity but were not designed for the operational dynamics of cloud-native systems. Similarly, models like NOA and functional views from AIOTI contribute crucial concepts but remain abstract regarding runtime deployment, governance, and end-to-end lifecycle management. Therefore, no single model

offers a complete and implementable design for applications across the industrial continuum.

This analysis identifies several critical architectural drivers that a potential design must address: support for heterogeneity and legacy integration as in NOA, decentralized coordination for multi-factory and multi-stakeholder data spaces as envisioned by FIWARE, and to address cross-cutting concerns such as security and governance. In addition, concepts proposed by academic architecture such as LASFA, SITAM, or 8C Architecture should also be considered. “Finally, an effective design must consider industrial standards to ensure compatibility and practical applicability.

The review also points out that adopting cloud-native and event-driven paradigms such as EDA and Pub/Sub communication models is essential for enabling the flexibility and scalability required in modern industrial contexts. To address these requirements, a new reference architecture is proposed. The following sections detail the design methodology and present the resulting design.

## 6.3 Design Process and Methodology

This section presents in detail the process adopted for designing the proposed architecture. It also outlines the applied methodology and discusses the corresponding architectural drivers that guided the design process.

### 6.3.1 Design Methodology

This architectural design is guided by the *Attribute-Driven Design (ADD) 3.0* methodology [452], developed by the Software Engineering Institute (SEI) at Carnegie Mellon University. This is a systematic, iterative framework that emphasizes the central role of *quality attribute requirements (QARs)* such as scalability, modifiability, and security over functional requirements in designing system architecture. It ensures that stakeholder concerns, business goals, and system context drive the structure of the resulting architecture. The process involves several iterations, and in each iteration, architectural elements are refined to satisfy a prioritized set of QARs, while explicitly addressing key technical concerns and contextual constraints.

### 6.3.2 Architectural Drivers

Architectural drivers are the collection of inputs that collectively shape the design of the architecture [452]. In the ADD methodology, these drivers guide all design decisions. They consist of the system’s primary purpose and objectives, its quality goals, its required functionalities, the key technical concerns it must address, and the external constraints it must adhere to. The following subsections detail each of these drivers for the proposed reference architecture.

## Design Purpose and Objectives

The purpose of this design effort is to create a state-of-the-art reference architecture for a next-generation industrial platform. The primary objectives of this design are twofold:

- *Structure goal:* To create a unified and extensible reference architecture that resolves the structural limitations of traditional models like ISA-95 by introducing a decoupled, data-centric, and event-driven design.
- *Quality goal:* To enable operational robustness by treating critical quality attributes including security, governance, observability, and lifecycle management as foundational principles that are embedded in the design.

To summarize, these objectives aim to establish a reference architecture that not only bridges the gap between legacy industrial models and modern cloud-native principles but also provides an adaptable foundation for future industrial systems.

## Quality Attribute Goals

*Quality Attribute Goals* define the non-functional, qualitative properties that are most critical to the success of the system [452]. These goals directly influence the selection of architectural patterns. Four such goals have been defined as below:

- **QAR-1: Trustworthiness:** The system must be secure and enforce identity and data usage policies across all interactions in a distributed, multi-stakeholder environment.
- **QAR-2: Governance:** The system must support data sovereignty and comply with legal and business regulations through a auditable policy framework.
- **QAR-3: Observability:** The system's operational state must be fully transparent, providing high-quality telemetry to allow for effective operational monitoring, rapid root-cause analysis and continuous improvement.
- **QAR-4: Maintainability:** The architecture must support agile evolution of all system components through automated, version-controlled lifecycle management processes.

These quality attribute goals establish the foundation for a secure, transparent, and adaptable architecture. They ensure that the proposed system not only meets functional requirements but also maintains the essential non-functional properties needed.

## Primary Functional Requirements

*Primary Functional Requirements* define the specific behaviors and features the system must provide to deliver on its previously defined purpose [452]. Four primary requirements have been defined as follows:

1. **FR-1: Unified Asset Representation:** The system shall provide a central and stateful digital representation (Digital Twin) for every physical asset (e.g., pumps, robots) and logical asset (e.g., production lines, work orders) in the ecosystem, serving as the single source of truth for current and historical states. This requirement directly aligns with the central role of the FIWARE Context Broker. Furthermore, it reflects the RAMI 4.0 model, where the Digital Twin is a core element of the Asset layer. The explicit separation of Digital Twin types and instances also conforms to the IEC 62890 standard, which implements the RAMI's *Life Cycle & Value Stream* axis.
2. **FR-2: Edge-based Operational Intelligence (Low-Latency Analytics):** The system shall support the deployment of operational intelligence services as close to physical assets as possible. These services must perform real-time condition monitoring, anomaly detection, and closed-loop process optimization. This design choice is consistent with NOA's principle of performing M+O tasks at the edge. In addition, this approach addresses the *Edge* pillar of the CC as promoted by AIOTI, reducing latency and network overhead by processing data near its source.
3. **FR-3: Cloud-based Strategic Intelligence (High-Latency Analytics):** The system shall offer cloud-based intelligence services to support long-term business goals such as predictive maintenance, simulation, and enterprise-wide planning. Functionally, this corresponds to the capabilities found in Levels 3 (MES) and 4 (ERP) of the ISA-95 automation hierarchy. However, by utilizing a modern service-oriented model that consumes data from the Digital Twin Hub, the system avoids the data silos problem. This division of responsibilities between Edge and Cloud also reflects a widely adopted architectural pattern identified by Taxonomy previously.
4. **FR-4: Business Process Orchestration:** The system shall be able to coordinate complex, stateful, and long-running business processes. This capability is essential for enabling end-to-end automation—from high-level business events (e.g., *New Customer Order*) to factory-floor execution. Making orchestration a dedicated architectural component ensures a clean separation of concerns by isolating workflow logic from individual services. This improves modularity, maintainability, and overall system clarity by distinguishing the “what to do” (business intelligence) from the “how to do” (execution and coordination).

These functional requirements have been synthesized from the reviewed work in previous section. They define the system's essential capabilities for realizing a fully integrated and intelligent industrial platform by ensuring consistent asset representation, balancing intelligence services distribution between Edge and Cloud layers, and coordinating complex business processes.

## Architectural Concerns

*Architectural Concerns* are the key technical problems, questions, or goals that the architect must explicitly address and solve through the design [452]. They are the

primary drivers for the specific structural decisions made in the architecture and are defined as below:

- **AC-1 (Decomposition & Decoupling):** The central concern is to design a structure that avoids the rigid layer model of the ISA-95 pyramid. The architecture must solve the problem of data silos by creating a decoupled system of services. This will be addressed by adopting two main patterns. A *Hub and Spoke pattern* will be used for the intelligence and orchestration core, centered around a stateful hub that acts as the single source of truth for all asset representations. Communication among services will primarily be handled through a *Publish/Subscribe pattern* to ensure loose coupling and a flexible, non-hierarchical information flow.
- **AC-2 (Semantic Interoperability):** The architecture must solve the problem of heterogeneity at multiple levels, including legacy protocols and various data formats. This will be addressed by a two-part strategy. A dedicated adaptation layer will be designed to handle technical protocol translation for legacy assets. Downstream from this, a centralized semantic enrichment engine will be responsible for transforming all data into a common and semantically rich information model before it is used by the rest of the platform.
- **AC-3 (Secure OT/IT Convergence):** The architecture must address the critical security concern of safely extracting data from protected OT networks without exposing them to risks from the IT world. This will be addressed by designing a secure, layered gateway for all OT data, implementing the core idea of the NOA.
- **AC-4 (Operational Completeness):** The architecture must explicitly address the cross-cutting concerns required to run the system efficiently. This will be addressed by defining a set of *Foundational Pillars*. This ensures that security, compliance, observability, and agility are core qualities embedded in every component in the system.

By systematically addressing these issues, the architecture establishes a cohesive foundation that balances flexibility with reliability, ensuring that the design is both conceptually sound and practically implementable within industrial environments.

## Constraints

*Constraints* are non-negotiable decisions, requirements, or limitations that are imposed on the architecture and must be adhered to [452]. They define the fixed boundaries of the design space. The architecture must operate under the following constraints:

- **CON-1 (Hybrid Environment & Brownfield Support):** The architecture must be designed to operate across a hybrid Compute Continuum environment. It must also support the integration of existing legacy brownfield assets. This constraint is fundamental, as it dictates the need for an adaptation layer to bridge the OT/IT divide, directly aligning with the pragmatic approach of NOA.

- **CON-2 (Cloud-Native & Open Standards Mandate):** The implementation of this architecture must use cloud-native technologies. To mitigate vendor lock-in and ensure interoperability, the design must prioritize the use of open standards wherever feasible (e.g., OPC UA, MQTT, and standards from data space ecosystems).
- **CON-3 (Standards Alignment & Interoperability):** The design must be conceptually compatible with and mappable to established industry frameworks to ensure a common language with the broader ecosystem. Specifically, it must align with: (1) *RAMI 4.0*: For its structural map of Industrie 4.0 components; (2) *IEC 62890*: For its formal Type/Instance model, which will be implemented in the Asset Representation Services. (3) *Data Space Principles*: The Data Space Interface must be designed for compatibility with emerging European data sovereignty frameworks like Gaia-X [453] and its industrial implementation in the automotive sector, Catena-X [454].

These constraints ensure that the design remains both practically deployable (through hybrid and brownfield compatibility) and technically future-proof (by adhering to cloud-native principles and open standards). In addition, alignment with well known industrial models guarantees conceptual consistency and interoperability with the broader Industry 4.0 ecosystem.

## 6.4 The Proposed Reference Architecture

After the architectural drivers have been defined, this section presents the main components of the proposed architecture.

### 6.4.1 Architectural Overview

The proposed Cloud-native Reference Architecture for the Compute Continuum in Industry 4.0 (CRACI), presented in Figure 6.7, is designed to satisfy the architectural drivers identified in the previous section. It follows a service-oriented and data-centric approach, deliberately moving away from the rigid hierarchies of traditional models.

The primary organizing principle is its horizontal axis, representing the physical or logical deployment location of each component. From left to right, the architecture progresses from distributed data sources to centralized cloud resources:

- *Shop Floor (Far Left)*: Contains physical assets and platform services (e.g., Adaptation Services, Edge Platform Services). Operational Intelligence Services can also be deployed here to meet low-latency requirements for real-time monitoring and control.
- *Regional Core (Middle)*: Hosts services required for consistency, such as Asset Representation Services (Digital Twin Hub), and Orchestration Services.
- *Cloud (Far Right)*: Contains compute-intensive, non-latency-sensitive services like Strategic Intelligence Services, model training, and global optimization.

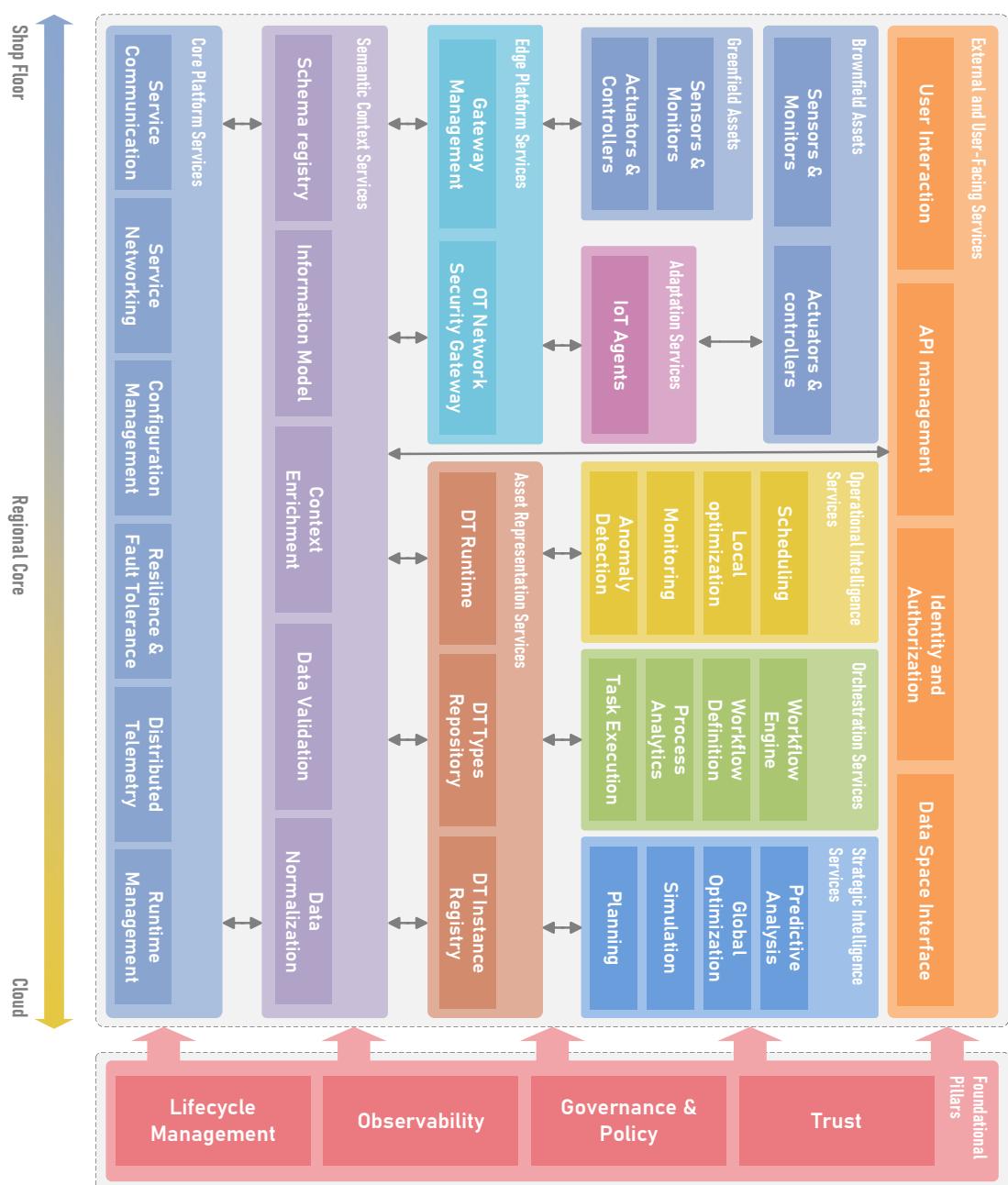


FIGURE 6.7: Cloud-native Reference Architecture for the Compute Continuum in Industry 4.0 (CRACI).

All external interactions are managed through the *External & User-Facing Services*, which span from the physical device level to the central cloud, meaning external integration can happen anywhere. The system is built on domain-agnostic *Core Platform Services* and governed by four Foundational Pillars: *Trust, Governance & Policy, Observability, and Lifecycle Management*. These pillars enforce system-wide quality attributes for all components, regardless of deployment location.

### The Four Foundational Pillars

The Foundational Pillars define the underlying principles that shape the architecture's design. They correspond to four identified QARs mentioned earlier and represent important system-wide concerns.

**Trust (QAR-1)** In a federated ecosystem where independently managed domains collaborate, establishing mutual trust among users, services, and devices is essential. Local trust domains form the basis for integration into federated trust frameworks such as Gaia-X, where verifiability and transparency are prerequisites for participation. Every interaction must be authenticated and authorized, enforced through strong Identity and Access Management (IAM), encryption of data at rest and in transit, and immutable audit logging for accountability.

**Governance and Policy (QAR-2)** This pillar defines mechanisms that keep the system compliant with defined rules, regulations, and stakeholder agreements. Governance encompasses data sovereignty, legal compliance (e.g., EU General Data Protection Regulation (GDPR)), and business logic enforcement. Policies define rules for data retention, access control, and privacy protection. These policies are enforced at runtime to maintain consistent behavior across distributed services. Governance mechanisms also enable participation in regulated data spaces, aligning system behavior with external trust, reputation, and compliance requirements, as highlighted in Gaia-X [453] and Catena-X [454] initiatives.

**Observability (QAR-3)** Observability allows operators to understand system behavior by collecting and analyzing telemetry data, including logs, metrics, and traces, from all components. It enables rapid root-cause analysis, system health assessment, and continuous performance optimization.

**Lifecycle Management (QAR-4)** Lifecycle Management governs the lifecycle of all artifacts such as software, infrastructure, and ML models, through automated, version-controlled processes. It unifies modern cloud-native practices, integrating DevOps and MLOps for streamlined development and operations. Using Infrastructure-as-Code (IaC), all configurations are codified and managed under version control. CI/CD pipelines automate the build, release, and run stages, while GitOps workflows enable declarative, pull-based deployment across the continuum. This pillar promotes agility, reproducibility, and traceability throughout the platform's lifecycle.

## 6.4.2 Functional Breakdown of Architectural Components

This section provides a detailed description of each CRACI component. They are presented in the order of the architecture's data value chain, beginning with data generation at physical assets and proceeding through transformation into information, state, and action.

### Physical Assets (Brownfield & Greenfield)

*Physical Assets* are the source of all operational data within the architecture. They represent the physical machines and control systems on the factory floor. The reference architecture differentiates two classes of assets:

- *Brownfield Assets*: This component represents the existing, legacy industrial equipment. These assets are often critical to production but were designed before the prevalence of modern networking standards, communicating via proprietary or serial protocols (e.g., Modbus RTU/TCP, PROFIBUS, DeviceNet, analog signals (4-20mA), and digital I/O). Their integration is a primary architectural challenge that requires a dedicated adaptation strategy. They are often isolated on non-routable OT networks and may require physical wiring.
- *Greenfield Assets*: This component represents modern, IIoT-ready industrial equipment designed with native IP-based connectivity and support for standard, interoperable protocols (e.g., OPC UA, MQTT, AMQP, REST APIs.). These devices are normally connected via Ethernet-based networks and can be integrated into standard IP networks. Although integration is less complex, it must follow controlled procedures.

Both categories are further composed of two types of sub-components:

- *Sensors & Monitors*: These are the data sources, responsible for perceiving the physical world. These include everything from simple temperature sensors to complex, high-resolution cameras.
- *Actuators & Controllers*: These are responsible for acting upon the physical world. This includes Programmable Logic Controllers (PLCs), robot arms, servo drives, or valves that execute commands received from the control and orchestration layers.

By explicitly differentiating between greenfield and brownfield assets, CRACI can be applied both to the provisioning of new systems and to the integration of existing ones. This design decision directly implements **CON-1**.

### Adaptation Services

*Adaptation Services* represent components designed specifically to address the integration challenges related to brownfield assets. Their primary function is to act as a translation layer, enabling legacy industrial devices to behave like modern, protocol-compliant endpoints within the broader platform (**AC-2** and **CON-1**). These services are implemented with *IoT Agents*, which are specialized, stateless, and highly configurable software modules. Each agent is designed to handle a

specific industrial protocol. These agents perform real-time protocol translation by converting different formats into the platform's standardized messaging schema, typically MQTT messages with structured JSON payloads, before forwarding the data upstream. This abstraction enables seamless interoperability across heterogeneous environments and supports incremental implementations.

### Edge Platform Services

This component functions as a single governed entry point for all physical assets, supporting the integration of both natively connected Greenfield assets and adapted Brownfield assets. The two main functions provided by this component are:

- *Gateway Management*: This service manages the hardware at the edge, including Industrial PCs, field gateways, and other edge nodes. It automates the complete operational lifecycle, from zero-touch provisioning for initial deployment, remote configuration for ongoing adjustments, to automated deployment of containerized services. This service is therefore crucial for secure and continuous operation of the edge component.
- *OT Network Security Gateway*: This component enforces a secure boundary between the protected OT network and the enterprise IT infrastructure (**AC-3**). It implements robust network security mechanisms such as firewalls, intrusion detection and prevention systems (IDS/IPS), and VPN termination. Its primary function is to prevent unauthorized access to sensitive legacy systems while allowing governed data extraction. Moreover, it serves as the controlled interface through which commands can be sent back into the OT domain, enforcing strict verification and authorization protocols, similar to the *Verification of Request* principle in NOA. This permits only trusted and authenticated requests to affect critical control systems, preventing attacks such as the well-known Stuxnet [455].

The *Edge Platform Services* component serves as a secure gateway connecting the OT and IT domains, directly implementing **AC-3**.

### Semantic Context Services

The *Semantic Context Services* component ingests context-poor but structured data streams from the Edge and transforms them into rich and semantically meaningful information. This process enables higher-level intelligence to operate on fully enriched data. The functionalities provided by this component are:

- *Schema Registry*: A version-controlled, authoritative repository of all data schemas used across the platform. It enforces structural consistency by serving as the single source of truth for data formats associated with different asset types.
- *Data Validation*: The initial processing stage, where incoming messages are validated against the corresponding schemas in the registry. This guarantees structural integrity and prevents malformed data from entering the system.

- *Data Normalization*: A dedicated service that standardizes incoming data to a unified internal format. Responsibilities include critical transformations such as converting all units to a common baseline (e.g., pressure to bar, temperature to Celsius), which maintains consistency for downstream analytics.
- *Information Model*: A centralized semantic graph that maintains the relationships, hierarchies, and metadata of all physical and logical assets. It captures knowledge such as “Sensor S1 is part of Motor M2, which belongs to Pump P3 on Line B,” and stores asset metadata like manufacturer, installation date, and documentation links.
- *Context Enrichment*: The final and most important function in the semantic enrichment pipeline. This service augments validated and normalized data by querying the Information Model and attaching relevant relational and business context. For instance, a basic message like `{"deviceId": "SN-123", "value": 50}` is enriched into a fully contextualized event: `{"assetId": "Pump-123", "metric": "temperature", "value": 50, "unit": "Celsius", "line": "B"}`.

The *Semantic Context Services*, together with the *Adaptation Services*, implement the concern **AC-2** and directly address one of the key challenges in CC-based systems, namely interoperability.

## Core Platform Services

*Core Platform Services* component provides the essential runtime environment required for the operation of a cloud-native system. It implements key principles from the Twelve-Factor App methodology. Here, several functionalities are provided:

- *Service Communication*: Provides the asynchronous, event-driven communication backbone for the system (for example, a publish/subscribe event bus). It ensures loose coupling between services by decoupling message producers from consumers and supporting event-driven interaction patterns.
- *Service Networking*: Manages networking-related tasks across services. This includes service discovery and dynamic service registration, as well as load balancing and related functions.
- *Configuration Management*: Offers a centralized and secure method for managing service configuration, in line with the Twelve-Factor principle of externalized configuration. It provides version-controlled storage for application settings and integrates secret management systems for handling sensitive credentials.
- *Resilience & Fault Tolerance*: Implements essential reliability patterns, such as circuit breakers, retries, and timeouts, so that the system can recover gracefully from partial failures. This suite of capabilities protects against cascading failures and contributes to system robustness under degraded conditions.
- *Distributed Telemetry*: Collects, aggregates, and exposes logs, metrics, and distributed traces from all services. Logs are treated as event streams, consistent

with modern logging practices, and telemetry data is used for diagnostics, alerting, and performance optimization.

- *Runtime Management*: Acts as the runtime control plane for all containerized workloads. It provides automated deployment, scaling, and lifecycle management.

The *Core Platform Services* component provides the essential infrastructure for the cloud-native architecture, implementing **AC-1** and **CON-2**. This component will be discussed in more detail in Chapters 7 and 8.

### Asset Representation Services

The *Asset Representation Services* component functions as the data repository for all Digital Twins, implementing functional requirement **FR-1**. Located at the core of the architecture, it consumes semantically enriched information and maintains a persistent, stateful model of assets and processes. This hub decouples higher-level logic from data ingestion pipelines and serves as the primary interface for intelligence and orchestration services. The main functionalities provided by this component are:

- *DT Types Repository*: A design-time catalog for managing templates of asset types. It implements the *Type* concept from IEC 62890, capturing shared properties, structures, and capabilities for each class of assets (e.g., all Model-X Pumps) (**AC-1** and **CON-3**).
- *DT Instance Registry*: A registry of all instantiated Digital Twins, each mapped to a unique physical asset. It governs the lifecycle states (e.g., created, commissioned, decommissioned) and links each instance to its corresponding DT Type, implementing the *Instance* concept of IEC 62890 (**CON-3**).
- *DT Runtime*: The runtime engine that holds the live state of each Digital Twin. It continuously ingests contextualized data streams to update twin attributes and exposes standardized APIs (e.g., REST or NGSI-LD) for state queries or event subscriptions. To avoid a single point of failure, this component must be implemented as a distributed store. Consequently, its design must address the trade-offs defined by the CAP Theorem [456]. Given the varied requirements of industrial control, the runtime must support both *strong consistency* for critical, time-sensitive state data and *eventual consistency* for less critical data to balance correctness with performance and availability.

The *Asset Representation Services* implement the Digital Twins, one of the key enabling technologies of Industry 4.0. These Digital Twins are linked to physical assets, forming a bidirectional connection between the real world and their virtual representations. This connection enables manufacturers to monitor, simulate, and optimize operations throughout the entire product lifecycle.

### Operational Intelligence Services

The *Operational Intelligence Services* component provides low-latency analytics and control applications designed to operate at the Edge. These services enable immediate,

localized responses to changes in the physical environment and support time-critical decision-making. This implements the requirement **FR-2**, identified in the previous design step, and consists of the following functionalities:

- *Edge-based Scheduling*: Handles short-term scheduling and coordination of edge-local resources such as Autonomous Mobile Robots (AMRs), collaborative robots, or production lines within a constrained time window.
- *Local Optimization*: Implements closed-loop control logic to optimize operations of individual machines based on real-time inputs.
- *Condition Monitoring*: Continuously monitors the real-time operational parameters of assets and compares them to detect early signs of faults.
- *Anomaly Detection*: Analyzes high-frequency sensor data streams to detect statistically or heuristically significant deviations from normal behavior.

Consistent with the previously developed taxonomy, CRACI differentiates between latency-sensitive and non-latency-sensitive services. This differentiation allows for service placement according to latency requirements. The *Operational Intelligence Services* primarily includes latency-sensitive services.

### Orchestration Services

The *Orchestration Services* component coordinates stateful, long-running business processes across distributed services. It executes actions rather than making decisions. This component implements Functional Requirement **FR-4**, and provides the following functionalities:

- *Workflow Engine*: The core runtime that interprets and executes business process models, enabling the dynamic orchestration of services across domains.
- *Workflow Definition Repository*: A version-controlled repository for business process definitions, typically expressed in industry-standard formats such as Business Process Model and Notation (BPMN).
- *Process Analytics*: A monitoring and observability layer that tracks key metrics and execution states of workflows, supporting root-cause analysis and performance optimization.
- *Task Execution*: Worker modules responsible for executing individual workflow steps.

The *Orchestration Services* component orchestrates the workflows that exist within the system. An example of such a service is discussed in more detail in Parts V and VI of this dissertation.

### Strategic Intelligence Services

The *Strategic Intelligence Services* component provides support for long-term analytics and data-driven decision-making. These services are typically deployed in cloud

environments and operate over large, historical datasets to support enterprise-level strategy and optimization. This component implements Functional Requirement **FR-3**.

- *Predictive Analysis:* Executes advanced machine learning models trained on historical data to anticipate future system states or failures (e.g., predictive maintenance, demand forecasting).
- *Global Optimization:* Performs enterprise-wide optimization tasks such as production planning, logistics coordination, or energy efficiency improvements.
- *Simulation:* Runs simulations to evaluate scenarios, policy changes, or control strategies before deployment.
- *Planning:* Supports strategic and tactical planning processes by aggregating insights from historical trends, business KPIs, and analytical models to inform executive decisions.

In contrast to the *Operational Intelligence Services*, which are mostly time-sensitive, the *Strategic Intelligence Services* are non-time-sensitive and focus on long-term analyses that require larger amounts of data and greater computational resources.

### External and User-Facing Services

The *External and User-Facing Services* component is responsible for communicating with other systems. This component provides several core functionalities, which are discussed in detail below.

- *API Management Gateway:* Serves as the programmatic entry point for all machine-to-machine interactions, including communication with enterprise systems (e.g., Enterprise Resource Planning (ERP)) and partner applications.
- *Identity and Authorization Services:* This component implements the Trust pillar by managing digital identities for users and machine clients. It supports secure authentication methods and authorization, and provides identity-management capabilities for administrative and user-facing tasks.
- *User Interaction Services:* Delivers user experiences through interactive dashboards and administrative tools.
- *Data Space Interface:* A specialized gateway that enables sovereign, secure, and standards-compliant data exchange with external entities in federated ecosystems such as Gaia-X and Catena-X. It supports protocols from the International Data Spaces Association (IDSA) and enforces digital usage contracts via cryptographic means to ensure policy-compliant data consumption.

The *External and User-Facing Services* act as the secure, managed boundary that controls all interactions between the internal platform and external actors. Its primary goal is to expose the system's data in a policy-compliant manner through both human-facing and programmatic interfaces.

## 6.5 Architectural Design Rationale

The proposed reference architecture is the result of a systematic design process that synthesized best practices from previously reviewed related work while explicitly addressing their limitations in the context of a modern, cloud-native Compute Continuum. This section provides the justification for key architectural decisions by demonstrating how CRACI's design aligns with and extends the principles of ISA-95, RAMI 4.0, NOA, AIOTI, FIWARE, LASFA, SITAM, and 8C Architecture.

### 6.5.1 Decoupling the ISA-95 Hierarchy

As defined in the design objectives, the primary goal of the CRACI reference architecture is to resolve core limitations of the ISA-95 model. This was achieved through two design elements, which are *Hub-and-Spoke model* and *Event-Driven communication*, addressing identified gaps:

- *Replacing the Rigid Hierarchical Model:* Instead of a vertically layered structure with limited flexibility, CRACI uses a pub/sub communication model to facilitate cross-layer communications. This design enables flexible information flows, analogous to the *data highway* concept proposed by NOA or the specialized data paths in FIWARE. This enables a more adaptable, scalable, and responsive design.
- *Enabling Distributed Intelligence at the Edge:* To overcome ISA-95's centralization of intelligence, CRACI distributes decision-making capabilities across the CC. The intelligence services can be deployed across the continuum, from the low-latency edge (*Operational Intelligence Services*) to the central cloud (*Strategic Intelligence Services*).
- *Supporting Cloud-Native and Cross-Domain Integration:* ISA-95 lacks mechanisms for cloud-edge collaboration or inter-organizational processes. CRACI implements cloud-native principles through the *Core Platform Services* and integration interfaces like the *Data Space Interface*. These enable scalable orchestration, secure API exposure, and standards-based data sharing across domains (e.g., Gaia-X or IDS-based ecosystems).
- *Eliminating Access and Data Silos:* The proposed reference architecture systematically overcomes the data silos inherent in the ISA-95 pyramid by implementing a data-centric model. Instead of forcing data through rigid hierarchical layers, all contextualized information is channeled into a central *Asset Representation Services* hub, which acts as a single source of truth. This structure, combined with an event-driven communication infrastructure, allows any authorized service to access data directly, eliminating traditional data silos.
- *Supporting Analytics Workloads:* The CRACI architecture addresses the analytical shortcomings of ISA-95 by separating the real-time operational path from the historical analytics path. This is enabled by a decoupled event bus, which allows the same data stream to be consumed in parallel by multiple specialized services. The Asset Representation Services subscribe to the stream to manage

live, transactional state (an OLTP workload). Simultaneously, a data sink service archives the stream into a long-term analytical store. This allows the Strategic Intelligence Services to perform high-volume, historical analysis (an OLAP workload) without degrading the performance of the real-time system.

In summary, the CRACI reference architecture overcomes ISA-95 limitations by applying cloud-native and event-driven principles.

### 6.5.2 Extending RAMI 4.0 with Operational Aspects

The CRACI architecture provides the missing operational aspects of the RAMI 4.0 model by incorporating several structural patterns:

- *Lifecycle Management as Runtime Semantics*: RAMI 4.0 introduces a *Life Cycle & Value Stream* axis as a conceptual timeline. CRACI transforms this axis into an operational capability embedded in the Lifecycle Management pillar. Furthermore, the Asset Representation Services implement the IEC 62890 *Type/Instance* distinction through the DT Types Repository and DT Instance Registry, enabling structured asset lifecycle governance.
- *Support for Cross-Cutting Concerns*: Whereas RAMI 4.0 references external models for security, observability, and governance, the CRACI architecture embeds these directly as foundational pillars (*Trust, Governance & Policy, Observability, and Lifecycle Management*). These pillars ensure both quality attributes and cross-cutting concerns embedded to every component.
- *Revising Hierarchical Assumptions*: RAMI 4.0's *Hierarchy Levels* axis may suggest rigid, layered data flows. In contrast, the CRACI architecture adopts *Hub-and-Spoke* and *Event-Driven* communication patterns to support dynamic, bidirectional, and cross-layer interactions, as detailed earlier in the design.
- *Formalizing External Systems Integration*: The *External and User-Facing Services*, specifically the *Data Space Interface*, formalize external integration by supporting standards from ecosystems like the International Data Spaces Association (IDSA) and Gaia-X. This design provides a foundation for addressing data sovereignty and compliance requirements, enabling secure and policy-driven collaboration in modern industrial networks.

In summary, the CRACI architecture can be viewed as a more concrete implementation and extension of the RAMI 4.0 model, with a focus on applying cloud-native principles and enabling inter-organizational structures.

### 6.5.3 Completing the NOA Concept

The proposed reference architecture implements the goals of the NOA while extending its concepts in a more complete design. Each of the following design choices directly addresses a core limitation of NOA:

- *Structuring the Abstract M+O Domain*: CRACI provides this missing structure by decomposing the M+O domain into clearly defined, distributed services. It

explicitly defines *Operational Intelligence Services* for low-latency analytics at the edge and *Strategic Intelligence Services* for long-term optimization in the cloud, with their complex interactions managed by the dedicated *Orchestration Services* component.

- *Extending NOA from a Complementary Pattern to a Full System:* The CRACI architecture advances the NOA concept by presenting a unified design for building a complete industrial system from the ground up. By supporting both brownfield and greenfield assets, its design keeps NOA's non-intrusive principles while emphasizing the boundary between IT and OT through the use of an *OT Network Security Gateway*, functionally similar to the Verification of Request component in NOA.

In summary, CRACI addresses the main limitations of NOA by providing a concrete architectural design for IT services. At the same time, it retains and extends the key idea of NOA about a secure gateway connecting OT and IT domains.

#### 6.5.4 Generalizing the FIWARE Architecture

The proposed reference architecture operates at a higher level of abstraction and incorporates runtime design principles missing in FIWARE:

- *Reducing Ecosystem Lock-In through a Technology-agnostic approach:* The proposed architecture avoids ecosystem lock-in by abstracting patterns such as the Context Broker into the *Asset Representation Services* component. This enables the same publish/subscribe and data decoupling benefits without requiring strict adherence to NGSI-LD or a fixed set of enablers as found in FIWARE. Developers are free to implement the architecture using popular cloud-native tools (e.g., Kafka, Redis, or custom APIs).
- *Systematic Structuring of Cross-Cutting Concerns:* The CRACI architecture explicitly incorporates *Trust, Governance & Policy, Observability, and Lifecycle Management* as foundational pillars.
- *Orchestration Across the Continuum:* The architecture includes components like the *Orchestration Services* and the *Runtime Management* layer in the *Core Platform Services*, which manage the microservice lifecycle, enforce runtime policies, and enable service portability at runtime in a distributed environment.

In contrast to the FIWARE Foundation, which has a strong incentive to promote the use of technologies within its own ecosystem, CRACI provides a cloud-native, technology-agnostic industrial architecture.

#### 6.5.5 Implementing the AIOTI Functional Models

The CRACI architecture provides the necessary structure, demonstrating how all of AIOTI's required functions are logically positioned and implemented. The functional views of AIOTI are embedded in the design as follows:

- *Data Management and Artificial Intelligence:* *Data Management* is implemented through the *Semantic Context Services* (validation, normalization, enrichment) and the *Asset Representation Services* (Digital Twin models). *Artificial Intelligence* is realized via the *Operational Intelligence Services* for low-latency edge analytics and *Strategic Intelligence Services* for enterprise-wide, long-term prediction and optimization.
- *Resource Management and Networking:* These functions are fulfilled by the *Core Platform Services* and *Edge Platform Services*, which manage compute, storage, connectivity, and telemetry within the whole system.
- *Orchestration:* The AIOTI Orchestration function is implemented through the *Orchestration Services* component, which supports stateful process execution, workflow modeling, and task distribution. This separation allows intelligent components to focus on decision-making while the orchestration layer ensures coordinated and reliable execution.
- *Security & Privacy, Trust & Reputation, Monitoring & Observability:* Rather than treating these as peer-level functional blocks, CRACI elevates them into cross-cutting *Foundational Pillars: Trust, Governance & Policy, and Observability*. This design embeds critical concerns such as identity management, compliance, data sovereignty, and runtime telemetry in every component of the system, not just at isolated points.

In summary, the CRACI architecture can be regarded as a concrete implementation of the AIOTI Functional Model, providing less abstract and more practical guidance for building a modern Industrial Compute Continuum.

### 6.5.6 Improving LASFA’s Data Architecture

The proposed CRACI reference architecture can be seen as the operational implementation of the conceptual model presented by LASFA.

- *From Functional Layout to Operational Design:* LASFA’s primary contribution is its two-dimensional visualization of where Digital Twins and Digital Agents are located. The proposed architecture builds on this functional view by defining how these components must operate, governed by the Foundational Pillars. Instead of merely showing component locations, CRACI mandates that each must be trustworthy, observable, governed by policy, and managed through an automated lifecycle.
- *Evolving Data Architecture from Nested Clouds to a Central Hub:* LASFA proposes a decentralized data structure with nested “Local Clouds” for each process or production line. CRACI refines this into a structured Hub-and-Spoke model, where all semantically enriched data is consolidated into a central *Asset Representation Services* (the hub). This creates a single source of truth that decouples all intelligence services (the spokes) and avoids inconsistencies across local clouds.

In summary, while LASFA provides an intuitive visualization of a smart factory's functional layout, CRACI extends this by defining the underlying operational principles and engineering patterns required to build, deploy, and govern such a system at scale.

### 6.5.7 Formalizing the Cross-Cutting Concerns of the SITAM

The proposed reference architecture extends the foundational concepts of the SITAM by formalizing its cross-cutting concerns into governing principles and updating its Service-Oriented Architecture (SOA) pattern for a fully cloud-native, operational environment.

- *Formalizing Cross-Cutting Concerns into Foundational Pillars:* SITAM identifies “Cross-Architectural Topics” such as SOA Governance, Data Quality, and Security & Privacy. CRACI extends this by elevating these concerns into four explicit *Foundational Pillars*, which are Trust, Governance & Policy, Observability, and Lifecycle Management.
- *Evolving the Integration Model from ESB to a Cloud-Native Event Bus:* SITAM relies on a hierarchical Enterprise Service Bus (ESB) structure as a classic SOA integration pattern. CRACI modernizes this approach with a more flexible, non-hierarchical, and event-driven communication model implemented via its *Core Platform Services*. This replaces the ESB with a decoupled event bus and incorporates cloud-native runtime management (e.g., container orchestration), making it better suited for dynamic and scalable distributed systems.
- *Expanding Analytics into Distributed Intelligence:* SITAM introduces an *Analytics Middleware* centered on a *Manufacturing Knowledge Repository*. CRACI extends this by distributing intelligence across the compute continuum. It defines *Operational Intelligence Services* for low-latency analytics at the edge and *Strategic Intelligence Services* for long-term analytics in the cloud, both drawing from the central *Asset Representation Services* hub.

In summary, while SITAM proposes the data-driven factory concept, CRACI extends it with the operational and governance semantics required for modern, distributed industrial systems.

### 6.5.8 Implementing the Integration Layers of 8C Architecture

The proposed reference architecture provides a more formal and operationally complete framework for the concepts of horizontal and lifecycle integration introduced by the 8C Architecture.

- *Implementing Horizontal Integration with Trust and Governance:* The 8C architecture introduces the Coalition and Customer facets for cross-company and customer integration. CRACI implements this through the *External and User-Facing Services*, governed by the *Trust and Governance & Policy* pillars. This provides concrete mechanisms for managing federated identities and ensuring data sovereignty in multi-stakeholder ecosystems.

- *Implementing Content with Lifecycle Management:* The *Content* facet in the 8C model emphasizes the need for product traceability records. CRACI implements this via the *Asset Representation Services*, serving as a single source of truth. Moreover, the *Lifecycle Management* pillar provides an automated framework for building, deploying, and maintaining the services.
- *From Facets to System-Wide Pillars:* While the 8C model introduces its “3C” facets as cross-cutting concerns, CRACI formalizes these into four *Foundational Pillars*. These are non-negotiable, system-wide principles, which are not explicitly addressed in the 8C model.

In summary, while the 8C model’s addition of Coalition, Customer, and Content facets identifies the limitations of vertical models, CRACI implements these ideas using concrete engineering patterns.

A summary of the discussions above among CRACI and related models is given in Table 6.1

TABLE 6.1: Feature Comparison of Industry 4.0 Architectures.

Architectural Feature	ISA-95 [430]	RAMI 4.0 [435]	NOA [441, 442]	FIWARE [439, 440]	AIOTI Model [443]	LASFA [444]	SITAM [445]	8C Architecture [446]	Proposed CRACI
Formal Industry Standardization	✓	✓	—	—	—	—	—	—	—
Industry-driven Operational Model	—	—	✓	✓	✓	—	—	—	—
Academic Proposal	—	—	—	—	—	✓	✓	✓	✓
Cloud-Native Principles	—	—	—	○	○	○	○	○	✓
Decoupled / Event-Driven	—	✓	○	✓	○	○	✓	✓	✓
Secure IT/OT Integration	○	○	✓	○	—	—	○	—	✓
Cross-Domain and Ecosystem Integration	—	✓	—	○	○	○	✓	✓	✓
Cross-Cutting Concerns	—	○	—	—	✓	—	✓	○	✓
Brownfield Support	✓	○	✓	○	—	—	○	—	✓
Data Sovereignty / Federated Support	—	○	—	✓	✓	—	—	○	✓
Human-Centric Design	—	—	—	—	○	○	✓	○	○
Concrete Implementation Blueprint	—	—	○	✓	—	✓	✓	○	✓

**Legend:** ✓ Fully Addressed; ○ Partially Addressed; — Not Addressed.

## 6.6 Chapter Summary

This chapter addressed the gap between traditional industrial automation models and the demands of modern systems. The review of industrial standards like ISA-95, IEC 62890, and RAMI 4.0, alongside operational frameworks like FIWARE and NOA,

indicates that although each provides valuable aspects, no single model offers a complete blueprint for designing, deploying, and managing applications across the CC. Existing models are either too hierarchical, too abstract, or too prescriptive.

Addressing this gap, this chapter introduced the Cloud-native Reference Architecture for the Compute Continuum in Industry 4.0 (CRACI). Developed using the systematic ADD 3.0 methodology, CRACI is designed to be modular and extensible. The key novelty of this design is the use of cloud-native and event-driven communication principles, which resolve data silos and enable flexible, cross-domain service interaction.

Another important aspect of CRACI is the integration of *Foundational Pillars*, which are *Trust*, *Governance & Policy*, *Observability*, and *Lifecycle Management*, as crucial architectural principles. This integration makes critical quality attributes systematically embedded in every component of the system, providing a robust foundation for secure, compliant, and agile operations. As demonstrated in the design rationale, CRACI synthesizes the structural clarity of RAMI 4.0 and the pragmatic integration strategy of NOA, while providing the operational and runtime semantics missing in more abstract functional models like AIOTI. In addition, unlike more prescriptive frameworks like FIWARE, which promote a specific technology stack, CRACI is intentionally technology-agnostic. It also translates the conceptual contributions of academic proposals such as the functional layouts of LASFA, the human-centric SOA principles of SITAM, and the horizontal integration facets of the 8C architecture into a concrete engineering design. It defines architectural patterns and capabilities, aligning with cloud-native principles, but does not require a fixed set of tools, thus serving as a reference architecture that avoids ecosystem lock-in and offers implementation flexibility.

This chapter has detailed the architectural design of CRACI and the justification for it. The following chapter will validate CRACI by presenting MAIA (Microservices-based Architecture for Industrial Analytics) as a concrete implementation. It will also demonstrate how CRACI's abstract components and principles translate into an operational system, demonstrating feasibility and effectiveness for modern industrial systems.

# 7 Validation of CRACI: A Case Study in Industrial Analytics

## 7.1 Introduction to the MAIA Use Case

To demonstrate the practical value of the CRACI reference architecture, this chapter analyzes a concrete implementation: MAIA (Microservices-based Architecture for Industrial Data Analytics), partly published in the peer-reviewed work [457]. The MAIA system introduces a decentralized, microservices-based architecture designed specifically to address the challenges of low-latency data analytics in modern industrial environments. The deployment scenario involves a dynamic shop floor divided into several *edge-enabled zones*. Each zone is equipped with a dedicated MEC (Multi-access Edge Computing) server and a private 5G gNodeB base station, providing Ultra-Reliable Low-Latency Communication (uRLLC). Figure 7.1 provides an overview of the scenario.

The underlying infrastructure supports a fleet of *Autonomous Mobile Robots* (AMRs) that handle complex logistics. To maintain real-time situational awareness and optimize power consumption, the AMRs offload a computationally demanding perception pipeline to the edge. The containerized perception service processes camera streams to identify obstacles and navigation points. This task is bound by a strict low-latency Service Level Objective (SLO) to support safe operation.

A critical failure point occurs during inter-zone mobility. Consider the AMR moving from Zone A (served by MEC-A) to Zone B (served by MEC-B). A naïve migration strategy, where the system reacts only after the handover, would introduce significant service interruptions. This temporary loss of situational awareness could lead to abrupt stops or unsafe navigation, violating operational SLOs. This scenario highlights the need for a proactive migration strategy for the offloaded services.

To maintain service continuity, the MAIA architecture implements a proactive migration pipeline. The process is orchestrated by the centralized control plane as follows:

1. *Path Prediction*: The process begins with the AMR continuously streaming its *location data* through the 5G gNodeB to the *Path Prediction Service* within the *Control Plane*. By analyzing this stream, the service predicts the robot's trajectory and anticipates a zone transition.
2. *Intelligent Orchestration*: When a handover is predicted, the *Path Prediction Service* sends a *Migration Recommendation* to the *Service Migration Orchestrator*. Acting on this, the Orchestrator initiates a proactive preparation sequence before the robot physically moves into the new zone:

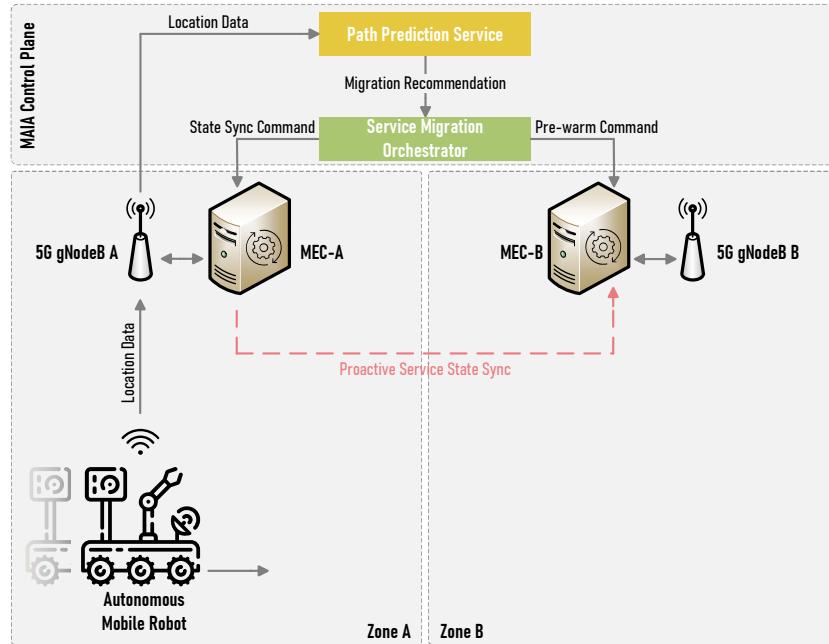


FIGURE 7.1: MAIA use case: Orchestration of service handover for mobile robotics.

- First, it issues a *Pre-warm Command* to MEC-B, instructing it to prepare a *warm* instance of the required service, eliminating the *cold-start* delay that would otherwise cause service interruption.
  - Simultaneously, it sends a *State Sync Command* to the currently active service running on MEC-A.
3. *Proactive State Synchronization:* The *State Sync Command* triggers the *Proactive Service State Sync* between the two edge servers. The live operational state of the service on MEC-A is continuously replicated to the warm instance on MEC-B.

By the time the AMR reaches the boundary of Zone B, a fully synchronized and ready-to-run service instance is already waiting for it on MEC-B. This allows the system to immediately redirect the robot's workload to the new server with minimal downtime. This preemptive approach transforms a potentially disruptive migration into a transparent background operation. The control plane shown in Figure 7.2 will be implemented by MAIA, a concrete implementation of the previously presented reference architecture CRACI.

## 7.2 Mapping MAIA Architecture to the Reference Architecture

To validate the CRACI design, this section maps its conceptual components to a concrete implementation in MAIA. Figure 7.2 provides a high-level overview of this alignment, with the implemented MAIA components highlighted as colored blocks. A detailed one-to-one mapping is presented in Table 7.1.<sup>1</sup>

<sup>1</sup>Note: The table layout (spanning across two pages) was assisted by an AI model (Google Gemini 2.5 Pro).

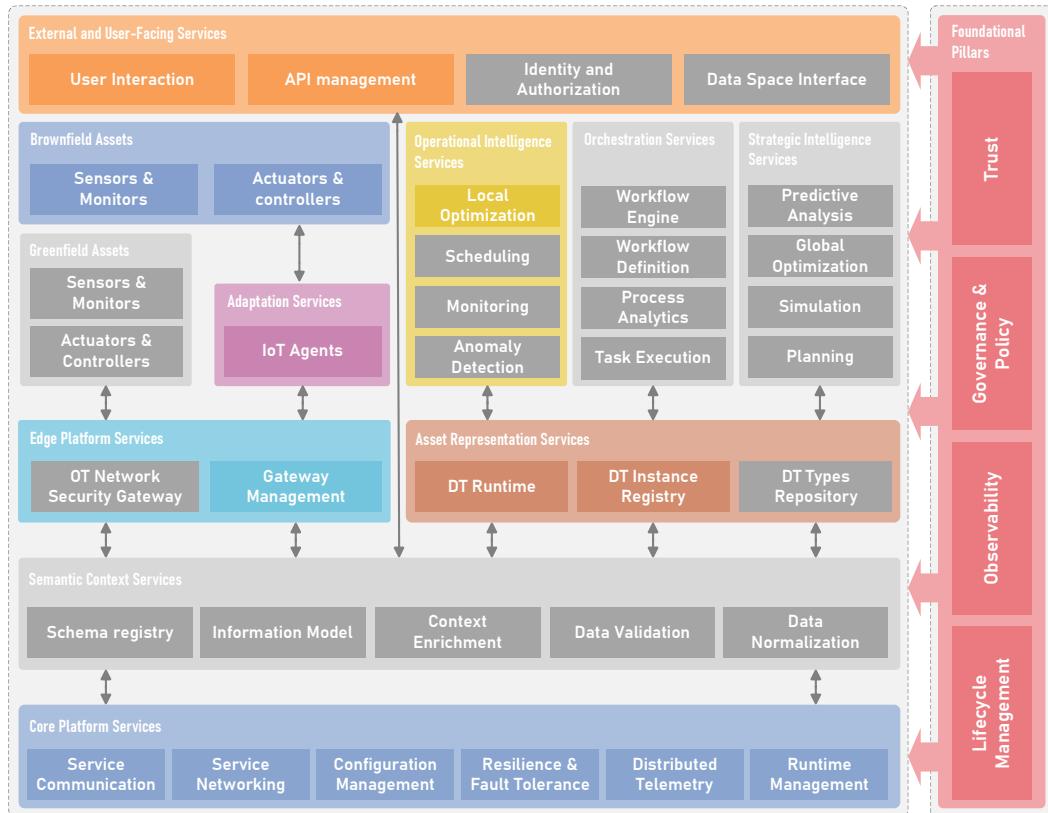


FIGURE 7.2: A Concrete Implementation of the Reference Architecture:  
The MAIA Architecture.

TABLE 7.1: Mapping of MAIA Components to the Reference Architecture

Reference Architecture Component	Corresponding MAIA Component(s)	Details
External and User-Facing Services	Web-based Interface	Provides a human–machine interface (HMI) for visualizing analytics results.
Asset Representation Services	Representation Service & Knowledge Base Service	A simplified digital twin implementation. Models each robot's state (ID, location), serving as the system's stateful memory.
Operational Intelligence Services	Path Prediction Service	A concrete example of an edge analytics service (robot path prediction).

*Continued on next page*

TABLE 7.1: – Continued from previous page

Reference Architecture	Corresponding MAIA Component	Details
Component	Component(s)	
Edge Platform Services	Gateway Service	Collects and prepares sensor data from AMRs before forwarding.
Adaptation Services	MQTT/HTTP Agent	Protocol translation: MQTT (from devices) to HTTP (for services).
Core Platform Services	Message Broker, Service Registration and Discovery, Configuration Management, Fault Tolerance, Service Monitor, Autoscaler	These services provide capabilities used by all microservices.
Foundational Pillars	(Implicitly addressed by implementation choices)	MAIA's microservices and automated scaling implement <i>Lifecycle Management</i> pillar. The use of the Circuit Breaker pattern and service isolation enhance Trust. Its integrated monitoring implements the Observability principle.

From this mapping, two key insights can be drawn. First, the MAIA implementation confirms the importance of the infrastructure-level components, which were defined in CRACI as *Core Platform Services*. The explicit implementation of discovery, monitoring, and lifecycle management services demonstrates that these capabilities are essential for microservices-based architecture. Second, it shows that the Foundational Pillars can be naturally included in a well-designed cloud-native architecture. Although not explicitly formalized, MAIA's use of patterns such as Circuit Breakers (supports resilience within the *Trust* pillar) and automated scaling (Lifecycle Management) illustrates how these capabilities can be integrated into the system's design using existing cloud-native tools.

## 7.3 Implementation

Before presenting the evaluation results, this section first discusses the implementation details of the MAIA architecture.

### 7.3.1 Implementation of the Core Platform Services

The MAIA prototype implements a platform through a set of microservices, which align closely with the defined *Core Platform Services*:

- *Service Communication:* MAIA adopts a message broker based on MQTT (Message Queuing Telemetry Transport) [458] using RabbitMQ<sup>2</sup> as the primary asynchronous communication mechanism. It manages multiple dedicated queues to decouple services and maintain non-blocking interactions.
- *Service Networking:* A dedicated *Service Registration and Discovery* component based on Netflix Eureka Service Registry<sup>3</sup> enables dynamic discovery and registration of microservices at runtime. This registry service is deployed redundantly to avoid single points of failure.
- *Configuration Management:* MAIA uses *Spring Cloud Config*<sup>4</sup> to centralize and manage configuration properties for all microservices. This maintains consistent deployment environments and allows runtime reconfiguration of service parameters. The use of a versioned configuration server also supports traceability and rollback.
- *Resilience & Fault Tolerance:* MAIA utilizes *Spring Cloud Circuit Breaker*<sup>5</sup> and *Spring Retry*<sup>6</sup> to handle service invocation failures. These mechanisms protect the system from cascading faults by isolating failures and enabling retry policies for transient errors.
- *Monitoring and Telemetry:* MAIA integrates two key services for observability:
  - *Service Logging:* Aggregates logs from all services into a centralized system for auditing and debugging.
  - *Service Monitoring and Management:* Exposes HTTP endpoints reporting service health and resource usage, which are used to generate real-time telemetry data. The platform uses *Spring Boot Actuator* to expose operational metrics and integrates with *Spring Boot Admin*<sup>7</sup> and *Spring Eureka* to provide a dashboard for monitoring the health and performance of all services in real-time.
- *Runtime Management:* *Service Autoscaler* is implemented to support automatic scaling. It monitors the message queue lengths and triggers horizontal scaling of subscribers based on demand thresholds.

---

<sup>2</sup><https://www.rabbitmq.com/>

<sup>3</sup><https://github.com/spring-cloud/spring-cloud-netflix>

<sup>4</sup><https://github.com/spring-cloud/spring-cloud-config>

<sup>5</sup><https://github.com/spring-cloud/spring-cloud-circuitbreaker>

<sup>6</sup><https://github.com/spring-projects/spring-retry>

<sup>7</sup><https://github.com/codecentric/spring-boot-admin>

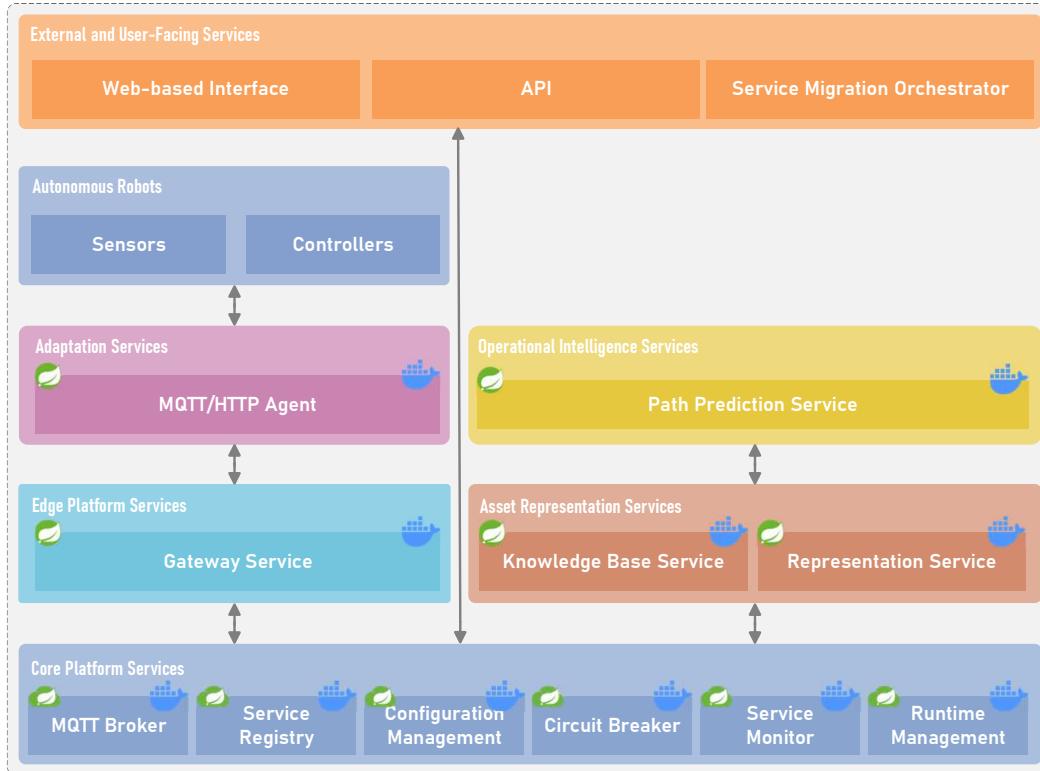


FIGURE 7.3: Implementation of MAIA Architecture.

### 7.3.2 Implementation of the Main Logic Services

The remaining services implement the application's main logic. All services in the system are implemented using Java and Spring Boot<sup>8</sup>, containerized using Docker<sup>9</sup> as illustrated in Figure 7.3.

- *Edge Platform & Adaptation Services:* The *Gateway Service* acts as the ingress for physical devices, exposing REST APIs, ingesting data, and forwarding updates to downstream services. This is an example of the synchronous-to-asynchronous transformation via an *Eventual Consistency* pattern [459].
- *Asset Representation Services:* The *Representation Service* acts as the runtime digital twin instance for each AMR. It tracks the robot's state, including its ID, real-time location, and currently associated Access Point (AP). Its primary function is proactive monitoring: when a robot's distance from its current AP exceeds a predefined threshold, the service anticipates a handover and sends an event to trigger the prediction process at the *Path Prediction Service*. This component analyzes the robot's recent trajectory to predict the most likely zone it will enter next. It then outputs a ranked list of up to three potential next APs with confidence scores, which are sent as a recommendation. The *Knowledge Base Service* acts as the system's long-term memory, storing the history of predictions from the *Path Prediction Service* and their outcomes. This data supports state management and enables continuous improvement of prediction models.

<sup>8</sup><https://github.com/spring-projects/spring-boot>

<sup>9</sup><https://github.com/docker>

- *External & User-Facing Services:* The *Web-based Interface* is the direct implementation of the *User Interaction* layer in the proposed architecture. It provides operators with a dashboard for visualizing network metrics, monitoring microservice health, and accessing real-time and historical recommendations from the analytics pipeline. This interface supports both observability and operational control.

All services were implemented following the microservice architectural style, as introduced earlier. Together, they form a data analytics pipeline that processes IoT data in an industrial context.

## 7.4 Evaluation and Results

After presenting the components of MAIA and mapping them to the previously proposed CRACI architecture, the following section describes the evaluation setup and presents the corresponding results.

### 7.4.1 Evaluation Setup

All evaluations were conducted on a dedicated machine with the following specifications:

- *Processor:* Intel i9-7900X, 10 cores / 20 threads, base frequency 3.30 GHz (boost up to 4.30 GHz).
- *Memory:* 64 GB DDR4 RAM at 3000 MHz.
- *Storage:* Intel 600p 1 TB NVMe SSD, 1800 MB/s sequential read, 560 MB/s sequential write.
- *Operating System:* CentOS 7, kernel 3.10.0-693.11.1.el7.x86\_64.
- *Software:* Apache Maven 3.5.2, Docker CE 17.09.1-ce, Docker Compose 1.17.0 (build ac53b73).

### 7.4.2 Experimental Validation and Analysis

To validate the CRACI design, a series of experiments were conducted using the MAIA prototype<sup>10</sup>. The goal were twofold: (1) to quantify the resource overhead inherent in a distributed cloud-native microservice architecture, and (2) to evaluate the system's end-to-end performance and scalability. Specifically, Docker was used to package and deploy the microservices, as Docker and its ecosystem have become the de facto standard for service deployment in cloud-native environments. Therefore, evaluating the overhead introduced by Docker can be regarded as a representative case study of the general overhead associated with adopting a cloud-native architecture. This analysis demonstrates that while microservices introduce a measurable overhead, the architectural benefits can justify this trade-off.

<sup>10</sup>The source code is publicly available at: <https://gitlab.com/haidinhtuan/microservices-backend>

## Analyzing Microservice Overhead and Mitigation

**Artifact Size** To evaluate the overhead introduced by the implementation stack, the deployable artifact sizes of both the Core Logic Services (e.g., Gateway Service, Representation Service) and the Core Platform Services (e.g., Service Discovery, Service Monitor, Autoscaler) were measured. Three packaging methods were compared: raw Java executables (.jar), Docker containers based on a standard base image (`openjdk:8`), and containers using a lightweight base image (`openjdk:8-jre-alpine`). The results are illustrated in Figure 7.4.

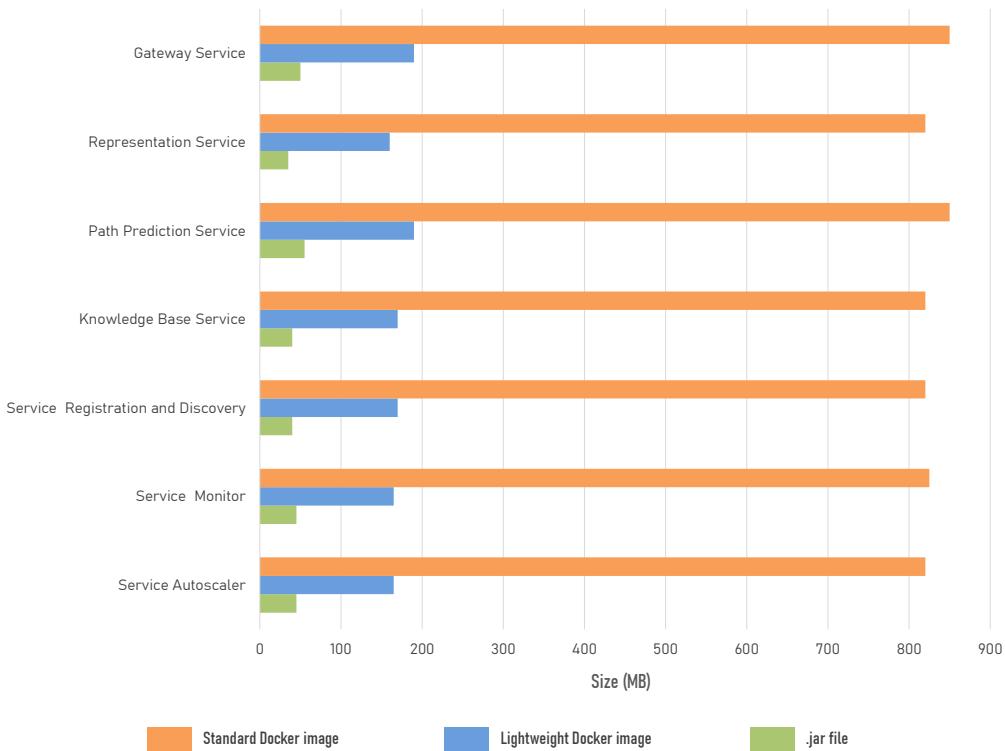


FIGURE 7.4: Comparison of Microservice Artifact Size: Raw Executable vs. Containerization.

The raw .jar files, which represent the uncontainerized application code and dependencies, are relatively small, ranging from 37.4 MB to 54.3 MB per service. However, when packaged using the standard `openjdk:8` Docker image (based on Debian Jessie), the artifact size increases significantly to over 800 MB, resulting in an increase of approximately 2000% or about 740 MB of additional overhead per service. In contrast, containers built with the minimal `openjdk:8-jre-alpine` base image (81.9 MB) demonstrate a significant reduction in size, producing final images in the 157–191 MB range. This corresponds to a reduction of up to 80.74% compared to default containers.

**Build Time** The build time of each service influences both developer productivity and the overall efficiency of CI/CD pipelines. To evaluate this, the build time for each microservice was measured under three different packaging methods. The resulting comparison is shown in Figure 7.5.

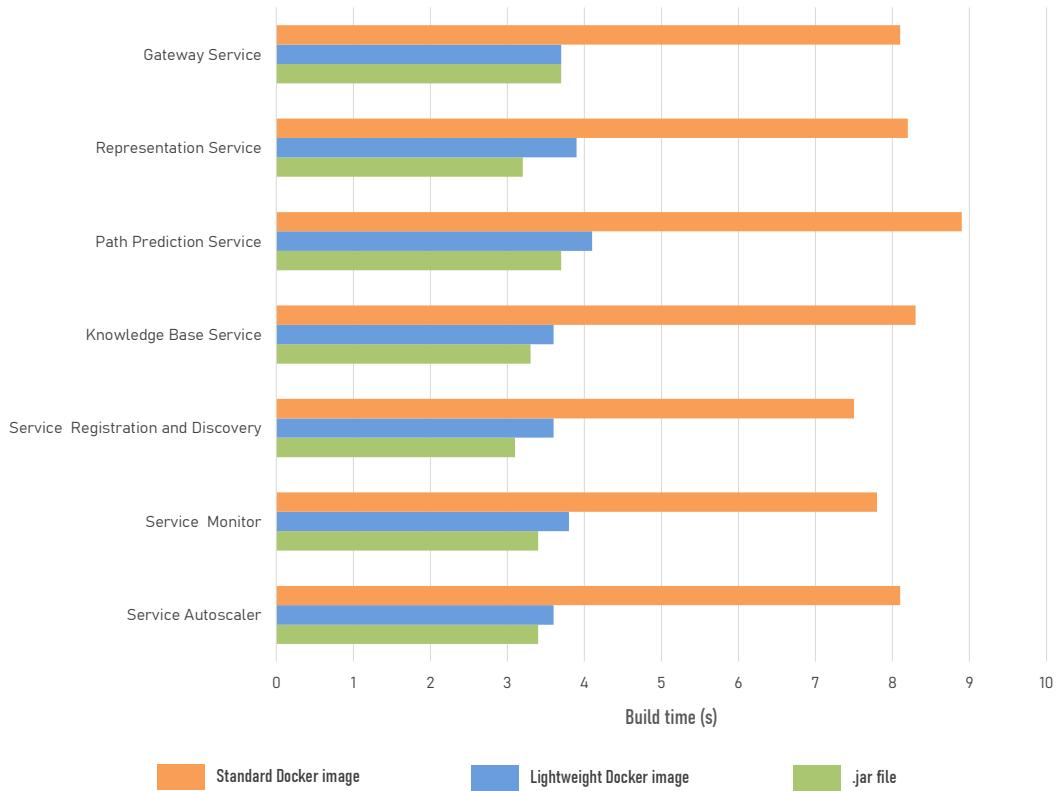


FIGURE 7.5: Comparison of Build Time for Raw Executable vs. Containerization.

The results confirm a clear correlation between artifact size and build time. Raw executables (.jar files) built using Maven consistently complete between 3.048 and 3.673 seconds, providing a reliable baseline. In contrast, building Docker containers with a standard base image (`openjdk:8`) significantly increases the build time, with durations ranging from 7.425 to 8.875 seconds, which is more than double the raw build time. This overhead primarily stems from downloading and assembling large base image layers.

Switching to a lightweight base image (`openjdk:8-jre-alpine`) yields substantial improvements. The corresponding build times drop to between 3.532 and 4.032 seconds, reducing the container build overhead by up to 59.76%. Importantly, the build time for lightweight containers closely approaches that of raw jar builds.

**Memory Consumption** Memory efficiency is a critical factor in microservice-based architectures, particularly when deploying to resource-constrained environments at the edge. This evaluation examines the memory footprint of containerized Java microservices within the MAIA prototype and explores tuning strategies to reduce excessive resource usage.

Initial measurements demonstrated unexpectedly high memory consumption. When executed with the default Java Virtual Machine (JVM) configuration on a host with 64 GB of RAM, each microservice consumed between 810.8 MB and 1,341 MB. Further investigation shows that this inefficiency stems from JVM ergonomics: JDK 8, by default, assigns a maximum heap size of 16 GB (one-fourth of total physical memory), regardless of the container’s memory limit. These results highlight a

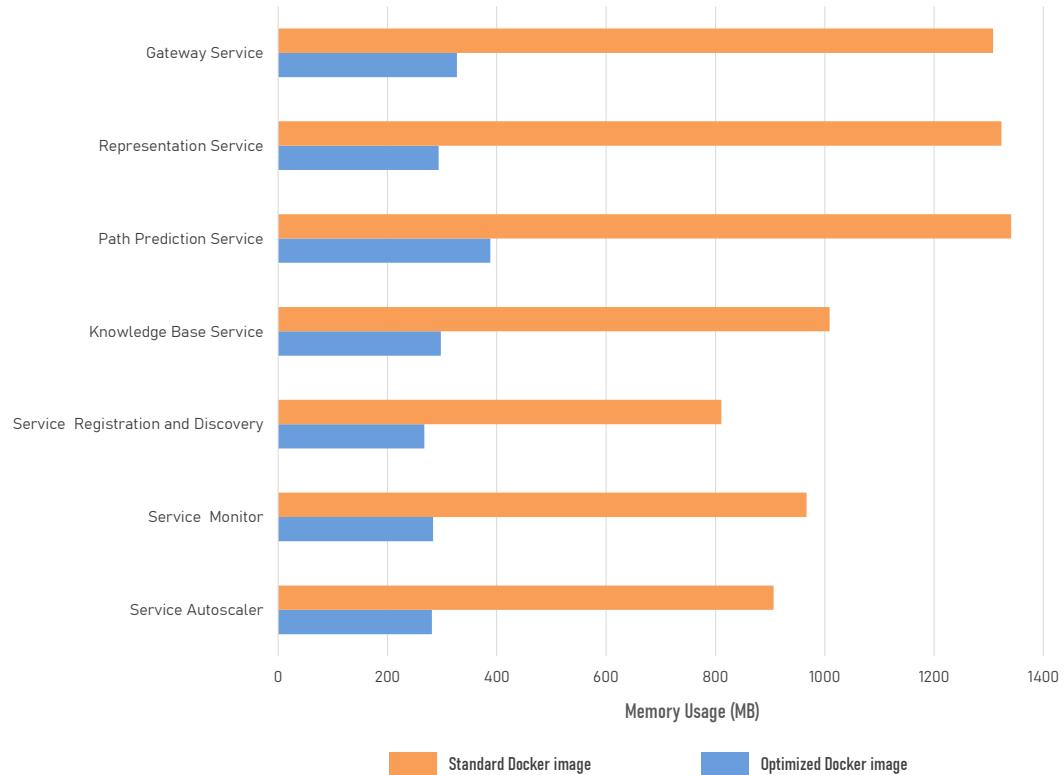


FIGURE 7.6: Impact of JVM Heap-Size Tuning on Container Memory Footprint.

well-documented issue, as older JVMs are not inherently *cgroup-aware*, resulting in substantial memory waste inside containers [460].

To address this issue, a systematic tuning process was carried out to determine the smallest viable heap size that maintains operational stability. The target metrics included: minor garbage collection (GC) durations under 50 ms, full GC intervals greater than 10 minutes, and no more than a 10% increase in service startup time. Through iterative testing with VisualVM and `jstat`, it was determined that setting `-Xmx64m` (a 64 MB maximum heap) was sufficient. Combined with container-aware JVM flags, this reduced memory usage dramatically.

For instance, the Representation Service's footprint dropped from 1,323 MB to 293.9 MB. Across all services, memory consumption was reduced by 67.01% to 78.79%, confirming the effectiveness of targeted heap-size tuning. The detailed results are presented in Figure 7.6.

**Analysis and Architectural Justification** Although containerization has become a widely adopted practice for deploying services in cloud-native environments, the evaluation demonstrated that a naïve *lift-and-shift* approach leads to significant inefficiencies. Default container images were excessively large (over 800 MB), build times were slow, and untuned JVMs consumed far more memory than necessary. However, the results also proved that these are not inherent flaws of the technology but consequences of poor practice. Through careful engineering processes, governed by the principles of the Lifecycle Management and Governance pillars, these trade-offs become manageable:

- *Lifecycle Management* (Controlling Artifact Size and Build Time): This pillar is realized through automated CI/CD pipelines. Instead of relying on individual developers to manually apply best practices, the pipeline enforces them automatically. To reduce artifact size, the pipeline is configured to use a standardized, lightweight base image (like Alpine) for all production builds, which was proven in the evaluation to achieve up to 80.74% size reduction. To reduce build time, the pipeline can enforce the use of lightweight images.
- *Governance & Policy* (Enforcing Optimization): To reduce memory consumption, a formal policy can be established: "All containerized Java services must use a *cgroup-aware* JVM and must have their heap size explicitly configured." This policy is then translated into code for automated deployment.

Ultimately, even with full optimization, a minimal microservice still incurs a non-trivial baseline overhead. According to the experiment results, a no-op minimal microservice still results in a 111 MB container image and 192 MB of runtime memory. This confirms that adopting a microservice architecture is a deliberate choice involving a clear trade-off. The system sacrifices resource efficiency for system-wide gains in modularity, scalability, and maintainability. For the complex, distributed system envisioned by CRACI, this is a necessary trade-off, and can be partly managed by the Foundational Pillars.

## Evaluating System Scalability and Performance

**System Scalability** To evaluate the scalability and responsiveness of the MAIA architecture, end-to-end latency was measured while processing 1 Hz location updates generated by a fleet of AMRs. Each request was assigned a unique identifier to enable *distributed tracing*, capturing the time from arrival to departure at each microservice.

A total of 16 test cases were conducted, varying the number of robots from 1 to 150, with each case repeated three times and mean values recorded. End-to-end latency was measured from the request's arrival at the *Gateway Service* to the delivery of the final recommendation to the *Web interface*, including all internal processing and inter-service delays. The results, depicted in Figure 7.7, showed three key findings:

1. *Latency scalability*: The total end-to-end latency increases from 10.872 ms (1 robot) to 45.293 ms (150 robots). Notably, the latency remains under 20 ms for up to 100 concurrently operating robots, which is well within acceptable bounds for near-real-time industrial applications.
2. *Communication bottleneck*: As the system scales, the relative proportion of time spent on inter-service communication increases substantially, from 67.62% with 1 robot to 86.97% with 150 robots. In contrast, the internal processing time within each microservice remains almost constant.
3. *Resource saturation*: Beyond approximately 120 robots, the latency curve exhibits exponential growth, indicating that system performance becomes increasingly constrained by resource saturation.

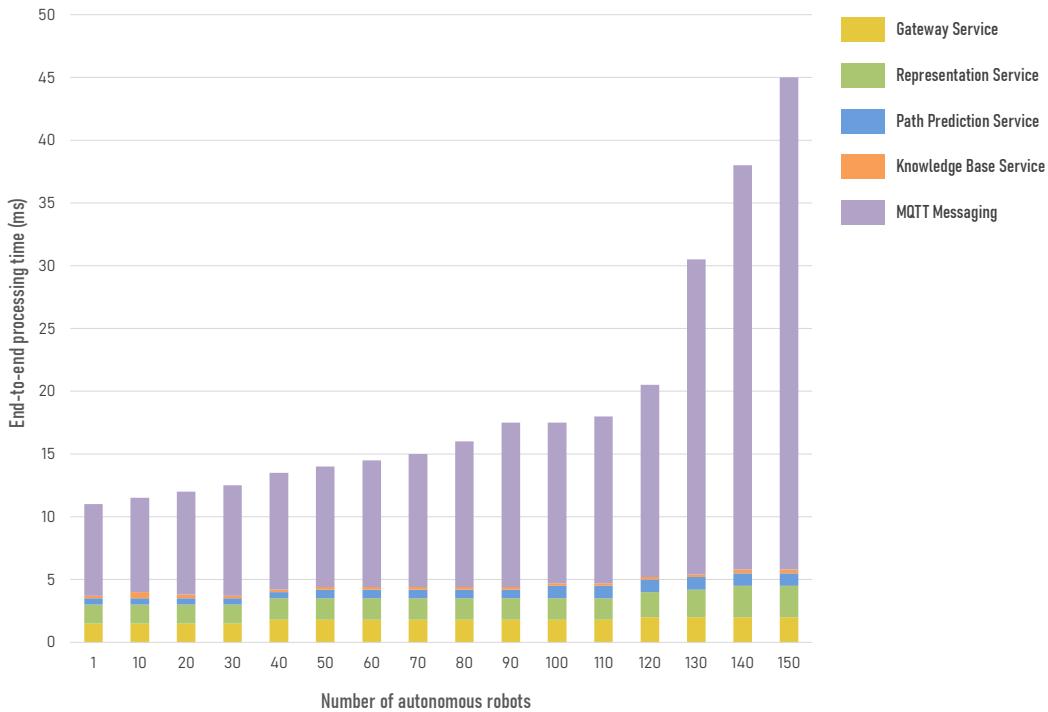


FIGURE 7.7: End-to-end processing time, adapted from a previously published work [457].

**Analysis and Architectural Justification** The latency evaluations demonstrate that as the system scales, the time spent in application logic becomes negligible compared to the overhead of the communication infrastructure. This shows that as parallel workload increases, the performance bottleneck shifts from *computation* to *coordination*. Several researchers also share the same conclusion in different contexts [461, 462]. This finding strongly validates two CRACI principles:

- *The Significant Role of the Observability Pillar:* The increasing dominance of message transport time, which is difficult to measure without proper instrumentation, emphasizes the necessity of the *Observability* pillar. For a distributed system such as MAIA, proper techniques need to be used to gain insights into systems' operations. For example, in MAIA case, the *distributed tracing* technique was used, which is a powerful tool to diagnose these network-level latencies.
- *The Value of Flexible Deployment:* This result also provides a clear justification for CRACI's strategy of deploying time-sensitive *Operational Intelligence Services* at the edge. Flexible deployment is a key advantage of the CC, enabling diverse services with varying latency and compute requirements to be efficiently deployed within industrial systems.

This insight confirms that service communication is a critical architectural element whose performance dictates overall system responsiveness, especially in a distributed system such as the continuum.

## 7.5 Evaluation Summary

The empirical evaluation of MAIA provides a quantitative validation of CRACI's core principles. The findings confirm the architecture's strengths while also highlighting the inherent trade-offs in its design.

### 7.5.1 Validated Strengths of the Reference Architecture

The MAIA implementation demonstrates the practical applicability of the CRACI design in several key aspects.

First, it demonstrates the feasibility of *achieving low-latency microservices at the edge*. The results show that a decentralized, microservices-based architecture can meet the strict sub-50-millisecond latency required for edge analytics. This finding validates CRACI's design choice to place real-time decision-making *Operational Intelligence Services* at the edge.

Second, the evaluation confirms the *effectiveness of the decoupled Hub-and-Spoke model*. By separating the Representation Service (the stateful hub) from the Path Prediction Service (the intelligence spoke), the architecture allows for independent scaling and development of components. This decoupling enhances modularity and enables independent evolution of services without compromising performance.

Finally, the results highlight the *significance of the Foundational Pillars*. The challenges observed with container overhead and messaging latency underscore that adopting microservices alone is insufficient. The need for optimized base images and efficient build times validates the importance of the Lifecycle Management pillar in governing CI/CD practices, while the identification of messaging overhead as the main bottleneck reinforces the role of the Observability pillar in providing detailed operational insight.

### 7.5.2 Identified Challenges and Areas for Improvement

The MAIA use case also identified the inherent trade-offs and complexities of this architectural style. These challenges reinforce the importance of CRACI's design principles and identify areas for future research.

First, *system complexity* remains a central concern in cloud-native architectures. As acknowledged by the authors [457], a distributed microservice-based architecture introduces its own operational complexity compared to a monolith. Managing dozens of services, their network interactions, and their data consistency requires a mature operational model, especially Lifecycle Management and Observability capabilities. They are essential prerequisites for successfully managing this complexity. This validates CRACI's approach of explicitly incorporating these as foundational design elements.

Second, the evaluation highlights the *resource consumption overhead* associated with containerized microservices, especially those running on the JVM. Compared to a monolithic implementation, microservices exhibit a noticeably larger memory footprint. This highlights a fundamental architectural trade-off: sacrificing resource efficiency for gains in flexibility and resilience. While CRACI's Core Platform Services (e.g., auto-scaling) help manage this, future work should explore more

resource-efficient runtimes (e.g., GraalVM native images) and languages (e.g., Go or Rust) for services where memory footprint is a primary concern.

Finally, *communication overhead emerged as the primary performance bottleneck*. The latency evaluation demonstrated that at scale, inter-service communication via the Message Broker became the dominant factor in end-to-end processing time, accounting for up to 86.97% of the total latency. Future research should focus on optimizing this inter-service overhead, potentially by developing specialized high-performance messaging fabrics or advanced data serialization formats.

## 7.6 Chapter Summary

This chapter's main goal was to validate the practical applicability of the CRACI reference architecture. By mapping its principles to a concrete implementation, MAIA, and empirically evaluating that implementation, this chapter has successfully validated the designs and also gathered important insights.

The evaluation provided three key points of validation. First, it demonstrated that while the resource trade-offs inherent to cloud-native technologies are significant, they can be systematically mitigated through the disciplined engineering practices embedded in CRACI's *Governance* and *Lifecycle Management* pillars. Second, the scalability analysis exposed the communication infrastructure as the primary performance bottleneck, emphasizing the importance of the *Observability* pillar and justifying CRACI's strategic emphasis on distributing intelligence toward the edge. Finally, the analysis of microservice overhead confirmed that it is a necessary and justified trade-off for achieving the agility required by complex industrial systems.

However, the MAIA implementation represents just one possible realization of the CRACI, built using a specific technology stack (Java/Spring Boot and Docker containers). This naturally leads to the next critical research question: *How do different implementation and deployment choices impact the performance, resource efficiency, and overall trade-offs when building services for the continuum?* Answering this requires moving beyond a single case study to a systematic comparison of the available options.

Therefore, having established CRACI as a feasible foundation, Part IV of this dissertation addresses this question directly. It will first present a comparative analysis of different microservice development frameworks (Chapter 8) and then provide a detailed analysis of the trade-offs between containers and unikernels as deployment paradigms (Chapter 9). These analyses will complement the results presented in this chapter, offering a more comprehensive foundation for the engineering of highly efficient industrial systems across the compute continuum.

## Part IV

# Implementing the Architecture: Comparative Analyses of Development and Deployment Paradigms



# 8 A Comparative Analysis of Implementation Paradigms

## 8.1 Introduction

Following the design and evaluation of the CRACI architecture in Part III, the next logical question concerns how the individual service components should be built? This chapter addresses this *how to build* question by providing data-driven data.

A central question for organizations developing new software is whether services should be built *from scratch* to maximize granular control and minimize external dependencies, or should a software framework be adopted to accelerate development and enforce best practices? This dissertation's main argument is that a framework-based approach is more appropriate for the demands of industrial applications, where operational consistency, scalability, and resilience are critical. This claim is supported through two main analyses, building upon findings from prior research by the author [463]:

- A conceptual analysis of the trade-offs between different development philosophies.
- An in-depth, empirical comparison of four representative microservice frameworks.

The focus of this chapter is on the construction of individual services. Topics related to runtime orchestration, placement, and migration are discussed in later chapters. This part of the dissertation, therefore, transitions from the conceptual design of the reference architecture to its practical implementation. It directly addresses **RQ2**: *What are the performance, resource efficiency, and scalability trade-offs of key implementation and deployment paradigms [...] when implementing services for this industrial architecture?* Through a comparative study of technological options for implementing the reference architecture, this chapter provides a structured assessment of implementation paradigms.

## 8.2 Background and Related work

A software framework can be defined as a *reusable, semi-complete* architecture that provides a standardized structure for a specific class of applications [464, 465]. From the literature, several key characteristics of software frameworks can be summarized as follows:

- *Inversion of Control (IoC)*: In contrast to a library, whose functions are invoked directly by the application, a framework dictates the overall program flow and makes callbacks to the developer's code [466].
- *Domain-specific*: Frameworks are typically designed for a particular problem domain, providing best practices, patterns, and abstractions for that domain [466, 464, 467].
- *Extensibility*: Frameworks are explicitly designed to be extended. They provide *hot spots* or *hooks*, such as abstract classes or interfaces, that allow developers to add custom functionality without modifying the framework's core, maintaining long-term adaptability [467, 468, 466].

Adopting a software development framework offers numerous advantages in productivity, interoperability, and quality. First and foremost, frameworks improve developer productivity by maximizing the reusability of both code and design patterns, allowing teams to focus on business logic [464, 466, 469]. At the same time, by reusing a consistent architectural pattern, developers can better manage complexities. By enabling a shared architectural mindset, frameworks encourage team collaboration and promote interoperability and scalability [469, 470, 471]. Consequently, this leads to better software quality and reliability, as applications are built on a tested core, reducing the likelihood of introducing new bugs [469, 472]. Finally, these frameworks benefit from active communities that provide resources, promote continuous improvement, and assist developers [473].

Despite their benefits, the adoption of software frameworks introduces significant challenges related to complexity, inflexibility, and operational transparency. The trade-off for a rich feature set is often a steep learning curve, a process that can significantly delay productivity [469, 474]. Furthermore, a framework often promotes certain solutions and technologies, leading to architectural rigidity, which can constrain developers and limit their ability to implement alternative solutions [469]. Finally, the same abstractions that simplify development can also lead to a complex software structure, introduce overhead, and cause difficulties in troubleshooting issues [474].

### 8.3 The Selected Microservice Frameworks

As established in Part III, CRACI's reliance on a microservice paradigm is a deliberate choice to ensure the developed system can meet critical industrial demands. Therefore, in this comparative analysis, four different microservice frameworks were selected, each using a different programming language: Go Micro (Go), Moleculer (JavaScript with Node.js), Lagom (Scala), and Spring Boot/Spring Cloud (Java). This set of frameworks reflects the variety of tools that can be used to develop microservices. In terms of language maturity, Java is considered an *old* language while Scala and Go both have been introduced in the last decade, and Node.js is one of the state-of-the-art run-time environment for JavaScript. In terms of framework applicability, although Spring Boot is widely used for microservices, it was not designed specifically for microservice architectures, unlike the other three

frameworks. A detailed high-level comparison is given in Table 8.1<sup>1</sup>. From this diversity, the analysis in the subsequent sections aims to highlight the fundamental differences in design philosophies, developer experiences (DevEX), maturity, and governance model.

## 8.4 Theoretical Comparative Analysis

The choice of a development framework is a critical architectural decision that establishes the foundational patterns and trade-offs for all services built within the CRACI architecture. The theoretical analysis of the four selected frameworks identifies a series of fundamental differences across four key aspects: design philosophy, developer experience, ecosystem maturity, and extensibility. The detailed analysis is summarized in Table 8.3 and Table 8.5<sup>2</sup>.

### 8.4.1 Design Philosophy

The main philosophical difference among the frameworks concerns their approach to state management and architectural opinionation level. At one end of the spectrum is Spring Boot/Cloud, which represents a pragmatic, flexible enterprise approach. It encourages stateless services and externalized persistence via familiar technologies like Object-Relational Mapping (ORMs) and ACID transactions (Atomicity, Consistency, Isolation, Durability) but does not strictly enforce it, allowing developers to integrate other patterns. In contrast, Lagom represents a highly prescriptive, reactive design philosophy. It is built from the ground up for stateful, resilient systems, strongly encourages advanced patterns such as Event Sourcing and Command Query Responsibility Segregation (CQRS) by default to prioritize responsiveness and elasticity, trading a steeper learning curve for long-term architectural integrity. In between these two extremes, Moleculer and Go Micro both adopt a lightweight, broker-centric, and stateless model aligned with Twelve-Factor App principles [449]. They prioritize developer productivity and runtime performance by treating state as a fully externalized concern, giving developers maximum control over the persistence layer's implementation.

### 8.4.2 Developer Experience and Testing Philosophy

The developer experience (DevEx) offered by each framework directly reflects its underlying philosophy. Spring Boot/Cloud provides a mature, feature-rich environment with extensive documentation and tooling like Spring Initializr, resulting in a moderate learning curve with high productivity for enterprise teams. Its testing philosophy is comprehensive, emphasizing integration and slice testing. Lagom, being highly opinionated, presents the steepest learning curve, as developers must follow its entire reactive stack, including Akka and CQRS. Its tooling, however, provides a powerful, integrated development environment for managing a complete system of services. Conversely, Moleculer and Go Micro offer the lowest barriers

<sup>1</sup>The table format was partially done with the help of an AI model (Google Gemini 2.5 Pro).

<sup>2</sup>The table format was partially done with the help of an AI model (Google Gemini 2.5 Pro).

TABLE 8.1: Comparative Analysis of Microservice Frameworks: Overview and Project Statistics (updated from [463]).

Aspect	Criteria	Spring Boot/Cloud	Lagom	Moleculer	Go Micro
<b>Overview</b>					
Supported Language	Java/Kotlin	Java/Scala	JavaScript (Nodejs)	Go	
Framework Type					
	General-purpose microservices, REST-centric with optional reactive support, full-stack enterprise features	Reactive, event-driven microservices, clear API boundaries, domain-driven design	Event-driven, broker-based microservices designed for low-latency and rapid development.	Lightweight RPC and Pub/Sub microservices with a pluggable, interface-based architecture	
Core Runtime Model	Servlet-based (Spring MVC) and reactive (WebFlux), managed via Spring Container	Akka actors and streams with Play Framework	Node.js event loop with Moleculer broker	Go runtime with plugin-based extensibility	
Initial release date	July 10th 2014	March 3rd 2016	February 16 2017	June 12th 2017	
Releases	314	49	85	334	
Commits	55461	2427	4282	3873	
Star	77603	2631	6288	22375	
Watch	3348	147	124	511	
Forks	41255	629	593	2373	
Contributors	1163	129	122	195	
Latest Release version	3.5.3	1.6.7	0.14.35	5.9.0	
Latest Release Date	June 20th 2025	December 14th 2021	November 6th 2024	June 24th 2025	
License	Apache 2.0	Apache 2.0	MIT	Apache 2.0	

TABLE 8.3: Comparative Analysis of Microservice Frameworks: Design and Development Experience (updated from [463]).

Aspect	Criteria	Spring Boot/Cloud	Lagom	Moleculer	Go Micro
<b>Design Philosophy</b>	Philosophy	Encourage stateless, REST-centric microservices, production readiness, modular, Convention over configuration.	Reactive, opinionated, event-driven, stateful microservices. Support Event Sourcing and CQRS by default with Akka's actor model.	Agile and event-driven, toolkit designed for rapid development and high performance, lightweight, broker-centric services.	Pragmatic minimalism; a lightweight, performant, and pluggable toolkit for RPC and Pub/Sub-based microservices.
	State Management	Unmanaged, externalized, Transactional management via Spring Data and JPA.	Better support for stateful services with ES/CQRS.	Unmanaged, externalized, aligning with Twelve-Factor App (stateless).	Unmanaged, externalize services.
<b>Developer Experience and Testing Philosophy</b>	Documentation	Very extensive	Extensive	Less Extensive	Limited
	Development Freedom	Moderate	Low	High	Very High
<b>Build tool &amp; Dependency Management</b>	Build tool & Dependency Management	Maven (default), Gradle, dependency management via Spring BOMs	sbt (Scala), Maven (Java). Lightbend's dependency management.	npm, yarn	Go modules
	Project Scaffolding	Spring Initializr (web-based)	Lagom Tech Hub Project starter (web-based)	moleculer-cli command	micro new command
<b>Tools</b>	IDEs	IntelliJ IDEA, VS Code, Eclipse with Spring Tools Suite (STS).	IntelliJ IDEA, sbt support, VS Code (via Scala plugins).	VS Code, WebStorm with Node.js tools.	VS Code, GoLand, LiteIDE, etc., with Go plugin support.
	Testing Philosophy	Optional hot-reloading (DevTools)	runAll command for local integrated testing.	Live Inspection & Debugging (REPL)	Runtime Interaction via CLI (micro)
<b>Developer Experience and Testing Philosophy</b>		Emphasis on unit, integration, and slice testing; Spring Boot Test, Testcontainers, and MockMvc/WebTestClient.	Unit and integration testing (Akka TestKit and Service TestKit); persistent entities testing (ScalaTest, JUnit).	unit and integration testing using Mocha, Jest, or other Node.js testing frameworks.	Go's native testing (testing package), table-driven tests, and integration testing with lightweight binaries.
<b>Learning Curve</b>		Moderate	High	Low	Low

TABLE 8.5: Comparative Analysis of Microservice Frameworks: Maturity and Extensibility (updated from [463]).

Maturity, Governance and Commercial Ecosystem					
Aspect	Criteria	Spring Boot/Cloud	Lagom	Moleculer	Go Micro
Maturity Level	Very mature	Moderately Mature	Moderately Mature	Moderately Mature	Moderately Mature
Release Cycle	Regular and predictable	EOL	Active maintenance	Active maintenance with minor updates frequently	
Governance model	Governed by VMware with open-source contributions	Governed by Lightbend with open-source contributions	Community-driven	Community-driven	
Community supports	Very Strong Community	EOL	Moderate active community	Moderate active community	
Commercial Support	Yes (VMWare Tanzu)	Yes (now EOL)	Yes (via Tidelift)	No	
Extensibility	Highly extensible. Integration-based: libraries can be integrated during build time.	Moderate extensibility. Stack-based: prescribed stack, limited infrastructure modularity.	Highly extensible. Support middleware, mixin, plugins.	Highly extensible. Interface-based: core functions are defined by Go interfaces.	
Modularity	High	Low	Very High	Very High	
Extensibility	Dependency Injection and Aspect-Oriented Programming (AOP).	standard Java/Scala libraries integrations	middleware system to hook into the request/event lifecycle and a mixin system for sharing code between services.	"Wrapper" concept, which is a clean middleware pattern for adding functionality.	
Scalability Orchestration	relies on external orchestrator (Kubernetes, Swarm, AWS ECS)	Relies on Akka Cluster's sharding and distribution mechanism	Nodes auto-discover to form a cluster.	relies on external orchestrator (Kubernetes, Nomad, Swarm)	
Deployment platform	Cloud-native (AWS, Azure, GCP), Kubernetes clusters, Docker, VMs; Primarily Cloud/Server. Edge-capable with optimization	Kubernetes, Docker, traditional VMs; often Server/Cluster environments; unsuitable for constrained edge.	Cloud or Edge	Cloud or Edge	

to entry. Moleculer provides a familiar workflow for Node.js developers, with powerful CLI tools and a built-in interactive REPL (Read-Eval-Print Loop) for rapid development. Go Micro offers a minimalist, performance-focused experience that utilizes Go's simple and efficient native tooling, giving developers maximum control at the cost of more manual setup.

### 8.4.3 Maturity, Governance, and Commercial Ecosystem

The long-term viability of an industrial system depends heavily on the maturity and governance model of its underlying framework. With a long development history and commercial backing from VMware, Spring Boot/Cloud is the most mature, offering a predictable release schedule and a clear Long-Term-Support (LTS) policy, crucial for enterprise stability. Lagom's End-of-Life status as of 2024, a result of a strategic pivot by its parent company, renders it unsuitable for new industrial projects despite its technical strengths, highlighting the risks of platform dependency. Moleculer and Go Micro represent mature, community-driven models. While technically robust and efficient, their informal governance and lack of formal LTS policies introduce some uncertainty for long-term, mission-critical industrial adoption.

### 8.4.4 Modularity and Extensibility

All frameworks provide modularity, but through different mechanisms. Spring enables powerful build-time modularity via Dependency Injection and its *starters* system, allowing features to be integrated without complex setup. Go Micro and Moleculer enable runtime extensibility through pluggable interfaces and middleware. Go Micro's *wrapper* pattern and Moleculer's plugin system for components like transporters and serializers provide a high level of flexibility, allowing developers to swap core functionalities without code changes. Lagom, in contrast, offered the least flexibility; its cohesive, opinionated stack emphasized stability over pluggability, tightly coupling the architecture to its core components like Akka and Kafka. This comparison shows that for a CC-based architecture like CRACI, the runtime extensibility of software frameworks provides a significant advantage over more rigid systems.

### 8.4.5 Alignment with the CRACI Architecture's Core Platform Services

One of the principal advantages of adopting a microservice architecture, and of using software frameworks in general, is the abstraction they provide. As discussed earlier, these frameworks primarily aim to reduce the inherent complexities of distributed systems, enabling developers to focus on building business logic [457]. Upon closer analysis of the functionalities these frameworks offer, it becomes evident that their functionalities align closely with the components of CRACI's Core Platform Services. Therefore, in this section, the analysis focuses on how each microservice framework

maps to the six core components of the Core Platform Services. Details are given in Table 8.7 and Table 8.9<sup>3</sup>.

**Service Communication** All frameworks support both synchronous and asynchronous communication patterns. Spring Boot/Cloud provides a comprehensive, flexible ecosystem, favoring synchronous RESTful APIs by default but extending the support for asynchronous messaging via Spring Cloud Stream<sup>4</sup>. By abstracting messaging protocols through *binders* and generic interfaces, it allows further extensions for other brokers. In contrast, Lagom is highly prescriptive, centering its entire architecture on a reactive, event-driven model using Akka Streams and Kafka as the default backbone. Moleculer and Go Micro provide the most flexible, pluggable solutions, treating the message broker (e.g., NATS, MQTT) as a swappable component, which grants architects greater control over the transport layer.

**Service Networking** The frameworks' service networking strategies range from external integration to fully built-in systems. Spring Boot/Cloud relies on integrating with external registries like Eureka<sup>5</sup> and Consul<sup>6</sup>, requiring separate infrastructure management efforts. In contrast, Lagom provides an implicit solution where networking is managed by the underlying Akka Cluster. Moleculer and Go Micro both adopt a *built-in* approach. Moleculer implements a decentralized, gossip-based registry for high availability, while Go Micro provides a pluggable registry that defaults to mDNS for development but can be backed by other systems like etcd<sup>7</sup>, simplifying the overall developer workflow.

**Configuration Management** In a distributed architecture, effective configuration management depends on centralized control and the ability to update services dynamically. Interestingly, these areas show the greatest differences among the frameworks. Spring Boot/Cloud offers the most comprehensive set of solutions. Its Spring Cloud Config Server<sup>8</sup>, typically backed by a Git repository, provides centralized, version-controlled configuration with support for dynamic refreshing and integration with secrets management tools like Vault<sup>9</sup>. Lagom and Moleculer adhere to standard externalization practices using HOCON files (Human-Optimized Config Object Notation) and environment variables respectively, but lack built-in support for centralized management. Go Micro offers a hybrid approach. It provides a robust, built-in Config interface designed for dynamic environments, supporting hot-reloading out-of-the-box. While it lacks a dedicated server component such as Spring, its pluggable architecture allows it to use backends like etcd or Consul to achieve a fully centralized and dynamic configuration system, offering a balance of built-in capability and operational flexibility.

---

<sup>3</sup>The table format was partially done with the help of an AI model (Google Gemini 2.5 Pro).

<sup>4</sup><https://spring.io/projects/spring-cloud-stream>

<sup>5</sup><https://github.com/spring-cloud/spring-cloud-netflix>

<sup>6</sup><https://github.com/hashicorp/consul>

<sup>7</sup><https://github.com/etcd-io/etcd>

<sup>8</sup><https://docs.spring.io/spring-cloud-config/docs/current/reference/html/>

<sup>9</sup><https://github.com/hashicorp/vault>

**Resilience and Fault Tolerance** Spring Boot/Cloud provides a powerful and flexible resilience strategy through Spring Cloud Circuit Breaker<sup>10</sup>, using libraries like Resilience4j<sup>11</sup> to offer several patterns such as *circuit breakers*, *retries*, *bulkheads*, *timeouts*, and *fallbacks*, all configurable within the Spring ecosystem. While requiring explicit integration, this design enables developers to tailor fault tolerance to system needs, thus maintaining high availability within distributed systems. Lagom, built on Akka, intrinsically supports resilience through *supervision hierarchies*, *automatic restarts*, and recovery using *event sourcing* and CQRS, with additional support for *circuit breakers* and *load control* via Akka Streams. Similarly, Moleculer offers a comprehensive built-in resilience toolkit, providing *circuit breakers*, *retries*, *timeouts*, *bulkheads*, and *fallbacks* natively with straightforward configurations, ensuring robust handling of failures without external libraries. Go Micro adopts a minimalist approach with built-in *client retries* and pluggable middleware for advanced patterns like *circuit breakers* and *timeouts*, allowing developers to extend resilience capabilities when required while maintaining a lightweight core.

**Distributed Telemetry** Spring Boot/Cloud offers a mature, integration-centric telemetry stack through Spring Boot Actuator and Micrometer<sup>12</sup> for metrics, supporting backends like Prometheus<sup>13</sup> and Datadog<sup>14</sup>, while tracing is handled via Micrometer Tracing with OpenTelemetry<sup>15</sup>, Zipkin<sup>16</sup>, or Jaeger<sup>17</sup>, providing distributed tracing and system-wide observability with minimal configuration. Logging is standardized through SLF4J<sup>18</sup>, ensuring consistent output across services. Lagom integrates with Lightbend Telemetry and Dropwizard<sup>19</sup> for metrics and supports tracing via Cinnamon and OpenTracing, providing instrumentation across Akka actors and streams, with basic health checks via Akka Cluster; however, fully utilizing this stack requires investment in the Lightbend ecosystem, with open-source users needing to configure third-party tools like Kamon<sup>20</sup> for observability. Moleculer provides built-in telemetry capabilities, including native metrics and tracing with Prometheus, Datadog, Jaeger, and Zipkin, along with structured logging through integrations with loggers like Pino<sup>21</sup> and Winston<sup>22</sup>, and heartbeat-based health checks, providing observability capabilities with relatively little setup. Go Micro follows a minimalist design, offering pluggable middleware for telemetry where metrics collection can be integrated with Prometheus and tracing with OpenTelemetry, while logging uses standard Go libraries, requiring explicit configuration to achieve full observability while maintaining the framework's lightweight footprint.

<sup>10</sup><https://spring.io/projects/spring-cloud-circuitbreaker>

<sup>11</sup><https://github.com/resilience4j/resilience4j>

<sup>12</sup><https://github.com/micrometer-metrics/tracing>

<sup>13</sup><https://prometheus.io/>

<sup>14</sup><https://www.datadoghq.com/>

<sup>15</sup><https://github.com/open-telemetry>

<sup>16</sup><https://github.com/openzipkin/zipkin>

<sup>17</sup><https://github.com/jaegertracing/jaeger>

<sup>18</sup><https://github.com/qos-ch/slf4j>

<sup>19</sup><https://github.com/dropwizard/dropwizard>

<sup>20</sup><https://github.com/kamon-io/Kamon>

<sup>21</sup><https://github.com/pinojs/pino>

<sup>22</sup><https://github.com/winstonjs/winston>

TABLE 8.7: Qualitative Comparison of Microservice Frameworks (Part 1: Communication, Networking, Configuration) (updated from [463]).

Pillar	Subcriterion	Spring Boot/Cloud	Lagom	Moleculer	Go Micro
Communication	Communication Paradigm	Sync (HTTP REST, gRPC), Async (Spring Cloud Stream with binders)	Sync (service descriptor), Async (Kafka via Alpakka), reactive streaming (Akka Streams)	Request-reply (over async bus), pub/sub (event bus)	Sync (gRPC), Async (broker)
	Protocols/Transport	HTTP/REST, gRPC, Kafka, RabbitMQ	HTTP/REST, gRPC, Kafka	TCP, NATS, MQTT, Redis, Kafka	gRPC, HTTP, NATS, MQTT
Networking	API Gateway/Routing	Spring Cloud Gateway	Play Framework Router	Built-in (moleculer-web)	Pluggable (Traefik, Kong)
	Load Balancing	Spring Cloud LoadBalancer	Provided by Akka	Built-in	Built-in
Service Discovery	Integration with Eureka, Consul, Zookeeper	Akka Discovery	Built-in (gossip-based), Redis, etcd	Built-in (mDNS), pluggable	
	Service Registry	Integration with Eureka, Consul, Zookeeper	Built-in (Akka Cluster)	Built-in registry	Pluggable (e.g., mDNS, etcd)
Configuration Management	Base Config Management	Properties/YAML files, profiles	HOCON via application.conf	JS/JSON/YAML config files	JSON/YAML files, env vars
	Centralized Config	Spring Cloud Config Server	N/A	N/A	Pluggable via etcd, Consul
	Secrets Management	Integration with Vault	N/A	N/A	N/A

TABLE 8.9: Qualitative Comparison of Microservice Frameworks (Part 2: Resilience, Telemetry, Runtime) (updated from [463]).

Pillar	Subcriterion	Spring Boot/Cloud	Lagom	Molecular	Go Micro
Fault Tolerance & Resilience	Circuit Breaker	Resilience4j, Hystrix (legacy)	Built-in	Built-in	Pluggable (via middleware)
	Retry	Resilience4j, Spring Retry	Provided by Akka	Built-in	Basic retries (built-in)
	Timeouts	Resilience4j, config	Provided by Akka	Built-in	Pluggable (via middleware)
	Bulkhead	Resilience4j	dispatcher via Akka	Built-in	N/A
Backpressure	Fallbacks	Resilience4j, Hystrix (legacy)	supported by Circuit Breaker	Built-in	N/A
	Relies on broker/binder (RabbitMQ, Kafka)	Load Control via Akka Streams	N/A	N/A	N/A
	Health Checks	Spring Boot Actuator	Akka Cluster Health Check	Heartbeat	Built-in (implicit via registry)
Telemetry	Metrics	Micrometer, Actuator	Dropwizard (via Play); Lightbend Telemetry	Built-in	Pluggable (via middleware)
	Logging	SLF4J, logs	SLF4J, logs	Built-in logger	Pluggable (standard Go libs)
	Tracing	Micrometer Tracing + OpenTelemetry/Zipkin/Jaeger	Cinnamon, OpenTracing	Built-in	Pluggable (via middleware)
Runtime	Service Lifecycle	Spring Boot Actuator	Akka Management	Dynamic service loading/unloading	Basic start/stop hooks
	Scaling	External (via orchestrator)	Built-in via Akka Cluster & Sharding	Built-in clustering	External (via orchestrator)
	Hot Reloading	DevTools (dev only)	Supported (in dev mode)	Supported	N/A
Management	Graceful shutdown	Supported	Supported (via Akka Coordinated Shutdown)	Supported (Context tracking)	Supported

**Runtime Management** Spring Boot/Cloud offers *fat JAR* packaging, Actuator endpoints for health checks, Kubernetes integration, and hot reloading during development via `DevTools`, providing production-grade reliability despite JVM's heavier footprint and slower startup. Lagom enables cohesive development with `sbt runAll`, hot reloading, and Akka Cluster for scaling and coordinated shutdown, though it shares JVM's resource overhead. Moleculer, with Node.js, delivers fast-starting services with clustering, a CLI and dashboard for scaffolding, testing, and dynamic service management, supporting hot reloads and graceful shutdown while balancing runtime efficiency with developer convenience. Go Micro emphasizes runtime efficiency with small, statically linked binaries, near-instant startup, and minimal memory usage, supporting graceful shutdown and scaling via external orchestrators, making it highly suitable for disposable, resource-efficient services while lacking heavier frameworks' integrated runtime management tools.

## 8.5 Empirical Comparative Analysis

Following the extensive theoretical analysis covering multiple aspects of the four selected microservice frameworks, this section presents an empirical evaluation of these frameworks. The evaluation begins with the description of the use case, followed by the experimental setup and the presentation of the results.

### 8.5.1 Use Case

This chapter focuses on an Industry 4.0 scenario used as the primary case study: a real-time logistics optimization system managing a fleet of Autonomous Mobile Robots (AMRs) on a factory floor. The use case captures key challenges in industrial data processing: continuously collecting location data from mobile assets, analyzing it through a processing pipeline, and deriving insights to improve routing efficiency, lower energy use, and minimize delivery delays.

The primary objective of this logistics system is to continuously monitor fleet movement, identify congestion hotspots, and estimate energy consumption. The microservice-based data processing pipeline (illustrated in Figure 8.1) logically follows these steps:

- *Data Ingestion*: A fleet of AMRs continuously streams real-time location data to a central message broker.
- *Location Matching (API-Bound)*: As the first microservice in the pipeline, the *Location Matcher* consumes raw coordinates and maps them to predefined indoor paths and zones using a local API.
- *Congestion Correlation (I/O-Bound)*: Upon receiving a matched route, the *Congestion Matcher Service* correlates it with a live congestion heatmap from an external geodatabase, retrieving context such as congestion severity across zones.
- *Energy Estimation (CPU-Bound)*: The final service, the *Energy Consumption Estimator*, takes the contextualized route data (including distance and

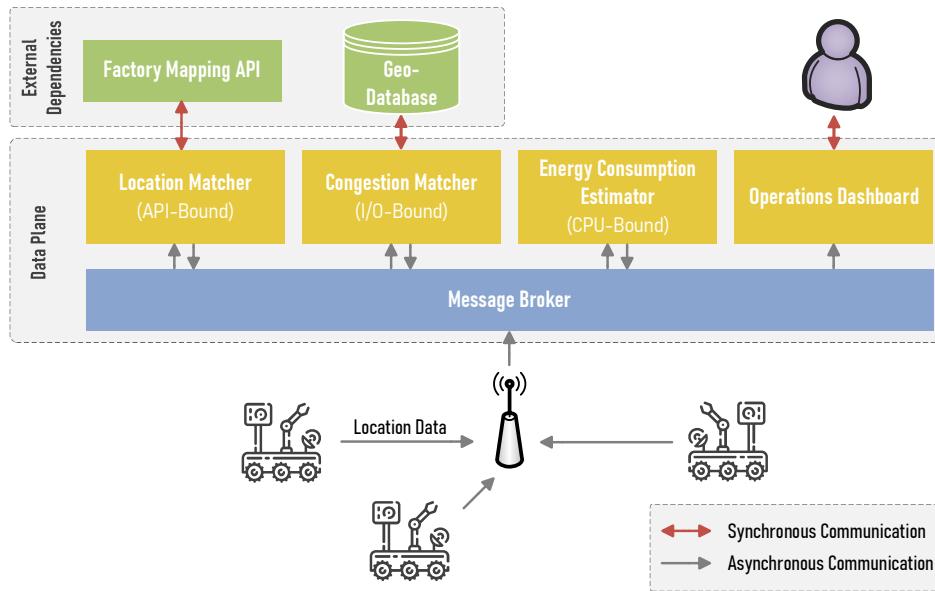


FIGURE 8.1: Overview of the AMR energy estimation pipeline.

congestion levels) and calculates the AMR's estimated energy usage. The final output is then pushed to an *Operations Dashboard* for action by factory floor personnel.

TABLE 8.11: Mapping of Original Toll System Components to the AMR Fleet Management Architecture.

Original Component (Toll System)	Abstract Architectural Pattern	New Component (AMR Management)	Industry Functionality	4.0
Vehicles	High-frequency data source	Autonomous Mobile Robots (AMRs)	The mobile assets in an automated logistics environment.	
Map Matcher Service	API-bound data enrichment	Location Matcher Service	Matches raw AMR coordinates to predefined factory floor paths and operational zones using a mapping API.	
Pollution Matcher Service	I/O-bound data correlation	Congestion Matcher Service	Correlates the AMR's route with a live congestion heatmap (stored in a database) to identify high-traffic segments.	
Toll Calculator Service	CPU-bound business logic	Energy Consumption Estimator	Applies an algorithm to calculate the AMR's estimated energy usage, factoring in distance and congestion.	
Dashboard	Data Sink / Visualization	Operations Dashboard	Provides live visualization of fleet routes, congestion, and energy costs to factory operators.	

### 8.5.2 Experimental Setup

The AMR logistics pipeline was implemented identically using the four frameworks: Spring Boot 2.2.4 (Java 8), Go Micro 1.18.0 (Go 1.13), Moleculer 0.13.0 (Node.js 13.1.4), and Lagom 1.6.0 (Scala 2.12.8). Services communicated via NATS (v2.1.2). An AMR Fleet Simulator replayed consistent test data by emitting periodic location updates from 10 robots. Tests ran on an Intel i5 machine with 16 GB RAM, with all microservices containerized and deployed via Docker and Docker Compose.

### 8.5.3 The Impact of JVM Warm-up on Service Startup

During initial testing, a significant latency spike was observed for the first messages processed by the Spring Boot and Lagom implementations, as illustrated for Lagom in Figure 8.2.

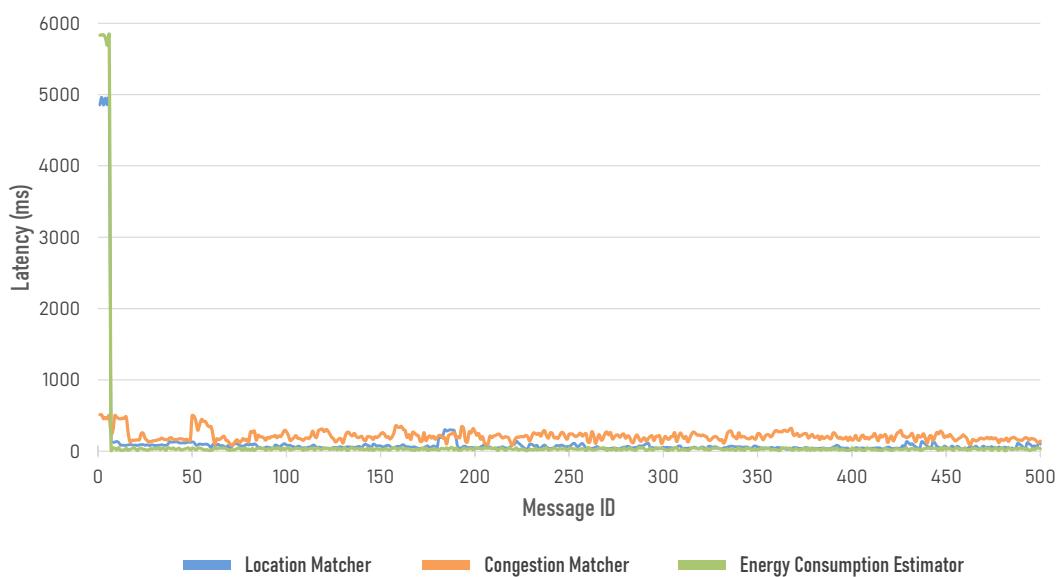


FIGURE 8.2: Example of long latency for the first messages processed by the Lagom-based implementation, demonstrating the JVM warm-up effect [463].

This behavior, which was not present in the Go Micro or Moleculer tests, is a well-documented characteristic of the Java Virtual Machine (JVM) known as the *warm-up* phase [475]. During this phase, the JVM performs just-in-time (JIT) compilation and other optimizations. To enable a fair comparison of steady-state performance and mitigate warm-up effects, all subsequent results presented in this chapter were gathered after an initial warm-up period of 50 message iterations (i.e., the first 50 messages were excluded from the analysis).

### 8.5.4 End-to-End Latency Performance

Figure 8.3 summarizes the end-to-end latency performance for the four pipeline implementations. While all frameworks exhibit broadly similar behavior, several key patterns can be identified and are discussed below. They will be discussed below.

The data show that for simple, synchronous API-bound workloads, Spring Boot can exhibit superior performance due to optimized HTTP client libraries and JVM

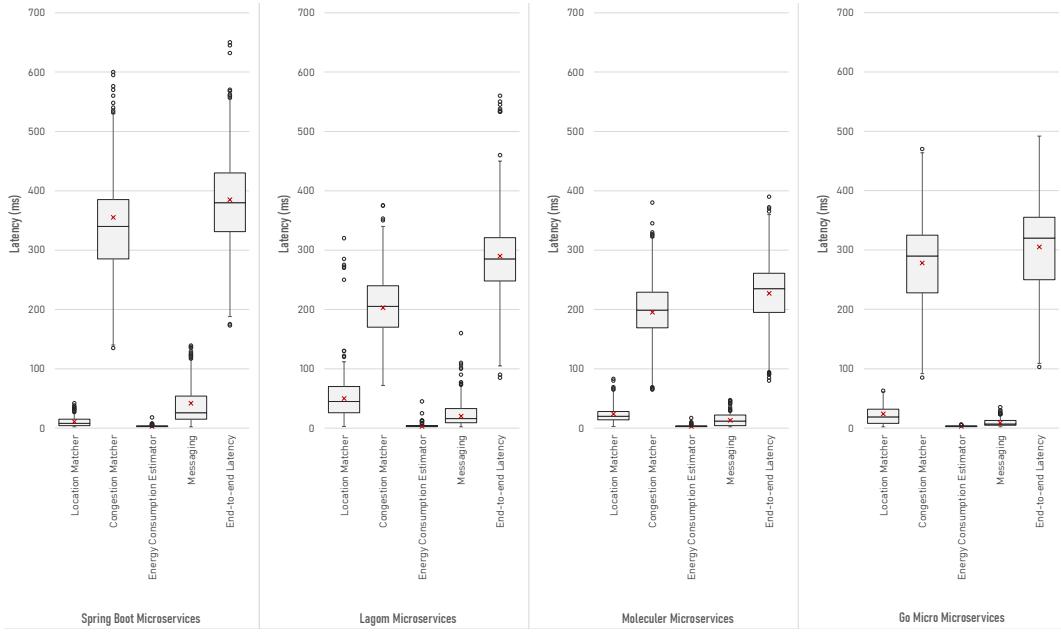


FIGURE 8.3: Comparison of latency performance for the microservices pipeline across frameworks [463].

runtime optimizations, even when using Apache HttpClient. This highlights that *lightweight* does not universally equate to *faster* for all workload types. Lagom's higher baseline latency can be traced to its Akka-based message routing and service locator layers, which add overhead for simple REST calls except under high concurrency. This latency difference demonstrates an engineering trade-off: Lagom's reactive architecture supports high-throughput, resilient systems but incurs measurable latency due to its abstraction layers.

The evaluation also shows that the primary system bottleneck is the *Congestion Matcher* service, whose database interactions dominate end-to-end latency across all frameworks. For example, in the Spring Boot implementation, the median latency of this service reaches 330.12 ms, while other services and messaging layers contribute under 50 ms each. Although this result cannot be generalized to all workloads, it highlights how external I/O can heavily influence latency in I/O-bound pipelines.

For I/O-heavy tasks, runtime choice significantly affects performance. Lightweight event-loop runtimes (Node.js) or efficiently compiled languages with modern concurrency (Go with goroutines) outperform traditional JVM-based frameworks. This suggests that the threading model, database drivers, or general overhead of the JVM environment introduce a significant performance penalty for this specific type of workload, particularly for Spring Boot. Interestingly, Lagom's I/O performance benefits from its underlying Play Framework and high-performance HikariCP connection pool, which reduces the overhead of managing database connections. This efficiency helps mitigate some of Lagom's JVM and reactive architecture overhead in I/O-bound scenarios.

For messaging, both Lagom and Spring Boot underperformed compared to Go Micro and Moleculer. This is mainly due to the lack of native support for the message broker used (NATS). For example, in the implementation using Spring, the raw Java NATS client (`io.nats.client.*`) was used indirectly, wrapped inside a Spring Boot

application context. Similarly, the Lagom-based implementation utilized a custom NATS client<sup>24</sup>, bypassing the framework’s native Kafka-based reactive streams API. In contrast, Go Micro and Moleculer natively utilize highly optimized NATS clients, benefiting from lightweight and minimal-abstraction integrations aligned with their concurrency models.

### 8.5.5 Resource Consumption

Besides latency, the resource efficiency of a service directly impacts its operational cost, especially in resource-constrained edge environments. The resource consumption was evaluated using two key metrics: deployment footprint and runtime CPU/memory usage.

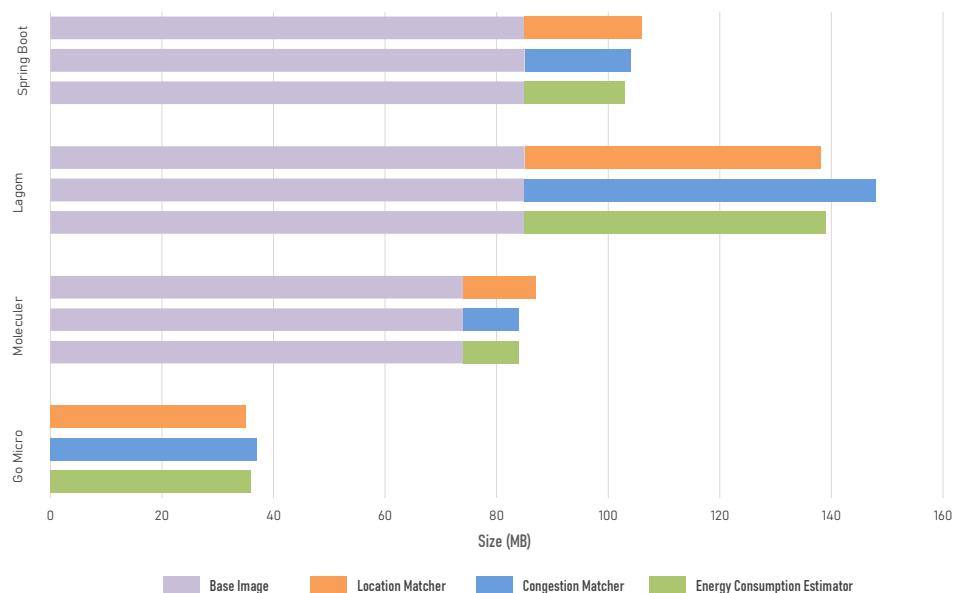


FIGURE 8.4: Docker image size comparison for the implemented services [463].

The size of the container images generated by each framework was evaluated, as smaller images enable faster deployment and scaling [476]. In this setup, informed by the previous chapter’s findings, Alpine-based images were used: Spring and Lagom used *openjdk:8-jre-alpine*, Moleculer used *mhart/alpine-node:10*, and Go Micro utilized the *golang* base image for building and the *scratch* image for deployment. The results are presented in Figure 8.4.

The most significant finding is the stark difference in image size between the natively compiled framework (Go Micro) and the frameworks that depend on a language runtime (Lagom, Moleculer, and Spring Boot). Go Micro produces exceptionally compact images because Go compiles to a native binary that can be deployed in a minimal *scratch* container, without the need for a large operating system and language runtime in the base image. For example, the Congestion Matcher service built with Go Micro (33.6 MB) is 78% smaller than its equivalent built with Lagom (156 MB). Moleculer also produces smaller images than the JVM-based frameworks, as it only needs to bundle the Node.js runtime with the source code. The

<sup>24</sup>The source code is publicly available at: <https://gitlab.com/haidinhtuan/scala-nats-client>

large size of the Lagom and Spring Boot images is a direct result of the requirement for a full JVM package.

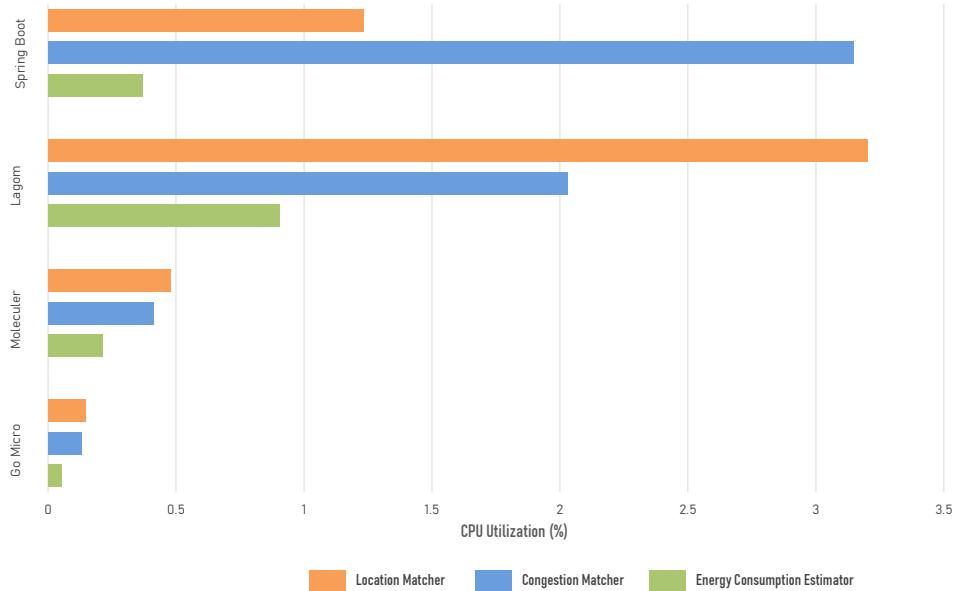


FIGURE 8.5: CPU consumption of the implemented services [463].

The analysis of runtime resource consumption, presented in Figure 8.5 and Figure 8.6, once again highlights the contrast between JVM-based and non-JVM frameworks. Both Spring Boot and Lagom exhibit substantially higher CPU and memory utilization compared to Go Micro and Moleculer. This significant overhead is a direct consequence of the resource requirements of the Java Virtual Machine itself. An interesting trade-off between the two JVM frameworks can be observed. For the API-bound *Location Matcher*, Spring Boot is more CPU-efficient, whereas Lagom consumes less CPU for the I/O-bound *Congestion Matcher*. This pattern is inverted for memory consumption, where Spring Boot's memory usage is higher for the *Congestion Matcher*.

Overall, Go Micro shows a high level of resource efficiency, consistently achieving the lowest CPU and memory footprint across all tested services. This validates its suitability for deployments in resource-constrained edge environments, which is an area of particular interest for the CRACI architecture.

When these findings are considered together with the latency analysis presented previously, a more complete view of overall performance can be seen. For example, Spring Boot delivers the lowest latency in the *Location Matcher* service, but this comes with considerably higher memory usage. Similarly, Lagom shows a slightly lower median end-to-end latency than Go Micro, yet requires significantly more CPU and memory resources. These results highlight the importance of evaluating frameworks from a holistic perspective, considering both latency and resource efficiency to support well-grounded architectural decisions.

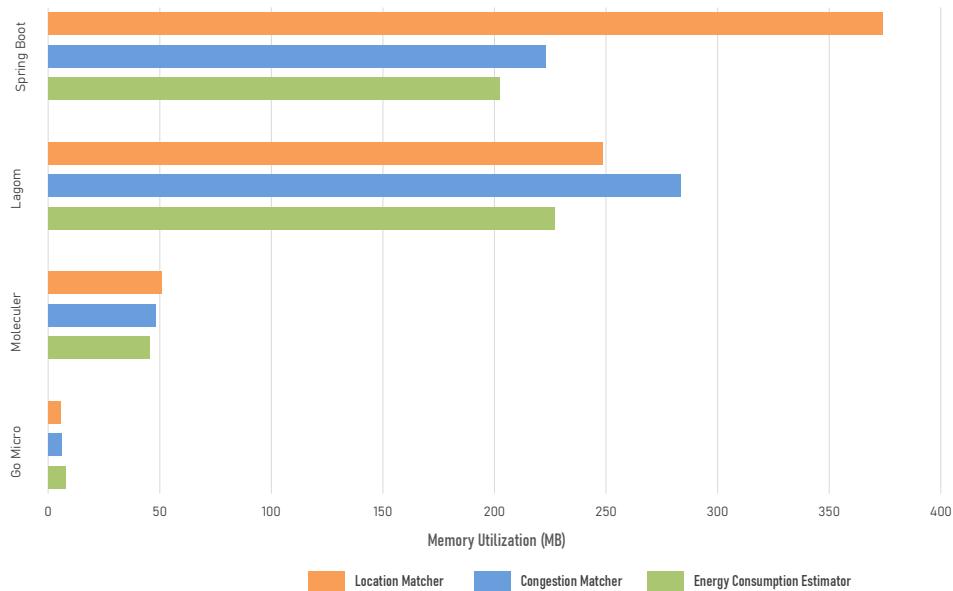


FIGURE 8.6: Memory consumption of the implemented services [463].

## 8.6 Chapter Summary

The preceding theoretical and empirical comparative analysis provides a detailed understanding of the implementation choices for the proposed CRACI architecture. The findings go beyond identifying a single “best” framework, instead suggesting a more nuanced, evidence-based approach to framework selection.

### 8.6.1 Framework Choice is Context-Dependent

The key takeaway from this analysis is that there is no universally optimal microservice framework. The selection process is multi-dimensional, balancing raw performance, resource efficiency, architectural philosophy, and the ecosystem richness. For instance, while Spring Boot excelled at simple API-bound tasks, it was the least performant for I/O-heavy workloads. This suggests that an architect implementing CRACI must adopt a *polyglot strategy*, selecting the right framework for the right job based on its specific location and role within the continuum. However, this raises an important point. In theory, microservices should be fully interchangeable. Yet, this also exposes a practical challenge: although microservices are theoretically interchangeable, the evaluation showed variations in service interface designs (e.g., NATS topic naming) that complicate cross-framework integration.

### 8.6.2 Validation of the CRACI Core Platform Services

Another key insight from this study is how the functionalities offered by modern microservice frameworks align with the CRACI architecture. The evaluated frameworks all provide built-in or pluggable solutions for nearly all functional blocks defined in the CRACI Core Platform Services, including service networking, communication, and resilience. This alignment indirectly validates the practical relevance of the CRACI reference architecture. It indicates that its defined features

capture the core capabilities needed to engineer state-of-the-art distributed industrial systems.

### 8.6.3 Performance Governed by Bottlenecks

The latency evaluation demonstrates that system performance is governed by its primary bottleneck. In the results, the I/O-bound *Congestion Matcher* service consistently dominated the end-to-end latency, regardless of the framework used. The inherent processing overhead of the frameworks themselves was minimal for CPU-bound and simple API-bound tasks. This observation leads to a crucial conclusion: optimization efforts must be focused on the slowest parts of a system—often external dependencies like databases or third-party APIs. Architectural choices that mitigate these bottlenecks (e.g., non-blocking I/O patterns, efficient connection pooling, caching) will have a far greater impact on overall performance than micro-optimizing a framework’s internal execution.

### 8.6.4 The Impact of Resource Efficiency

The evaluation results clearly show an order-of-magnitude penalty in resource consumption imposed by the JVM-based frameworks (Spring Boot, Lagom) compared to their lightweight counterparts. The analysis of deployment footprint (image size), memory utilization, and CPU overhead demonstrated a clear contrast. Go Micro, as a natively compiled framework, proved to be the most efficient, producing minimal container images and consuming the least CPU and memory. This result suggests important implications for the CRACI architecture, where the small operational footprint of compiled frameworks becomes a crucial advantage for deployment on resource-limited devices at the industrial edge.

### 8.6.5 Implementation Details Can Outweigh Theoretical Advantages

Perhaps the most subtle but important finding is that a developer’s specific implementation strategy can be more impactful than the theoretical advantages of a chosen framework. This was demonstrated in two key ways. First, the Lagom implementation’s performance, despite its blocking logic, was likely benefited by its use of the high-performance HikariCP connection pool. Conversely, the Go Micro implementation, despite its efficient runtime, likely suffered from an unconfigured default connection pool, leading to contention under load. This illustrates how strong performance in one part of the stack can compensate for inefficiencies elsewhere.

A similar pattern was seen with the NATS connector in Spring Boot and Lagom: while integration was possible, it was less optimized than in Moleculer or Go Micro, resulting in noticeably lower message-passing performance. Together, these findings emphasize that integration choices significantly influence overall performance. These insights require developers to apply domain expertise when designing and implementing such integrations.

### 8.6.6 The Trade-offs between Coupling and Interoperability

Finally, this analysis presented in this chapter reinforces the importance of CRACI's decoupled, event-driven design. Frameworks that encourage an API-first approach (Lagom, Moleculer, and Go Micro) help establish clear communication early in the development process. Furthermore, the choice of data interchange format presents a clear trade-off. Text-based, schema-less formats like JSON minimize coupling, while platform-specific protocols like Spring HTTP Invoker create tight platform coupling. Binary formats like Protocol Buffers offer a middle ground, enforcing a shared contract while maintaining language independence. This is a crucial feature for building scalable, polyglot industrial systems. For cloud-native architectures, selecting frameworks that rely on open standards generally provides greater long-term benefits than adopting tightly coupled technologies like Lagom's platform-specific stack.

This chapter has provided a clear answer to the first half of the implementation question: *how should services be developed?* The findings advocate for a polyglot strategy in which the choice of framework is decided by each service's workload characteristics, its position within the CRACI layers, and the desired balance between performance and developer productivity.

This conclusion, however, only addresses the *development* aspect. The second, equally important half of the question is: *how should these services be packaged and deployed?* The choice of deployment paradigm, specifically containers versus unikernels, has considerable impacts on resource efficiency, security, and startup latency, especially at the constrained industrial edge. Therefore, the following chapter will complete this investigation by providing a quantitative comparison of these two deployment models, offering a comprehensive understanding for both developing and deploying services within the CRACI architecture.

# 9 A Comparative Analysis of Deployment Paradigms

## 9.1 Introduction

The previous chapter demonstrated how the choice of programming language and framework can affect a microservice’s performance and resource usage. This chapter takes the analysis a step further by examining how the deployment paradigm, that is, how a service is packaged and deployed across the CC, shapes these same non-functional characteristics. Specifically, it compares two popular approaches in cloud-native environments: the widely adopted Docker container<sup>1</sup> and the emerging unikernels, represented by Nanos<sup>2</sup>.

The academic literature provides a strong theoretical basis for this comparison, highlighting a classic trade-off between specialization and generality. By compiling an application with only the necessary operating system libraries into a minimal, single-address-space machine image, they promise significant advantages. Research consistently demonstrates that unikernels offer a drastically reduced attack surface and stronger security isolation via the hypervisor, making them inherently more secure than containers, which share a host kernel [477, 478, 479]. This minimal design also leads to small image footprints and boot times measured in milliseconds, advantages that are critical for resource-constrained and latency-sensitive edge devices [480, 481].

On the other hand, Docker containers represent the pragmatic, general-purpose approach. Their primary strength lies not in their underlying technology alone, but in their vast and mature ecosystem [482, 483]. Tools like Kubernetes<sup>3</sup> and Docker Swarm<sup>4</sup> provide robust, industry-tested solutions for orchestration, scaling, and service discovery, a level of maturity the unikernel ecosystem has yet to achieve. This makes containerization the *de facto* choice for most applications, as it significantly improves software modularity, simplifies deployment, and aligns with established cloud-native practices [484].

While the literature clearly defines these theoretical trade-offs, a gap remains in the quantitative analysis of these paradigms under workloads specific to Industry 4.0. Several key questions therefore remain: How do the performance differences appear when using modern just-in-time (JIT) compiled runtimes such as Node.js, compared with ahead-of-time (AOT) compiled languages like Go? More importantly, how do these systems behave under the severe memory and CPU constraints typical of an

---

<sup>1</sup><https://github.com/docker>

<sup>2</sup><https://github.com/nanovms/nanos>

<sup>3</sup><https://kubernetes.io/>

<sup>4</sup><https://docs.docker.com/engine/swarm/>

industrial edge device? This chapter directly addresses these questions, providing the empirical data needed to answer the dissertation's second research question **RQ2**. To this end, a series of benchmarks was conducted to measure and quantify these trade-offs. Parts of the results and conclusions have already been published in a peer-reviewed publication [485]. The following hypotheses are proposed:

- **H1 (Static Properties):** Unikernels will demonstrate an order-of-magnitude advantage in startup time and a significantly smaller image footprint compared to containers, confirming the benefits of their minimal architecture.
- **H2 (Dynamic Performance):** The performance trade-off will be highly workload-dependent. While unikernels will excel in raw, single-core CPU-bound tasks due to minimal system overhead, Docker containers will show superior performance in I/O-bound, multi-core scenarios by using the host's highly optimized kernel. It is further hypothesized that a critical performance crossover will occur for JIT-based runtimes under memory pressure, where the stability of the mature Linux environment will outweigh the raw efficiency of the unikernel.

This chapter is organized to systematically evaluate the proposed hypotheses. Section 9.2 provides a background on the related deployment models. Section 9.3 details the experimental design and methodology. Section 9.4 presents the empirical results, and Section 9.5 discusses their implications. Finally, Section 9.6 concludes with a decision framework for selecting the appropriate deployment paradigm within the context of the CRACI architecture.

## 9.2 Background

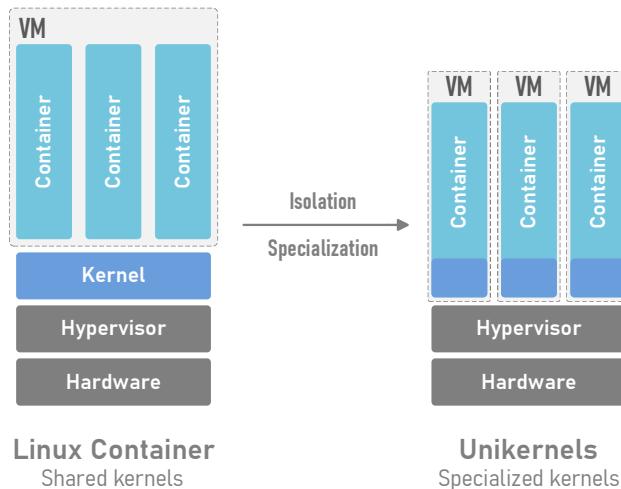


FIGURE 9.1: Architectural comparison showing a Linux container's reliance on a shared host kernel versus a unikernel's specialized, self-contained design running on a hypervisor (recreated from [486]).

Before comparing containers and unikernels, it is essential to identify their fundamental architectural differences. Although both represent forms of

virtualization, they are based on distinct design principles. Figure 9.1 illustrates the conceptual contrast between the two approaches.

Docker containers provide lightweight virtualization at the operating system level [487]. Instead of virtualizing the entire hardware stack, containers encapsulate an application and its dependencies into an isolated environment that runs directly on the host machine's OS. This is achieved by leveraging two core features of the Linux kernel:

- *Namespaces*: These provide isolation for system resources. Each container gets its own partitioned view of the process ID space, network interfaces, mount points, and user IDs, making it appear as if it is running on its own dedicated OS.
- *Control Groups (cgroups)*: These manage and limit the resources a container can consume, such as CPU time, memory, and I/O bandwidth.

Its defining characteristic is that *all containers share the same host OS kernel*. This eliminates the overhead of running multiple full operating systems, enabling rapid startup times and a higher density of applications per machine compared to traditional VMs [488].

Unikernels represent a fundamentally different, highly specialized approach to virtualization [489]. A unikernel is created using a library operating system, where the application code, its dependencies, and only the necessary OS functionalities (like a network stack or scheduler) are compiled together into a single bootable machine image. This creates a single-address-space artifact that runs directly on a hypervisor (like KVM or Xen) or on bare metal. Key architectural characteristics include:

- *No Shared Kernel*: Each unikernel is a unique, standalone bundle of kernel and application. It does not share a kernel with the host or other unikernels, providing strong, hardware-level isolation via the hypervisor.
- *Minimalist by Design*: All unnecessary components such as shells, user accounts, and unused drivers are completely excluded during the build process. This dramatically reduces the image size and attack surface.
- *No User/Kernel Separation*: By running as a single process in a single address space, unikernels eliminate the performance overhead associated with system calls and context switching between user mode and kernel mode.

In summary, while containers prioritize flexibility and ease of deployment through shared kernel isolation, unikernels take a minimal approach by compiling only what is necessary for a single-purpose system.

## 9.3 Evaluation Design

To test the hypotheses outlined in the introduction, a series of benchmarks was designed to measure the performance and resource trade-offs between Docker containers<sup>5</sup> and Nanos unikernels<sup>6</sup>.

---

<sup>5</sup><https://www.docker.com/>

<sup>6</sup><https://nanos.org/>

Two functionally identical applications were developed in Go (v1.22.2) and Node.js (v24.3.0) to compare an ahead-of-time (AOT) compiled language with a just-in-time (JIT) compiled language. This setup aims to reflect differences in execution models under each deployment paradigm<sup>7</sup>.

Each application exposes two REST API endpoints:

- *I/O-Bound Workload (/io)*: Returns a simple response to stress networking and scheduling with minimal application logic.
- *CPU-Bound Workload (/compute)*: Executes a 100-iteration SHA-256 hashing loop before responding.

Experiments were conducted on two dedicated servers located in the same data center. One hosted the application under evaluation, while the other generated test requests. This setup was designed to minimize interference between the test scripts and the application being tested. The configuration of the test server is as follows:

- CPU: Intel Xeon E3-1275 v6 (4 cores, 8 threads, 3.80 GHz) with hardware-assisted virtualization enabled to eliminate the overhead of software-based emulation.
- Memory: 64 GB DDR4 ECC RAM.
- Network: 1 Gbps full-duplex.
- OS: Ubuntu 24.04.2 LTS, Linux kernel 6.8.0-57-generic.
- Container Runtime: Docker v28.3.1.
- Unikernel Runtime: Nanos v0.1.54 with ops v0.1.43 via QEMU v8.2.2.

The evaluation focuses on two main aspects:

- *Static performance*: Image footprint (MB) and time to ready (cold-start latency).
- *Dynamic performance*: Throughput (req/s), latency (ms), and jitter (ms).

All test cases were conducted under two scenarios:

- *Resource-Rich Scenario*: 8 CPU cores, 2048 MB RAM, simulating a well-provisioned edge server.
- *Resource-Constrained Scenario*: Single CPU core, memory varied across 512 MB, 256 MB, 128 MB, and 64 MB, simulating industrial gateways.

## 9.4 Results and Analysis

The evaluation focuses on two key criteria: static properties and runtime performance. Both aspects were systematically evaluated under varying experimental settings, and the results are presented in the following sections.

---

<sup>7</sup>The source code is available at: <https://github.com/haidinhtuan/Unikernel-vs-Container>

### 9.4.1 Static Properties: Footprint and Startup Time

For the Go implementation, three artifacts were compared: the raw binary (4.6 MB), the Nanos image (7.6 MB), and a Docker image built with `golang:1.22-alpine` as the *builder* and `scratch` as the *base* (7.0 MB). The Docker image is slightly larger than the raw binary due to image metadata and layered filesystem overhead, while the Nanos image adds approximately 3 MB for the included Nanos kernel, as unikernels package their own kernel while containers use the host's kernel.

For the Node.js application, the raw binary produced with `pkg` measured 45.0 MB. The Docker image (`FROM node:alpine`) expanded to 127.0 MB because it includes the Alpine OS, the full Node.js runtime and V8 engine, and the application itself. In contrast, the Nanos image measured only 53.0 MB, containing the application, the Nanos kernel, and a minimal subset of Node.js libraries. As shown in Figure 9.2, for interpreted or JIT-compiled languages such as Node.js, Nanos achieves roughly a 2.4 times reduction in image size compared with Docker by removing the guest OS layer.

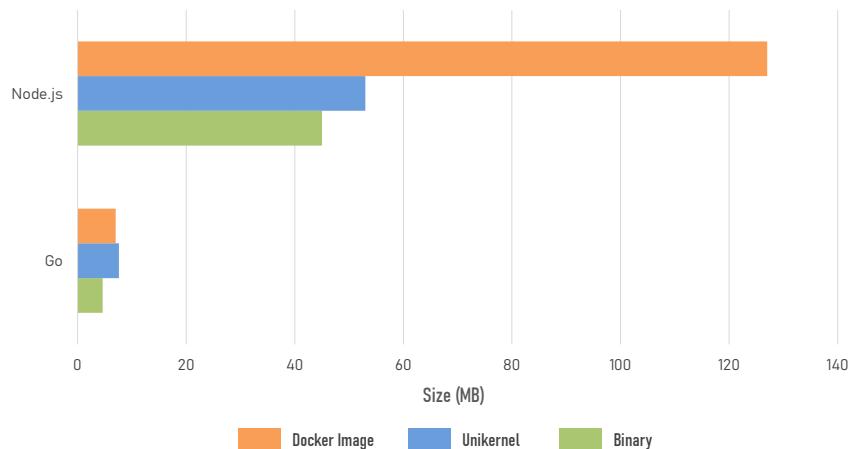


FIGURE 9.2: Comparison of image size between container and unikernel.

To measure boot time, a readiness-based approach was used, defining *Time to Ready* (TTR) as the interval between issuing the launch command and the application serving its first network request. For both Go and Node.js implementations, the unikernel consistently achieved a median TTR of 12 ms, whereas Docker containers required 158 ms (Go) and 219.5 ms (Node.js). These results indicate that the unikernel becomes operational approximately 13–18 times faster than the containerized equivalent, highlighting its highly deterministic startup behavior (Figure 9.3).

### 9.4.2 Performance Evaluation: I/O-Bound Workload

In addition to assessing the static properties of services packaged using containers and unikernels, this section further evaluates their runtime performance. The evaluation is structured around two main scenarios: resource-rich and resource-constrained, in order to systematically evaluate how each workload behaves under varying conditions.

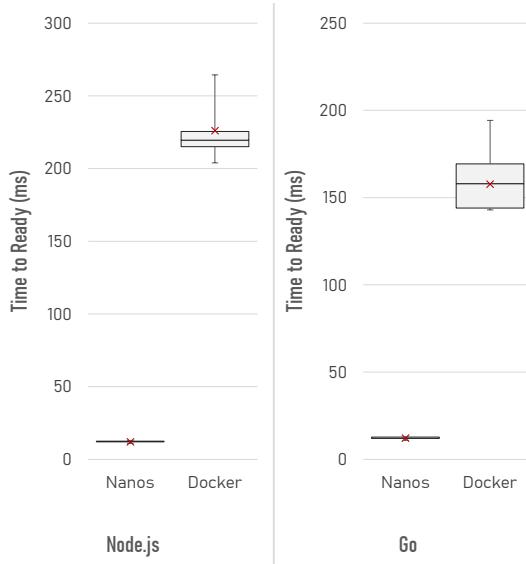


FIGURE 9.3: Comparison of Time to Ready between container and unikernel.

### Resource-rich Scenario

Both Docker and Nanos were allocated 8 CPU cores and 2048 MB of memory to eliminate hardware-related bottlenecks. For Go, Docker achieved 148,822.64 req/s, double that of Nanos (76,741.51 req/s) as shown in Figure 9.4. This gap is due to Docker's use of the host's optimized kernel-level networking stack, which efficiently manages highly parallel I/O workloads. In contrast, Nanos relies on user-mode networking through *ops*<sup>8</sup>, which introduces additional overhead and limits throughput. In contrast, for Node.js, Nanos outperformed Docker (47,368.19 req/s vs. 35,331.71 req/s). Because Node.js is single-threaded by default, it cannot fully utilize multiple CPU cores, shifting the bottleneck from network throughput to runtime efficiency. In this scenario, Nanos's minimal, OS-free environment reduces overhead, allowing the event loop to execute more efficiently and improving overall performance.

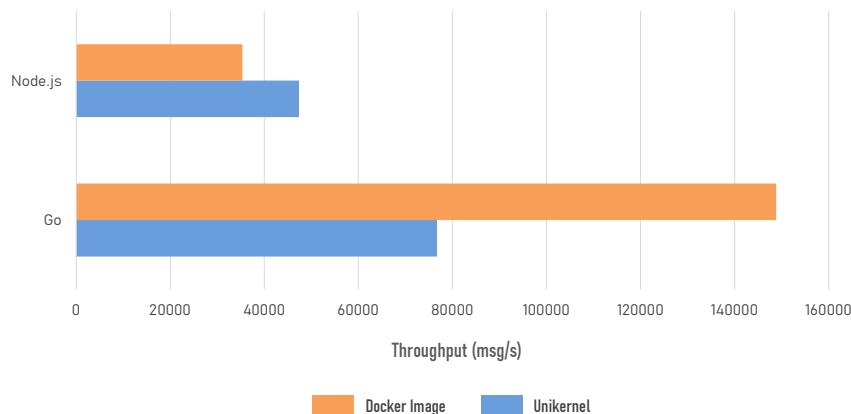


FIGURE 9.4: Comparison of Throughput between container and unikernel in a Resource-rich Scenario.

<sup>8</sup><https://docs.ops.city/ops/networking> (accessed on December 6, 2024).

The Go application running in Docker recorded an average latency of 2.67 ms with a very low standard deviation of 499.35  $\mu$ s, indicating very little jitter between requests. In comparison, the same application running as a Nanos unikernel had an average latency of 4.12 ms with a standard deviation of 47.58 ms. This is nearly 100 times higher than Docker’s, indicating a less predictable response time. Furthermore, the system dropped 37 requests due to timeouts, suggesting that while its average performance was good, it experienced intermittent saturation or scheduling contention that led to request failures.

### Resource-constrained Scenario

To emulate a typical resource-constrained industrial edge device, a second set of benchmarks was conducted in which both Docker and Nanos were limited to a single CPU core. The available memory was then varied across 512 MB, 256 MB, 128 MB, and 64 MB to analyze each platform’s behavior under increasing memory pressure. This setup was designed to identify the performance characteristics of each system in constrained environments.

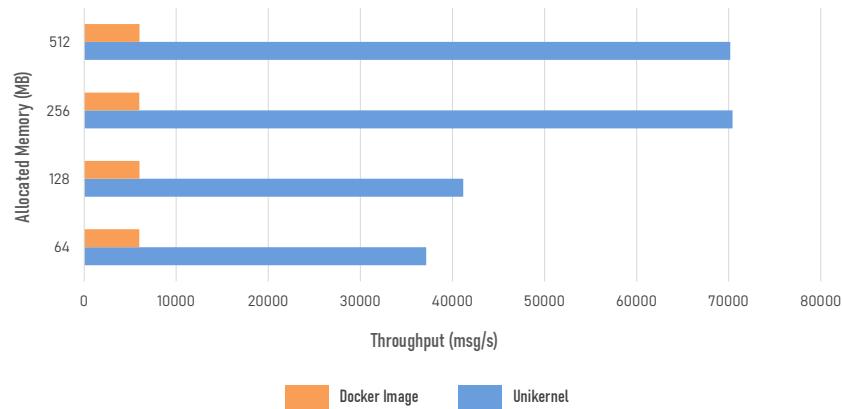


FIGURE 9.5: I/O-Bound Throughput for the Go Application under Varying Memory Constraints (1CPU). The Nanos unikernel consistently delivers an order-of-magnitude higher throughput.

Under these conditions, the Go application’s performance reversed the earlier results: Nanos consistently outperformed Docker, as illustrated in Figure 9.5 and Figure 9.6.

While Docker’s throughput remained stable across memory allocations, the Nanos unikernel consistently achieved higher overall performance. It delivered high and stable throughput at 512 MB and 256 MB (70,179.61 req/s and 70,410.22 req/s, respectively), with a correspondingly low average latency below 4 ms. At these memory levels, performance was not constrained by memory availability. However, when memory was reduced to 128 MB, the throughput dropped to 41,167.76 req/s, while the average latency increased to 4.31 ms. This result indicates that at 128 MB, memory availability has become the new primary bottleneck due to increased garbage collection overhead.

In contrast, the Docker container’s performance remained stable around 6,000 req/s across all tested memory levels. The average latency was also consistently high at around 72 ms. This stability, despite changing memory allocations, suggests

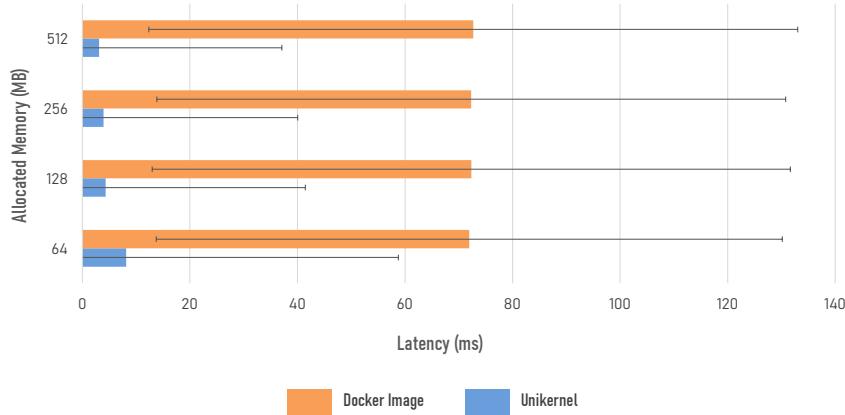


FIGURE 9.6: Average Latency and Standard Deviation for the Go Application under Varying Memory Constraints (1 CPU). Error bars represent the standard deviation. Nanos demonstrates both lower average latency and lower jitter.

that CPU availability is the primary limiting factor in these performance results. This observation aligns with the deployment configuration of the benchmark application, which relied on default runtime configurations. By default, Node.js operates on a single-threaded *event loop*, limiting its use of multiple CPU cores for concurrent requests. Go, on the other hand, uses *goroutines*, allowing it to utilize available CPU cores more effectively under concurrent workloads, resulting in improved scalability with increased CPU resources.

Unlike the straightforward results of the Go application, the Node.js benchmarks demonstrated a different pattern in which the containerized version outperformed the unikernel as memory became more constrained (Figure 9.7 and Figure 9.8).

At higher memory allocations, the Nanos unikernel performed slightly better than Docker (43,425.20 req/s vs 34,411.89 req/s at 512 MB). The performance advantage for Nanos can be attributed to the absence of a full guest OS, which eliminated background processes and reduced scheduling overhead, allowing the Node.js event loop to operate with greater efficiency. At 256 MB, the two platforms performed almost identically, suggesting this is the point where the memory advantage of Nanos is balanced by Docker.

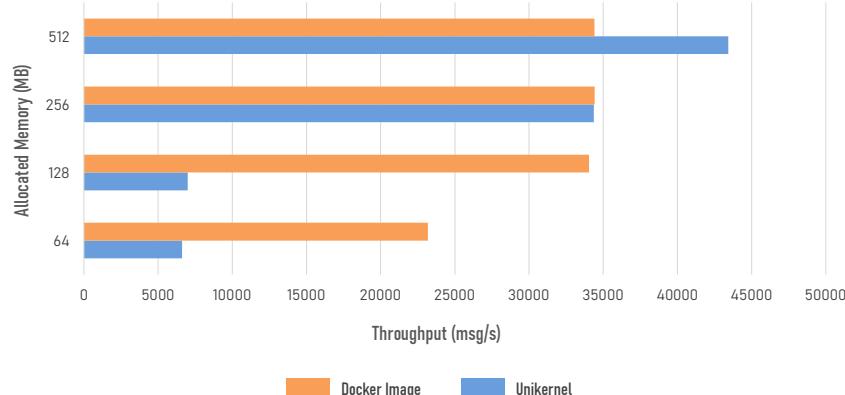


FIGURE 9.7: I/O-Bound Throughput for the Node.js Application under Varying Memory Constraints (1 CPU). A performance crossover occurs at 128 MB, where Docker performs better.

A clear performance reversal occurs at the 128 MB mark. The Nanos unikernel's throughput dropped sharply from 34,358.57 req/s (at 256 MB) to 6,992.61 req/s (at 128 MB), while the average latency increased significantly (from 9.17 ms to 25.00 ms). In contrast, Docker maintained stable and robust performance, maintaining 34,052.28 req/s at 128 MB and only degrading moderately to 23,175.36 req/s at 64 MB.

This performance degradation is also observed with the Go application, although to a lesser extent (41.53% drop vs. 79.65% for Node.js). This indicates that the runtime environment itself is a contributing factor to the observed performance decline. The Nanos unikernel, while lightweight and efficient under ample resources, used custom memory allocators [490] and lacks the advanced memory management mechanisms of the Linux kernel. As a result, under severe memory constraints, workloads that rely on JIT compilation and dynamic heap management, such as Node.js, are particularly vulnerable, revealing the limitations of unikernels in sustaining performance for such workloads in highly constrained environments.

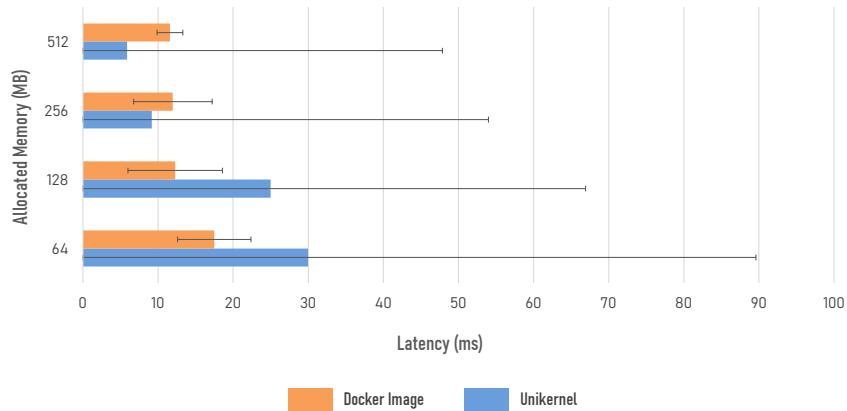


FIGURE 9.8: Average Latency and Standard Deviation for the Node.js Application (1 CPU). Note the increase in latency and jitter for the unikernel at 128 MB and below.

In summary, while Nanos unikernels offered a performance advantage with greater allocated memory, Docker containers provided a more stable platform for Node.js under highly constrained memory conditions. This finding was particularly significant for system architects, as it demonstrated that, for certain application stacks, the memory overhead of a container was a worthwhile trade-off for the performance stability gained from a mature and well-understood operating system environment.

#### 9.4.3 Performance Evaluation: CPU-Bound Workload

While many edge applications are I/O-bound, a significant and growing number of Industry 4.0 use cases involve intensive local data processing. These CPU-bound workloads include tasks such as data filtering, aggregation, real-time analytics, and executing machine learning inference models. These tasks have been conceptualized in CRACI's *Adaptation Services* and *Edge Platform Services*. The following test was therefore designed to minimize the impact of the networking stack and isolate the raw

computational efficiency of the application runtimes (Go vs. Node.js) on each platform (Nanos vs. Docker).

### Resource-rich Scenario

A particularly notable finding was the performance difference between the Go and Node.js applications. As shown in Figure 9.9, the Go application achieved a throughput of 91,655 req/s on Docker and 56,984 req/s on Nanos, up to 30 times higher than the Node.js application's (2,988.56 req/s and 3,027.46 req/s). This is a direct consequence of their different compilation models. The Go application, ahead-of-time (AOT) compiled into optimized native machine code, executed the computational loop with maximum efficiency. In contrast, the Node.js application, relying on the V8 engine's just-in-time (JIT) compilation, incurred significant runtime overhead in repetitive, CPU-intensive workloads. In addition, as also observed in the resource-rich I/O-bound scenario, Go demonstrated highly efficient multi-core utilization, reinforcing its suitability for compute-intensive industrial workloads.

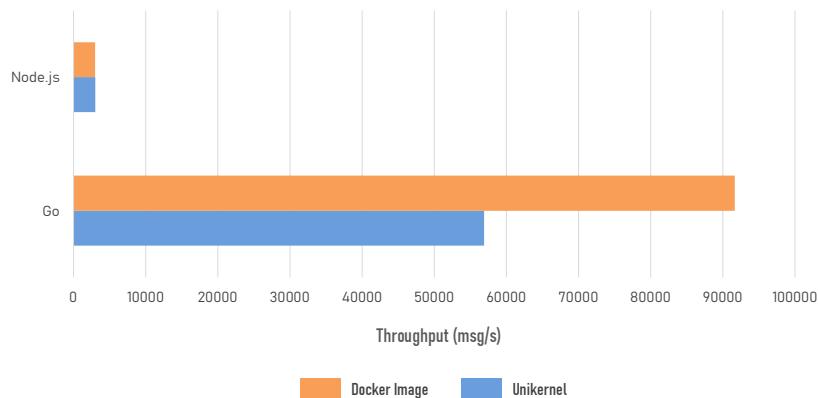


FIGURE 9.9: CPU-Bound Throughput in a Resource-rich Scenario (8 CPU, 2048 MB).

### Resource-constrained Scenario

This scenario evaluated the single-core computational efficiency of each platform under the same memory constraints used in the I/O-bound tests (512 MB, 256 MB, 128 MB, and 64 MB). The results are shown in Figure 9.10 and Figure 9.11.

Results from the throughput test showed a similar pattern as in the I/O-bound scenario, where the Go application consistently performed better on Nanos than on Docker across all memory levels. The unikernel, by design, has minimal system overhead as it does not rely on a separate guest operating system. When the Go application executed its hashing loop, the single available CPU core was almost entirely dedicated to that task. There is minimal *system noise* or background processes competing for CPU cycles. This high level of efficiency allows the application to achieve its maximum possible computational output on the given hardware. The performance degradation observed when memory was reduced from 256 MB to 128

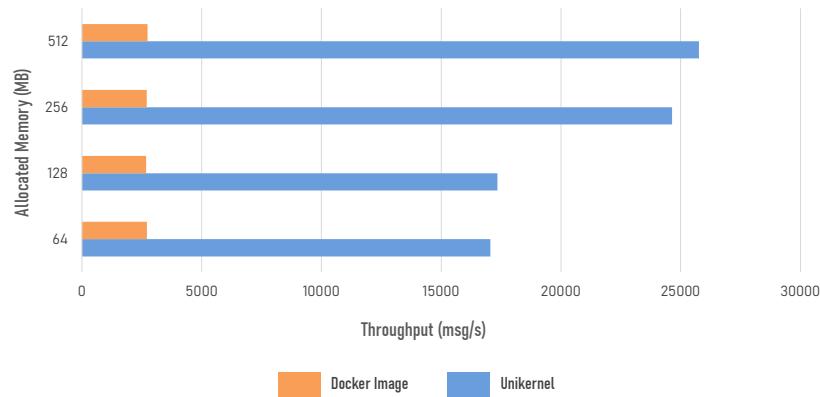


FIGURE 9.10: CPU-Bound Throughput for the Go Application under Varying Memory Constraints (1 CPU). The Nanos unikernel's throughput was an order of magnitude higher than Docker's across all memory levels.

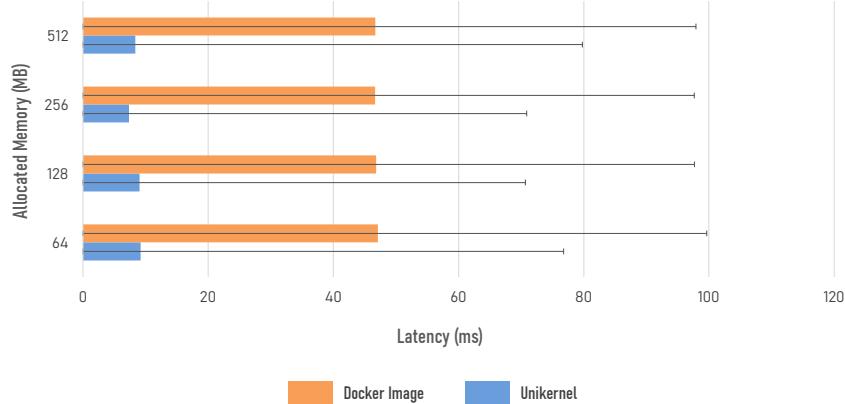


FIGURE 9.11: Average Latency and Standard Deviation for the Go Application (1 CPU). The unikernel demonstrated significantly lower average latency and less performance jitter compared to the container.

MB was a predictable consequence of increased memory pressure, which caused the garbage collector to consume a larger portion of the limited CPU time.

The Docker container's performance remained consistently low and flat at around 2,700 req/s. In this single-core configuration, the overhead of running the containerized environment along with associated context switching and system management tasks, consumed a significant portion of the available CPU cycles. This left less processing time available for the actual Go application workload. As a result, the system's bottleneck stemmed not from the application itself but from the architectural overhead inherent to the containerized environment.

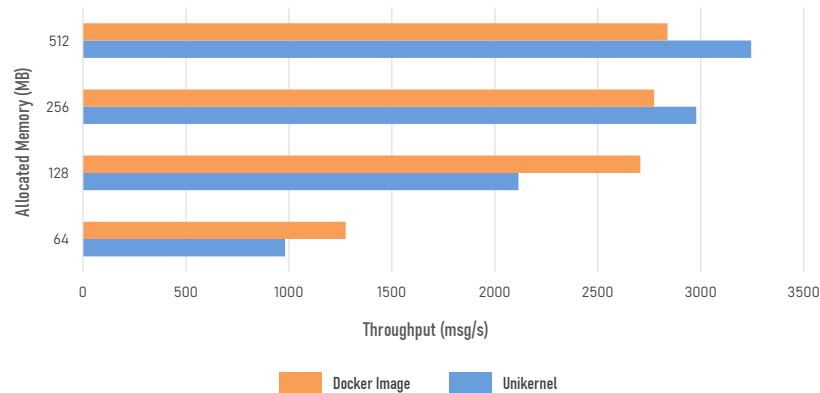


FIGURE 9.12: CPU-Bound Throughput for the Node.js Application under Varying Memory Constraints (1 CPU). A performance crossover occurred below the 256 MB memory level, where Docker's stability provided a performance advantage.

Once again with Node.js, the CPU-bound workloads demonstrated similar crossover pattern as previously discussed. In environments with sufficient memory, the Nanos unikernel demonstrated a slight performance advantage, delivering 6.85–12.54% higher throughput than the Docker container. With memory pressure being low, the primary differentiator is runtime overhead. As the allocated memory was reduced to 128 MB, the Docker container became the superior platform, outperforming the unikernel. The throughput of the Nanos unikernel decreased significantly (from nearly 2,977.81 req/s to 2114.19 req/s then to 980.90 req/s) as shown in Figure 9.12. In contrast, the Docker container's performance degraded more gracefully, maintaining a relatively stable throughput until a significant degradation at 64 MB. The latency results for the Node.js application, shown in Figure 9.13, confirmed this trend.

## 9.5 Discussion

The evaluation results highlight how unikernels (Nanos) and containers (Docker) differ in their architectural design and in how they impact application performance across various workloads, programming languages, and resource constraints. Similar to the conclusion from the previous chapter regarding microservice frameworks, there is no universal “best” platform; the optimal choice depends on the specific deployment context. Based on the conducted benchmarks, four key conclusions can be drawn.

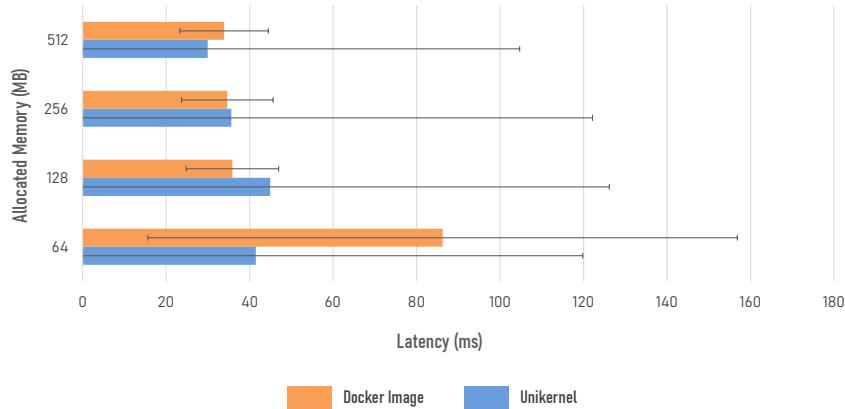


FIGURE 9.13: Average Latency and Standard Deviation for the Node.js Application (1 CPU). The chart highlights the increase in latency and performance jitter for the unikernel at 128 MB and below.

### 9.5.1 Static Properties: Unikernel Agility

The results strongly supported hypothesis **H1**, with unikernels achieving a startup time of approximately  $12\text{ ms}$ , compared to  $158\text{--}219.5\text{ ms}$  for Docker. This confirms the rapid instantiation advantages previously observed by Goethals et al. [480]. Within the context of the CRACI architecture, this capability represents a key enabler of resilience at the edge. Services deployed as unikernels can be scaled up or recovered from failures in milliseconds, enabling a level of responsiveness that is difficult to achieve with container-based deployments in time-critical Operational Intelligence Services.

Additionally, the observed *2.4 times reduction in image size* for Node.js applications aligns with Chen et al. [477], translating to lower storage and bandwidth requirements, supporting fast CI/CD cycles within CRACI’s Lifecycle Management pillar.

### 9.5.2 I/O Throughput: Kernel vs. Runtime Efficiency

The second hypothesis (**H2**) proposed that performance would be highly dependent on workload characteristics, and the I/O-bound results confirmed this. In the resource-rich, multi-core scenario, Docker outperformed Nanos for Go applications due to the highly optimized, kernel-level networking stack of the Linux host and Go’s efficient utilization of multi-core environments.

In contrast, Nanos outperforms Docker with the single-threaded Node.js application highlighted a different bottleneck. Here, the limiting factor was not the network stack but the Node.js event loop itself. In this context, the unikernel’s minimal system overhead provided an advantage, allowing the event loop to run with less interference and greater efficiency. It is also worth mentioning that Docker demonstrated lower jitter and fewer request drops, indicating its better predictability under heavy loads.

Overall, these results indicate that for I/O workloads, the optimal deployment paradigm choice depends on whether the bottleneck is the network layer (favoring Docker/Linux) or the application runtime (favoring the lighter unikernel).

### 9.5.3 CPU-Bound Performance: Specialization Advantage

For CPU-bound workloads, the evaluation showed that Go outperformed Node.js by up to 30 times, demonstrating the benefits of AOT compilation over JIT-based execution in high-load scenarios. In multi-core environments, Docker achieved higher throughput for Go due to the mature Linux scheduler efficiently distributing workloads across cores, while Node.js remained bottlenecked by its single-threaded model. In single-core, constrained environments, however, Nanos consistently outperformed Docker for Go, emphasizing the benefits of unikernel minimalism for compute-heavy tasks where every CPU cycle is critical. For Node.js, Docker again demonstrated more stability under severe memory constraints, maintaining throughput where the unikernel struggled. The stability of a standard Linux environment is critical to Node.js's performance under pressure, making the Node.js + Docker combination the more resilient option in severely constrained scenarios.

These findings validate the core value proposition of unikernels for specific, high-value industrial use cases. For Operational Intelligence Services at the extreme edge that perform intensive computation, such as real-time signal processing, data filtering, or cryptographic functions, a combination of unikernel and AOT-compiled languages offers a level of performance that containers cannot achieve on constrained hardware.

### 9.5.4 Crossover Point: Stability in Constrained Environments

In resource-constrained edge environments, the programming language determines the performance bottleneck. For Go, Nanos outperformed Docker by an order of magnitude across all memory levels in single-core tests. Although Nanos also demonstrated some degradation when reducing memory from 256MB to 128MB due to increased garbage collection, it still maintained a significant lead, demonstrating that unikernels can dedicate nearly all CPU cycles to execution, unlike Docker, which incurred additional system overhead.

For Node.js, the pattern differed: Nanos demonstrated an advantage at higher memory levels but suffered a sharp decline in throughput and increased latency below 128MB, while Docker maintained stable performance. These results reflect the unikernel's limited memory management capabilities, making JIT-based workloads like Node.js more vulnerable under severe constraints, partially confirming the **H2** hypothesis.

Overall, these findings suggest that for JIT-based services (e.g., Node.js, Java, Python), Docker remains the more resilient and practical option in constrained edge environments. In such cases, the container's resource overhead is justified by the stability and predictability of the Linux kernel.

## 9.6 Chapter Summary

The empirical evaluation results and discussion presented in this chapter demonstrate that selecting a deployment paradigm is not a binary decision but rather a complex architectural trade-off. The findings move beyond theoretical arguments and provide

a practical decision framework for engineers and architects implementing services for the CRACI architecture in Industry 4.0.

For Operational Intelligence Services performing intensive computation (e.g., analytics, signal processing) on highly resource-constrained edge devices, a unikernel running an AOT compiled language such as Go or Rust is the superior choice. This combination delivers an order-of-magnitude improvement in CPU efficiency, memory footprint, and startup latency. The minimal system overhead of unikernels enables maximal performance from limited hardware, making them particularly suitable for extreme-edge deployments.

For services implemented in JIT-compiled runtimes (e.g., Node.js, Java, Python) on edge devices where memory is a primary constraint, a Docker container is the more resilient and stable option. The mature memory-management and process-scheduling mechanisms of the Linux kernel provide the required stability, preventing the performance degradation observed with unikernels under severe memory pressure. In such environments, the architectural stability of Docker outweighs the raw resource savings offered by unikernels.

For high-throughput workloads on well-provisioned, multi-core edge servers or within cloud-based Strategic Intelligence Services, the optimal choice depends on the application's programming model. For parallel runtimes like Go or Rust, Docker containers perform better by utilizing the Linux kernel's optimized networking and scheduling capabilities. In contrast, for single-threaded runtimes where the runtime itself forms the bottleneck, unikernels offer a slight performance advantage due to their minimal system overhead, allowing applications to operate with less interference and achieve marginally higher throughput than Docker.

This chapter addressed **RQ2** by empirically quantifying the trade-offs between containers and unikernels. In summary, Docker containers and unikernels represent two distinct but complementary approaches, each with its strengths:

- *Containers* provide a versatile, developer-friendly, and mature ecosystem, making them the pragmatic default for a wide range of industrial applications.
- *Unikernels* deliver a minimal, secure, and highly efficient runtime for specialized, single-purpose tasks where every millisecond and megabyte count.

Together with the conclusions drawn in Chapter 8 regarding the selection of microservice development frameworks, the comparative analysis between containers and unikernels presented in this chapter provides a comprehensive perspective and several evidence-based insights for implementing the components of the CRACI architecture. It is clear that there is no single optimal choice applicable to all services under all runtime environments. Once again, these results indirectly confirm the suitability of a cloud-native architecture for large-scale systems.

Because the components are decoupled, system architects can independently select the most appropriate technology stack for each component based on its specific requirements. Cloud-native and distributed architectures offer great flexibility in optimizing individual system elements; however, this flexibility comes at the cost of increased complexity and a need for deep understanding of the architectural trade-offs. Often, these trade-offs are not as simple as "A is better than B" but involve factors that require careful research, analysis, and evaluation.

This dissertation will continue by focusing on the optimization of services, beginning with the challenge of service migration in cloud-native environments. Part V, which includes Chapters 10 and 11, discusses this topic in depth, addressing one of the challenges associated with service migration: the management of service state during migration.

## **Part V**

# **Managing State: Live Migration of Stateful Services**



# 10 Service Migration in Cloud-Native Environments

## 10.1 Introduction

To fully realize the potential of the Compute Continuum and the previously introduced CRACI architecture, services must be portable to enable load balancing, fault tolerance, and resource optimization across heterogeneous environments. The ability to dynamically migrate running services is therefore required to maintain flexibility in distributed systems.

A key distinction, however, exists between *stateless* and *stateful* microservices. As discussed in previous chapters, the Twelve-Factor App principles, which are already used in the design of the CRACI architecture, advocate statelessness to simplify scaling and management [449]. Nevertheless, many industrial applications, such as real-time control loops and interactive AMRs in the MAIA use case, require stateful operation. These services keep in-memory state to achieve low response latency, making reliance on external databases for every interaction impractical.

This need for statefulness creates a significant challenge: managing service state during the migration process. Migrating a service is not as simple as shutting it down and starting a new instance elsewhere. Stateful services require transferring the state (the data the service is currently working on, which is often stored in memory). Conventional migration methods, mostly based on techniques originally developed for VMs, treat a service's state as an opaque block of memory that needs to be transferred. Among them, simple *stop-and-copy* methods cause unacceptable downtime, while more sophisticated *live migration* techniques (e.g., pre-copy, post-copy) introduce significant implementation complexity and network overhead.

To address these limitations, this chapter introduces the Message-based Stateful Microservice Migration (MS2M) framework [491], a novel concept that utilizes the application's own communication infrastructure to orchestrate a live migration. By replaying the message stream, MS2M indirectly reconstructs the service state and reduces downtime. This aligns migration with CRACI's native event-driven and message-oriented architectural patterns. This chapter details the design, principles, and validation of MS2M, demonstrating its effectiveness as a lightweight, minimal-downtime migration mechanism for stateful services within the Compute Continuum.

## 10.2 Related Work

The primary goal of service migration is to transfer a running service from a source to a target host. The challenge of service migration in cloud environments, and, more broadly, across the *Compute Continuum* [492], has been extensively studied [493, 494]. A recent proposal, Ubiquitous Migration Solution (UMS), outlines a three-phase migration process: pre-migration, migration, and post-migration [495]. In the pre-migration phase, the source Coordinator requests migration, and the orchestrator creates a replica container to allocate the necessary resources. During migration, control messages are blocked, and a temporary *Frontman* container notifies users of unavailability. The container is then checkpointed, and its state is transferred. Finally, the source container is deleted, traffic redirected, and the *Frontman* container removed.

While UMS provides a robust migration framework, its reliance on the *stop-and-copy* approach leads to extended downtime, making it unsuitable for time-sensitive applications. This exposes a fundamental contradiction: stateful microservices are designed to reduce latency by retaining in-memory state, yet the downtime introduced by *stop-and-copy* undermines that very objective. As a result, *stop-and-copy* methods, as used in UMS, conflict with the primary purpose of stateful microservices, which is to guarantee low latency in dynamic environments.

To address these challenges, various *live migration techniques* have been developed to decouple the *downtime* from the *total migration time* and minimize the former. In *pre-copy migration*, memory pages are incrementally transferred while the service continues running, requiring only a brief interruption for final synchronization. For instance, Lu and Jiang [496] proposed MBDPC (Migration Based on Dirty Page Prediction and Compression), which utilizes machine learning to predict modified memory pages, reducing both data transfer volume and downtime. In contrast, *post-copy migration* suspends the service at the source, resumes it at the destination, and retrieves remaining memory pages on demand. Tazzioli et al. [497] integrated this approach into Kubernetes, achieving a reported 65% reduction in downtime. Further optimization is demonstrated by Bellavista et al. [498], who separated hot and cold states, significantly minimizing downtime.

A *hybrid pre- and post-copy* method combines the incremental data transfer of pre-copy migration with the on-demand fetching of post-copy techniques to balance downtime and total migration time. For example, Ma et al. [499] proposed a live migration method for Docker containers in edge environments, reducing file system synchronization by transferring only the top writable layer, cutting migration time by 56% to 80% under varying network conditions.

While effective in certain contexts, these *memory-centric techniques* introduce complexity in managing in-memory state and overlook a key principle of modern, event-driven microservices: in such systems, service state is not an arbitrary block of data but the accumulated result of the message history a service has processed [500]. This concept underpins the CRACI architecture and enables a paradigm shift in how stateful service migration can be achieved. Instead of relying on external memory-copying mechanisms, migration can utilize the system's existing communication fabric to replay the sequence of messages that originally formed

the service's state. This approach provides a simpler and more architecturally coherent alternative to low-level memory manipulation—a gap that the proposed Message-based Stateful Microservice Migration (MS2M) framework is designed to address.

## 10.3 The MS2M Framework: Concept and Design

This section presents the design of the proposed MS2M framework and the corresponding migration procedure.

### 10.3.1 Actors and Components

The MS2M framework involves three primary logical actors, which interact to coordinate the migration process:

1. *Source Host*: The server or node on which the microservice is currently running.
2. *Target Host*: The destination server or node to which the microservice will be migrated.
3. *Migration Manager*: A dedicated service responsible for orchestrating the migration process. It acts as the central controller, issuing commands to the hosts (nodes) and managing the migration from start to finish. In a microservice ecosystem, this entity itself can be deployed as a microservice, enabling self-managing capabilities within the application.

These actors communicate over the platform's existing control plane. The underlying technical components include the container runtime on each host (e.g., Docker, Podman), a checkpoint/restore engine (e.g., CRIU), and the application's message broker (e.g., RabbitMQ).

### 10.3.2 The Five-Phase Migration Process

The MS2M procedure is a carefully coordinated sequence of five phases, designed to guarantee state consistency and minimize the impact on the running service. The detailed process is illustrated in Figure 10.1.

1. **Checkpoint Creation.** The migration process begins when the Migration Manager sends a `ServicePauseRequest` to the source host. This triggers a critical preparation sequence:
  - The live service instance is temporarily paused and unsubscribed from its primary message queue to prevent message loss.
  - A secondary message queue is created on the message broker to mirror the primary queue from this point onward.
  - A checkpoint of the paused service is created using the underlying engine (e.g., CRIU).

- Once the checkpoint process finishes, the original service instance on the source host is resumed and re-subscribes to the primary queue. It continues processing requests as normal.
2. **Checkpoint Transfer.** Once the source service is back online, the checkpoint image is transferred from the source host to the target host. This transfer occurs in the background without any further interruption to the running service instance.
  3. **Service Restoration.** Upon successful transfer of the checkpoint, the Migration Manager issues a `ServiceRestoreRequest` to the target host. The target host restores the service instance from the checkpoint but configures it to subscribe only to the secondary message queue. To initiate the safe replay phase, the Migration Manager immediately enqueues a special `START_REPLAY` control message as the first message for the restored instance to consume.
  4. **Message Replay.** The newly restored service instance on the target host consumes its first message, recognizes it as the `START_REPLAY` control message, and sets an internal flag to enter a special *replay mode*. In this mode, it reprocesses all messages accumulated in the secondary queue to catch up its internal state while suppressing external outputs (no message publishing, database writes, or API calls). All actions are confined to in-memory updates, preventing side effects. Meanwhile, the source service continues handling live traffic.
  5. **Finalization.** Once the target service has replayed all messages in the secondary queue up to a designated point, the Migration Manager initiates the final hand-off by enqueueing a final `END_REPLAY` control message to the secondary queue.
    - The target instance consumes this control message, exits its *replay mode*, and immediately switches its subscription from the secondary queue to the primary message queue to begin processing live requests.
    - Simultaneously, the Migration Manager issues a final `ServiceStop` command to the source host, terminating the original instance.
    - The secondary queue is deleted, completing the migration process.

With this migration procedure, downtime is limited to the checkpointing phase, as the replay phase (state synchronization) is executed entirely in the background. The implementation of the prototype and its evaluation results are presented in the following sections.

### 10.3.3 Empirical Evaluation

To validate the MS2M framework and evaluate its performance, a proof-of-concept evaluation was conducted based on an industrial use case. The evaluation builds upon the use case of the MAIA architecture introduced in Chapter 7, reusing the offloaded stateful service for Autonomous Mobile Robots (AMRs) scenario. This use case was chosen as it encapsulates the essential characteristics of modern industrial systems: interactivity, high latency sensitivity, and inherent statefulness.

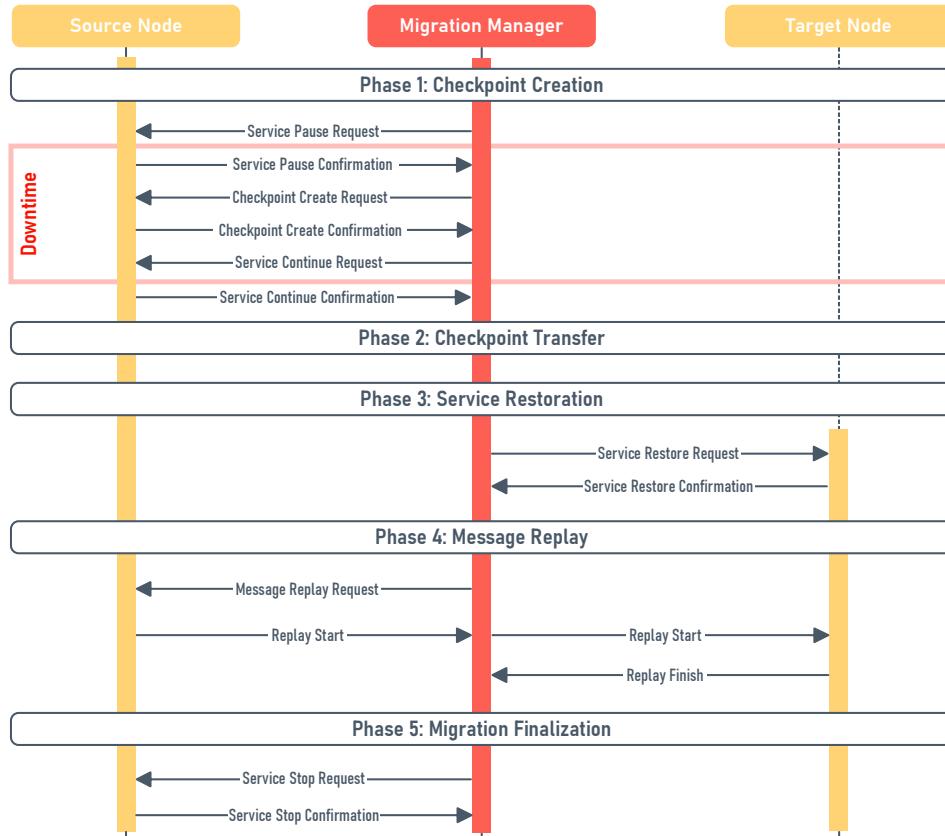


FIGURE 10.1: The proposed MS2M migration process.

## Experimental Setup

In a modern smart factory, AMRs navigate between different operational zones, each potentially managed by a local MEC server to ensure low-latency control. The evaluation simulated the migration of an AMR's stateful control service as it moved from one zone to another. This service processes the AMR's real-time camera feed and location data to maintain its operational context (e.g., current task, planned path), making state preservation during migration a critical requirement. The experiment setup was as follows:

- **Edge Zones (Hosts)**: Two distinct factory zones were emulated using two separate virtual machines on Google Cloud Platform (GCP): one representing the MEC server for *Zone A* and the other for *Zone B*. Both VMs were configured as `e2-medium` instances (2 vCPUs, 4 GB memory).
- **Industrial Network (Message Broker)**: The factory communication backbone was represented by a RabbitMQ (v3.8.11) instance hosted on a separate `e2-medium` VM.
- **AMR Service (Container)**: The stateful AMR control service was containerized using Podman (v3.1.0) on Ubuntu 20.10. Its state reflects the AMR's live operational context.
- **Orchestration (Migration Manager)**: A Python-based Migration Manager was implemented to orchestrate the migration, interacting with the hosts via Podman's REST API.

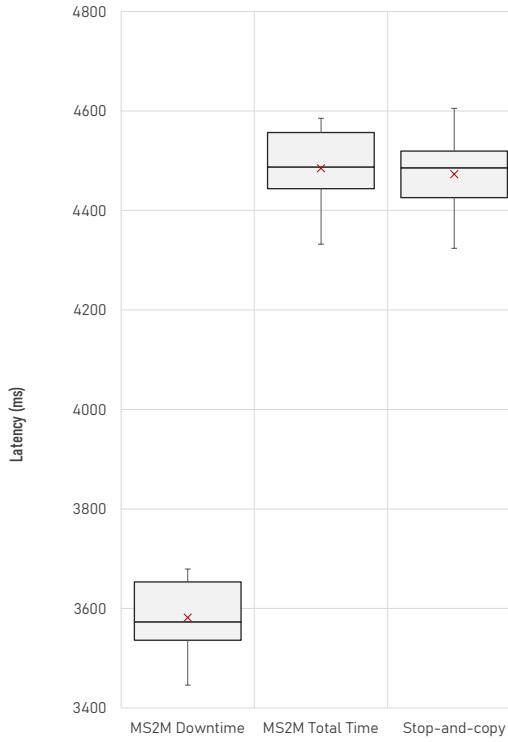


FIGURE 10.2: Overall time performance of MS2M compared to the baseline stop-and-copy [491].

To establish a performance baseline, a traditional stop-and-copy migration was implemented, where the service was fully stopped on the source host before being transferred and restarted on the target. Both MS2M and the baseline were evaluated using two relevant metrics, consistent with prior research [501, 502, 503]:

- *Total Migration Time*: The full duration from the checkpoint initiation until the service was fully operational and synchronized on the target (Zone B).
- *Service Downtime*: The period during which the AMR’s control service was completely unavailable.

The evaluation simulated the continuous operation of an AMR, with a script that automatically triggered 25 migration cycles from Zone A to Zone B.

## Results and Findings

The comparative results provide quantitative evidence of MS2M’s effectiveness. As shown in Figure 10.2, MS2M reduced the service downtime to an average of 3,581.84 ms, a 19.92% improvement over the 4,472.72 ms of downtime recorded with the stop-and-copy method (it should be noted that in stop-and-copy, the total migration time and the downtime are the same). This is possible because MS2M limits downtime to the brief pause for checkpoint creation (Phase 1), while keeping the original service operational during checkpoint transfer and message replay. By doing so, it minimizes service interruption and confirms that the message-based replay mechanism is a viable and more efficient alternative for latency-sensitive stateful services.

The total migration time under MS2M averaged 4,852.86 ms, representing an overhead of 8.5% over the stop-and-copy baseline. This additional delay results directly from the background replay phase and highlights an important architectural trade-off: MS2M exchanges a longer background state synchronization process for reduction in perceived downtime.

While the results are promising, a deeper analysis exposed the performance bottlenecks and architectural limitations of the framework in its current proof-of-concept implementation.

- *Performance Bottlenecks:* Further profiling (Figure 10.3) showed that the checkpoint and restore operations dominate the overall migration time, consuming up to 88.04%. As checkpoint duration scales linearly with memory size [504], this inherently limits MS2M’s performance for memory-intensive workloads. A secondary bottleneck is the risk of *unbounded message replay*, where high message rates can lead to excessively long migration times.
- *Architectural Limitations of the Message Replay Model:* The evaluated MS2M protocol handles side-effects through a "sandboxed replay" mode. While effective, this approach introduces significant architectural trade-offs:
  - *Invasiveness:* To safely suppress external outputs, the application developer must explicitly modify the service’s core business logic to include a special *replay mode*. The service must be aware of the migration process and alter its behavior accordingly (e.g., by checking a flag before executing any database write or API call). Such coupling between application logic and migration control adds significant implementation complexity.
  - *Fragile:* The current model lacks robustness for services involving non-deterministic operations (where the same input may not produce the same output such as operations involve timestamps or random values) or dependencies on external state. These factors can cause divergence between the replayed and original states, reducing consistency guarantees.

In summary, the proof-of-concept validated the core principles of the MS2M framework, demonstrating that a communication-aware migration can significantly reduce service downtime. However, the evaluations also made clear that achieving general-purpose applicability across different workload types requires a more robust and non-invasive approach. This will be explored in the next chapter.

#### 10.3.4 Chapter Summary

This chapter addressed the challenge of migrating stateful services within the dynamic and heterogeneous environments of the Compute Continuum and Industry 4.0. It showed that conventional migration techniques, which rely on low-level memory transfer, are often too disruptive to meet the high-availability requirements of industrial systems.

To overcome these limitations, the Message-based Stateful Microservice Migration (MS2M) framework was introduced. This novel approach shifts the focus from copying state to *indirectly reconstructing it* from an application’s message history.

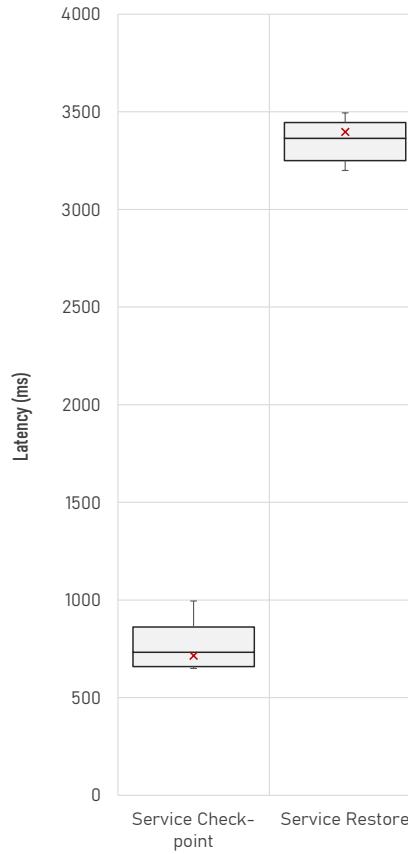


FIGURE 10.3: Overall time performance of MS2M [491].

By utilizing the existing messaging infrastructure in cloud-native systems, MS2M synchronizes state through controlled message replay, allowing the service to remain operational throughout most of the migration process.

The experimental evaluation results confirmed the core principles of this design. MS2M reduced service downtime by 19.92% compared to the traditional stop-and-copy baseline, demonstrating its practicality as a minimally disruptive migration method for stateful microservices. The results also indicate a broader architectural insight: communication-aware migration, which relies on application-level semantics, offers a more effective approach to managing state in distributed systems than treating services as opaque black boxes.

At the same time, the evaluation also exposed trade-offs in the current prototype's replay model. While the *sandboxed replay mechanism* successfully prevents duplicate side effects, it remains invasive. That means the current implementation requires modifications to the service logic, and it lacks robustness when dealing with non-deterministic or externally dependent services. These findings clearly suggest the need for further improvement.

Building on this initial prototype, the next chapter explores how the MS2M framework can be integrated into an orchestration environment, specifically Kubernetes, to enable message-based migration at scale. It also presents several design modifications and optimizations aimed at addressing the limitations identified in this chapter.

# 11 Optimizing Stateful Microservice Migration in Kubernetes

## 11.1 Introduction

The previous chapter presented the MS2M framework as a feasible proof-of-concept, demonstrating that utilizing an application’s messaging infrastructure can significantly reduce service downtime during migration. However, its evaluation also highlighted the limitations of the initial prototype: a potential performance bottleneck under high message rates. Moreover, in practice, cloud-native applications are often deployed and managed through *orchestrators*. To make MS2M more applicable in real-world scenarios, this chapter integrates it into the de facto standard orchestrator, Kubernetes, and introduces several design modifications addressing the limitations identified in the initial prototype.

Transitioning into a fully orchestrated environment introduces several technical complexities. Kubernetes operates on a *declarative, desired-state* model, which conflicts with the simple, imperative commands of the MS2M prototype. A robust migration solution must therefore be integrated into Kubernetes’ control loops. Moreover, it can no longer target a single container but must align with Kubernetes’ hierarchical resource model, including higher-level abstractions such as `Pods`, `Deployments`, and the unique identity of `StatefulSets`.

This chapter addresses these challenges and presents extensions that enable integration, building on the author’s previously published, peer-reviewed work [505]. The key contributions of this chapter are therefore threefold:

- *Kubernetes Integration*: The chapter details the architectural integration of MS2M with the Kubernetes ecosystem, utilizing its experimental Forensic Container Checkpointing (FCC) feature as the underlying technical mechanism for creating stateful checkpoints of running `Pods`.
- *Performance Optimization under High Load*: A novel Threshold-Based Cutoff Mechanism is introduced to address the unbounded message replay problem identified in the previous chapter, to provide predictable migration times even under high traffic.
- *Extended Support for Complex Workloads*: The MS2M process is adapted to support one of Kubernetes’ most critical workload types for stateful services—the `StatefulSet`—demonstrating the concept’s applicability.

The remainder of this chapter is organized as follows. Section 11.2 describes the architectural adaptations required to integrate MS2M with the Kubernetes control plane. Section 11.3 introduces the performance optimization mechanism. Section 11.4 explains the procedure adaptations necessary for migrating `StatefulSets`. Section 11.5 presents an evaluation of the proposed enhancements and discusses their performance trade-offs. Finally, Section 11.6 summarizes the findings and concludes the chapter.

## 11.2 Integrating MS2M with Kubernetes

To transform the MS2M framework from a conceptual model into a practical tool for industrial environments, it must be integrated with existing orchestration platforms such as Kubernetes. This integration requires adapting the original MS2M procedure to align with Kubernetes' declarative API and its resource management model.

### 11.2.1 The Adapted Four-Actor Model

The original three-actor model of the MS2M framework is extended to a four-actor architecture to accommodate the Kubernetes control plane. In this updated model, direct communication between the Migration Manager and the host nodes is replaced by interactions mediated through the Kubernetes API Server. This ensures that all actions comply with the cluster's security policies, resource constraints, and desired-state model. The revised model includes the following actors:

- *Source Node & Target Node*: Kubernetes worker nodes that host the Pods. They execute migration-related commands received from the control plane via their local `kubelet` agents.
- *Kubernetes API Server (AS)*: The central control component of the Kubernetes cluster. All administrative and operational commands are sent as requests to the API Server.
- *Migration Manager*: This remains the orchestrator of the migration process. However, in this revised model, it interacts exclusively with the Kubernetes API Server to issue migration commands and monitor migration progress, rather than communicating directly with the nodes.

One important technical foundation for integrating MS2M into Kubernetes is the *Forensic Container Checkpointing* (FCC) feature, introduced as an experimental feature in v1.25<sup>1</sup>. FCC extends the `kubelet` to allow authorized users to create and restore checkpoints of running containers using CRIU. Instead of invoking CRIU (Checkpoint/Restore In Userspace)<sup>2</sup> directly as in the original MS2M prototype, the framework now uses the FCC API as the low-level mechanism to implement the checkpoint creation and service restoration phases in the MS2M process.

---

<sup>1</sup><https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG/CHANGELOG-1.25.md>

<sup>2</sup><https://criu.org/>

### 11.2.2 The Integrated Pod Migration Procedure

With these changes, the migration of a single stateful Pod is presented in this section. The procedure follows the original five-phase MS2M process, but each phase is executed through corresponding Kubernetes API requests, as illustrated in Figure 11.1.

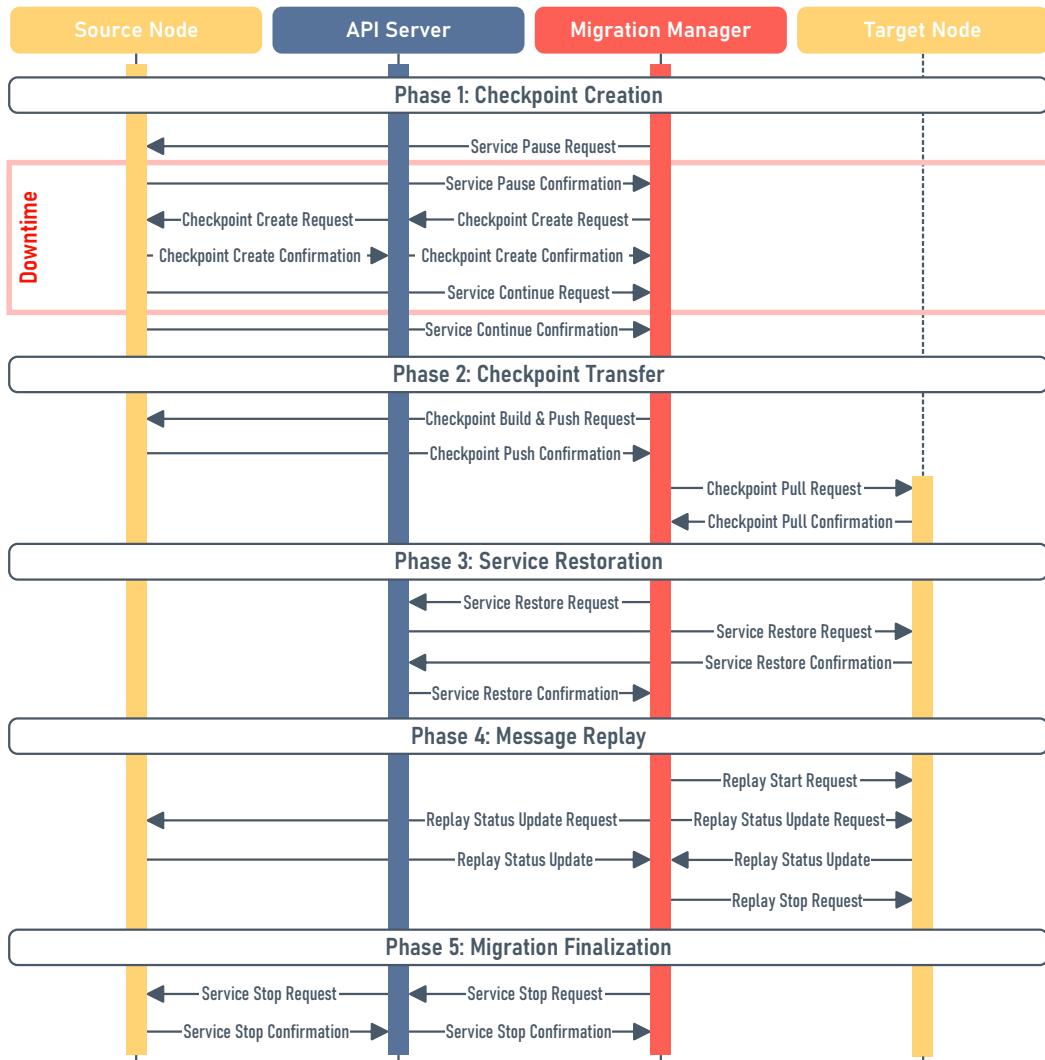


FIGURE 11.1: Detailed Migration Process of MS2M in Kubernetes for individual Pods.

1. **Checkpoint Creation.** The Migration Manager initiates the migration by sending a checkpoint request to the Kubernetes API Server, targeting the container within the source Pod.
  - The API Server forwards this request to the `kubelet` on the Source Node.
  - The `kubelet` pauses the container, creates the checkpoint using FCC, and then immediately resumes the container.
  - Simultaneously, the Migration Manager triggers the message replay mechanism by creating a secondary message queue to buffer all new incoming messages.

2. **Checkpoint Transfer.** This phase introduces a change compared to the original MS2M prototype. Instead of performing a direct host-to-host file transfer, the checkpoint is now handled through a container registry (as illustrated in Figure 11.2), decoupling the source and target nodes.
  - The Migration Manager instructs the Source Node to package the checkpoint into a dedicated container image.
  - This image is then pushed to a central Container Registry (e.g., Google Artifact Registry or Docker Hub).
3. **Service Restoration.** The Migration Manager generates a new Pod manifest mirroring the configuration of the original Pod. The manifest includes an annotation directing the `kubelet` on the Target Node to restore the container from the checkpoint image stored in the registry.
  - The manifest is submitted to the Kubernetes API Server, which schedules the new Pod on the Target Node.
  - To prepare the target Pod for state synchronization, the Migration Manager enqueues a special `START_REPLAY` control message as the first message into the secondary queue.
4. **Message Replay.** Once the restored Pod is running, it subscribes to the secondary message queue. It first consumes the `START_REPLAY` message, which instructs it to enter a *replay mode*. In this mode, it reprocesses all mirrored messages to synchronize its internal state while suppressing external outputs. During this period, the original Pod on the Source Node continues to handle live traffic.
5. **Finalization.** Upon completing the Message Replay phase, the Migration Manager orchestrates the final hand-off by enqueueing an `END_REPLAY` control message into the secondary queue. The finalization sequence is as follows:
  - The target Pod consumes the `END_REPLAY` message, exits its replay mode, and immediately switches its subscription from the secondary queue to the primary message queue to begin processing live requests.
  - The Migration Manager then sends a deletion request to the API Server to terminate the original Pod on the Source Node, completing the migration.

With the introduction of the Kubernetes API Server, both the actors and the details of the migration procedure have been adapted. These changes enable the MS2M framework to handle the migration of entire Pods rather than individual containers.

### 11.3 Performance Optimization: The Threshold-Based Cutoff Mechanism

Although the MS2M framework successfully minimizes downtime, the initial analysis identified a potential performance bottleneck under high-traffic conditions. As the rate of incoming messages approaches the service's maximum processing capacity, the message replay phase (Phase 4) can become excessively long, or in extreme

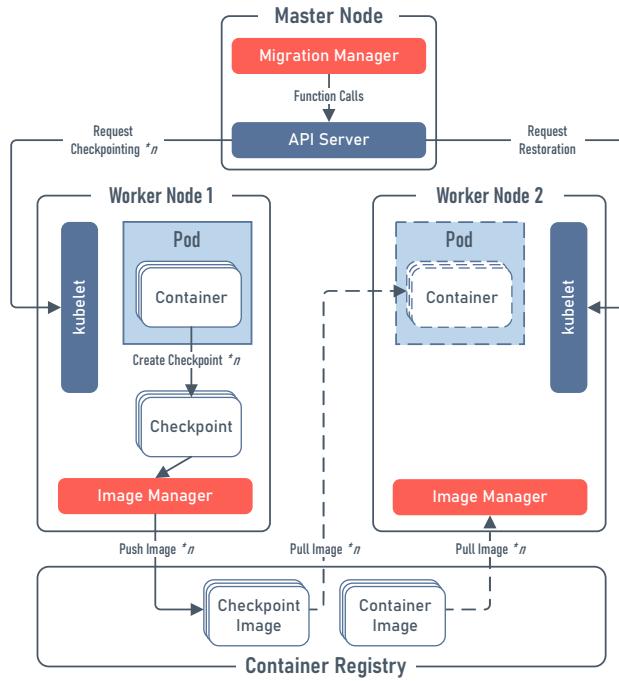


FIGURE 11.2: Overview of MS2M integration into Kubernetes [505].

cases, theoretically unbounded. This prolonged replay not only consumes significant resources on the target node but also delays the completion of the migration, reducing the system's overall predictability.

To address this issue, a heuristic optimization strategy named *Threshold-Based Cutoff Mechanism* is introduced. This mechanism imposes a deterministic limit on the duration of the state synchronization phase, so that migrations complete within a predictable time window even under heavy load. Details of this mechanism will be presented in this section.

### 11.3.1 Problem Formulation: Modeling the Replay Queue

To analyze the behavior of the replay phase more formally, the process can be modeled using principles from queuing theory. The system is represented as a classic  $M/M/1$  queue, mathematically capturing the relationship between message arrival rate and processing times. This model assumes that messages arrive according to a Poisson process (rate  $\lambda$ ) while a single server (the target microservice) processes them with exponentially distributed service times at rate  $\mu_{\text{target}}$ .

Although the assumption of a memoryless process (meaning the probability of an event occurring is independent of past events) simplifies what are often *bursty* (messages arrive in irregular bursts rather than at a steady rate) or *deterministic* (messages arrive at a fixed rate) industrial workloads, the  $M/M/1$  model can still be a useful mathematical model for defining the system's stability condition. The main parameters influencing the replay time are:

- $\lambda$ : Incoming message rate at the source microservice (messages per second).
- $\mu_{\text{target}}$ : Message processing rate at the target microservice (messages per second).

- $T_{\text{replay\_max}}$ : Maximum acceptable duration for message replay phase at the target microservice.
- $T_{\text{accum}}$ : The period during which messages accumulate in the secondary queue.

The total number of messages to be replayed, denoted as  $N_{\text{messages}}$ , is given by:

$$N_{\text{messages}} = \lambda \cdot T_{\text{accum}}$$

The time required to replay these messages,  $T_{\text{replay}}$ , is therefore:

$$T_{\text{replay}} = \frac{N_{\text{messages}}}{\mu_{\text{target}}} = \frac{\lambda \cdot T_{\text{accum}}}{\mu_{\text{target}}}$$

In MS2M, the source Pod continues operating while the target Pod replays buffered messages. This creates a situation where the target service, with a processing rate of  $\mu_{\text{target}}$ , must catch up to a message queue that is still growing at the arrival rate of  $\lambda$ .

This system can only reach a synchronized state if the processing rate of the target service instance is greater than the arrival rate ( $\mu_{\text{target}} > \lambda$ ). If  $\lambda \geq \mu_{\text{target}}$ , the replay queue will grow indefinitely, and the migration will never complete. This scenario, where the migration time becomes unbounded, must be avoided to ensure system stability and predictability.

### 11.3.2 The Cutoff Mechanism: A Deterministic Solution

To guarantee convergence, the open-ended replay process must be made deterministic. The cutoff mechanism achieves this by turning state synchronization into a *finite batch problem*. By terminating the source service after a calculated duration, the mechanism limits the period during which new messages can accumulate. This results in a finite set of  $N$  messages with a bounded and predictable replay time, ensuring that the migration always completes within a specified QoS window.

This is achieved by stopping the source service after a specific duration,  $T_{\text{cutoff}}$ , measured from checkpoint initiation. This ensures that the total number of messages to be replayed ( $N_{\text{messages}}$ ) is bounded. A QoS parameter,  $T_{\text{replay\_max}}$ , defines the maximum acceptable replay duration. This parameter is operationally important because, during the replay period, both service instances run in parallel, duplicating workload and increasing resource consumption. The condition  $T_{\text{replay}} \leq T_{\text{replay\_max}}$  must therefore be true. By rearranging the formula to solve for the maximum allowed accumulation time, the cutoff threshold is determined as:

$$T_{\text{cutoff}} = T_{\text{accum}} \leq \frac{T_{\text{replay\_max}} \cdot \mu_{\text{target}}}{\lambda}$$

Applying this threshold guarantees that the replay phase completes within the required time frame, keeping the migration predictable and controlled even under heavy load.

### 11.3.3 The Optimized Migration Procedure

The introduction of the cutoff threshold  $T_{\text{cutoff}}$  modifies the migration procedure, as illustrated in Figure 11.3. The changes are grouped in the Phase 4.2, these include:

- **Initiate Service Stop:** Instead of waiting for the replay to complete, the Migration Manager proactively stops the source service once the  $T_{\text{cutoff}}$  period has elapsed. It sends a `DELETE` request for the source Pod to the Kubernetes API Server, terminating the original service instance. Simultaneously, it instructs the message broker to stop mirroring messages from the primary queue to the secondary queue. This action freezes the secondary queue, creating a finite, bounded set of messages that must be replayed. Messages continue to accumulate in the primary queue, but they will be handled by the new service instance after the hand-off.
- **Bounded Message Replay:** The target Pod's state synchronization now occurs in a deterministic process:
  1. **Bounded Replay:** The target service processes the now-finite batch of messages from the secondary queue. This synchronizes its state to the exact moment the source Pod was terminated and the message mirroring was stopped.
  2. **Live Catch-up:** Once the secondary queue is empty, the target service's state is fully synchronized. It immediately subscribes to the primary queue and begins processing the backlog of live messages that accumulated while it was replaying. The migration is now complete, and the secondary queue is deleted.

This mechanism changes the trade-off. It introduces a secondary period of downtime (from the moment the source Pod is stopped until the target Pod completes the replay), but it guarantees that the total migration time remains predictable and bounded. By allowing operators to configure  $T_{\text{replay\_max}}$ , this mechanism provides direct control over the balance between service downtime and overall migration duration. Details of about this trade-off will be examined in the later evaluation.

## 11.4 Extending Support for StatefulSets

While migrating Individual Pods is the first step for MS2M's integration in Kubernetes, many critical industrial applications, such as databases, distributed logs, or clustered message brokers, are typically deployed using Kubernetes' StatefulSets. These workloads introduce a stricter set of constraints that alter the migration process. This section details the unique challenges posed by StatefulSets and presents an adapted MS2M procedure to support their migration.

### 11.4.1 The Challenges of StatefulSets

StatefulSets are specifically designed to manage stateful applications by providing guarantees that standard Deployments do not. The two most important guarantees that affect the migration process are:

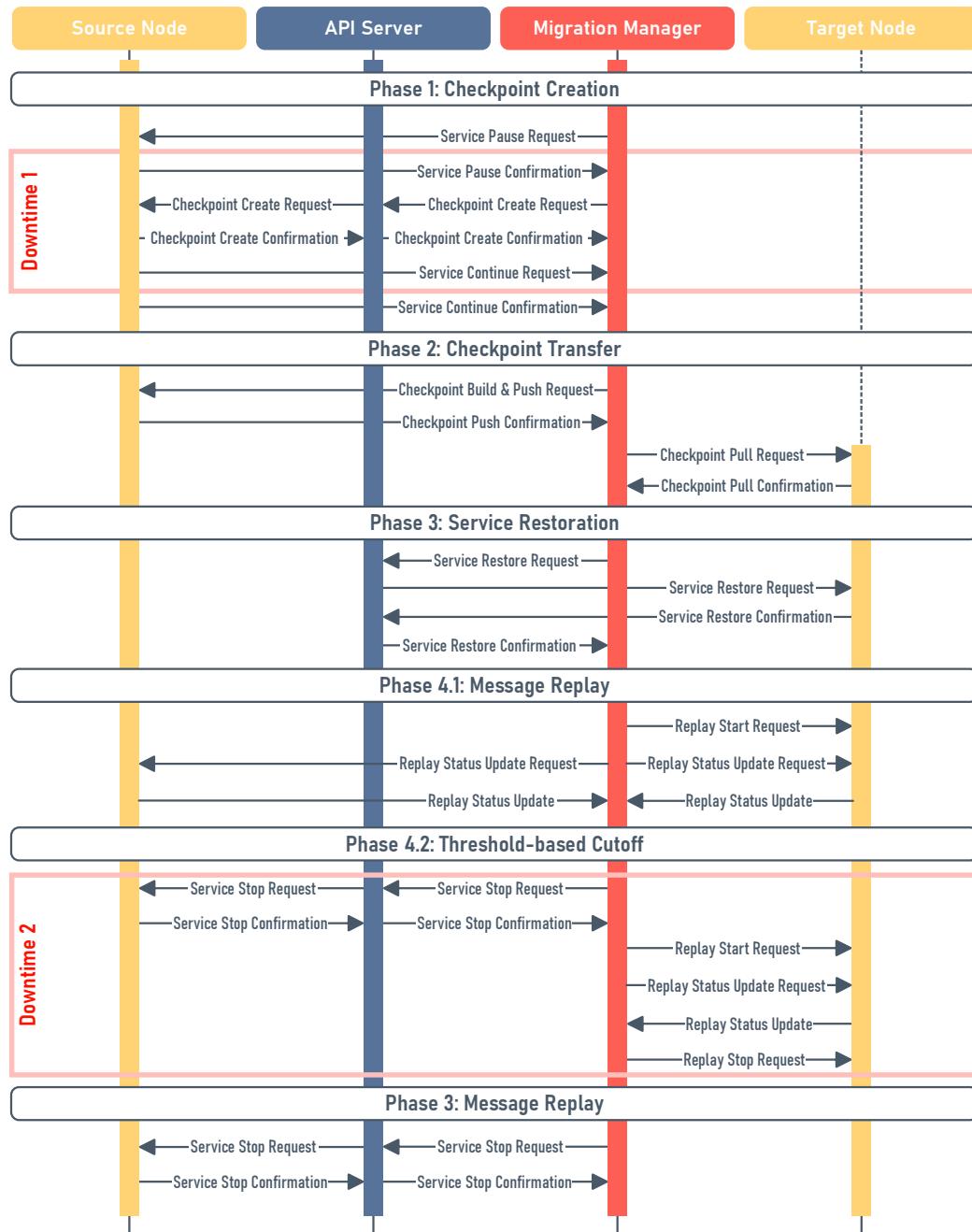


FIGURE 11.3: Detailed migration process of MS2M in Kubernetes for Individual Pods with Threshold-Based Cutoff Mechanism [505].

- *Stable, Unique Network Identity:* Each Pod in a `StatefulSet` is assigned a stable, predictable hostname based on its ordinal index (e.g., `my-app-0`, `my-app-1`). This identity is permanent and unique across the cluster.
- *Stable, Persistent Storage:* Each Pod is typically associated with its own `PersistentVolume`, which guarantees data persistence even if the Pod is rescheduled.
- *Heightened Risk of Side-Effect Duplication:* Since `StatefulSets` are the standard for deploying databases and other persistent services, the risk of side-effect duplication during message replay is a key concern. The naïve replay of messages against a service that writes to a `PersistentVolume` would lead to data corruption. Therefore, any migration strategy for `StatefulSets` must address the application-level consistency challenge.

The unique network identity creates a major challenge for live migration. Kubernetes strictly prohibits two Pods with the same `StatefulSet`-managed identity from running simultaneously. This means the original MS2M concept, which relies on having both the source and target Pods running concurrently during the message replay phase, cannot be directly applied. The source Pod must be terminated before a new Pod with the same identity can be created on the target node.

### 11.4.2 The Adapted Migration Process for StatefulSets

To accommodate these constraints, the MS2M procedure for `StatefulSets` must be modified. The core principles of checkpointing and message-based state reconstruction remain, but the timing of the service termination is different, resulting in a trade-off between maintaining identity and minimizing downtime. The adapted procedure, as illustrated in Figure 11.4, proceeds as follows:

- **Phase 1: Checkpoint Creation** and **Phase 2: Checkpoint Transfer.** These phases remain unmodified.
- **Phase 3: Service Stop (Pre-Restoration).** Before the target Pod can be created, the Migration Manager must first delete the source Pod. It sends a `delete` request to the Kubernetes API Server. This creates a period of downtime, as no active instance is available to process messages.
- **Phase 4: Service Restoration.** Once Kubernetes confirms the termination of the source Pod, the Migration Manager creates the new Pod on the target node, instructing it to restore from the checkpoint image. To prepare for the replay, the Migration Manager enqueues a `START_REPLAY` control message into the secondary queue.
- **Phase 5: Message Replay.** The newly restored Pod starts and subscribes only to the secondary queue.
  - Upon consuming the `START_REPLAY` message, it enters *replay mode* (suppressing side-effects).

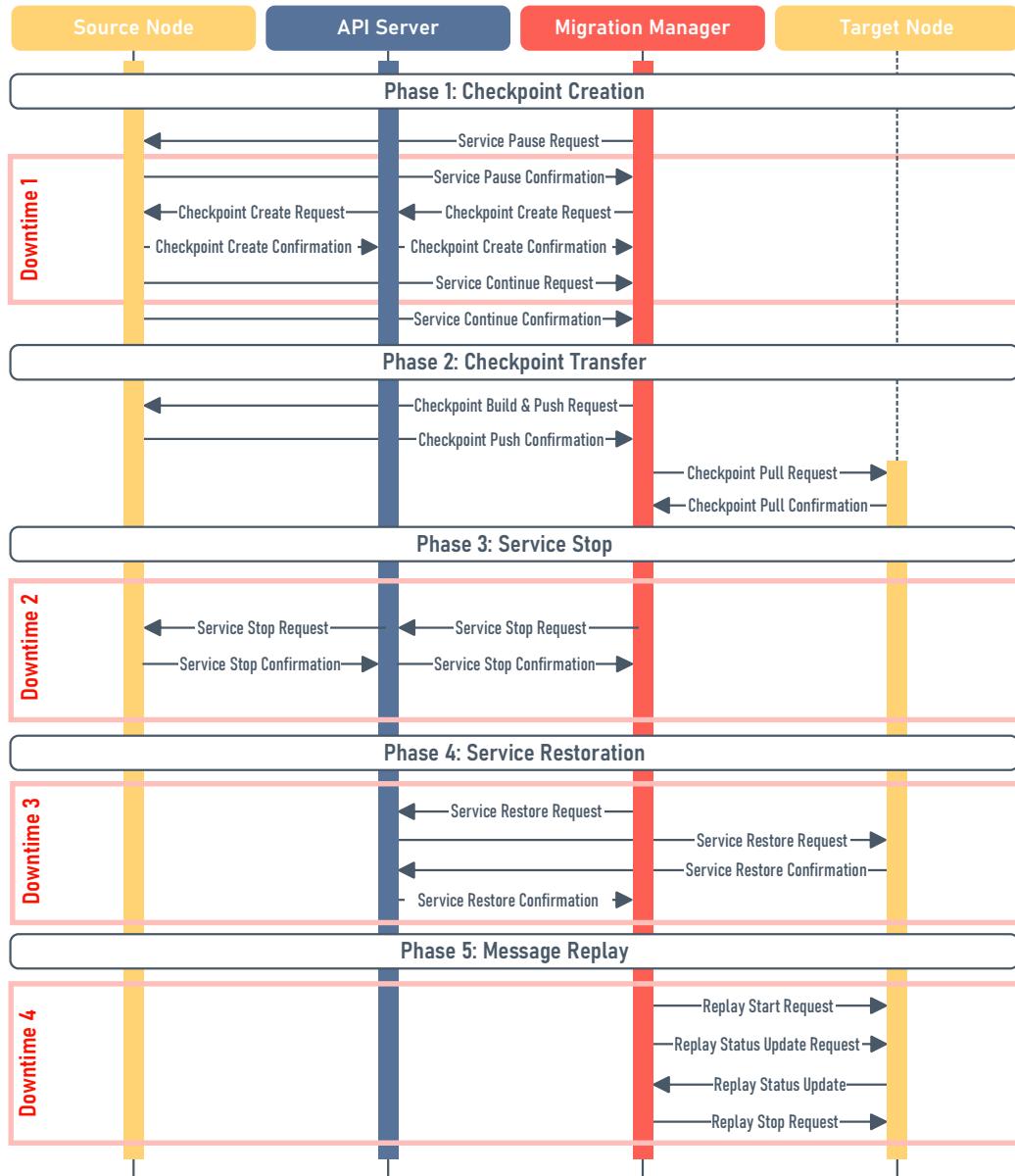


FIGURE 11.4: Detailed migration process of MS2M in Kubernetes for StatefulSet Pods [505].

- It then processes the bounded number of messages from the secondary queue, synchronizing its state to the point just before the original Pod was terminated.
- To finalize, the Migration Manager enqueues an END\_REPLAY message. Upon consuming this, the target Pod exits replay mode, switches its subscription to the primary message queue, and begins processing live traffic. This finalizes the migration process.

### 11.4.3 Analysis of the Trade-off

This adapted procedure successfully enables the migration of StatefulSet Pods while preserving their identity and state. However, this improvement comes at the

cost of longer downtime. Unlike the individual Pod migration, the downtime for a `StatefulSet` now includes not only the initial checkpointing pause but also the entire duration of the service restoration and message replay phases.

While this results in higher downtime compared to the migration procedure for individual Pods, it is still an improvement over a traditional stop-and-copy migration, where the state would have to be checkpointed and restored during downtime. The MS2M approach for `StatefulSets` still utilizes the efficiency of checkpointing and the indirect state reconstruction via message replay, ensuring that the total period of service downtime is minimized within the limits imposed by `StatefulSet` constraints.

## 11.5 Evaluation and Analysis

To evaluate the performance of the integrated and optimized migration strategies, a series of evaluations were conducted. This section details the experimental setup and presents a comparative analysis of the four migration procedures:

- *Stop-and-Copy*: The baseline cold migration method.
- *MS2M for Individual Pods*: The MS2M live migration strategy integrated into Kubernetes.
- *MS2M with Cutoff Mechanism*: The optimized live migration strategy using the Threshold-Based Cutoff Mechanism.
- *MS2M for StatefulSets*: The adapted procedure for `StatefulSet`.

### 11.5.1 Experimental Setup

Microservices were implemented in *Java 17* with *Spring Boot 3.1.5*, while the Migration Manager was developed in *Python 3.10*. Two microservice types were deployed:

- *Producer*: A single-instance Pod that continuously sends messages to RabbitMQ.
- *Consumer*: A scalable, multi-instance `StatefulSet` consuming messages from RabbitMQ queues.

The evaluation ran on *Google Compute Engine* with three *e2-medium* VMs (2 vCPUs, 4 GB RAM, *Ubuntu 20.04* with *cgroup v1* for CRIU compatibility). Services were orchestrated by *Kubernetes v1.30* with *CRI-O v1.28* and *CRIU* for checkpoint/restore. *Rsync* over *SSH* transferred checkpoint files, and *Buildah 1.37.0* created OCI-compliant images for migration, stored in *Google Artifact Registry*. *Persistent Volumes* were used to guarantee data consistency, and *RabbitMQ v3.13* managed inter-service communication.

### 11.5.2 Evaluation Results

The following parameters were configured for the evaluation: the *message rate* (messages sent per second by the producer) to vary workloads, and the *message processing time* (average consumer processing time). A baseline of 10 messages per

second with a processing time of 50 milliseconds (max processing capacity: 20 messages/sec) was set. The message rate was then varied while keeping processing time constant to analyze its impact. Each migration strategy was evaluated across these configurations, with each test repeated 10 times.

Similar to the evaluation presented in Chapter 10, this analysis also uses *Total Migration Time* and *Downtime* as key performance metrics.

### Stability evaluation

In the baseline stop-and-copy migration, the migration time remains relatively constant across different message rates, averaging 49.055 seconds. This is expected since the service is fully suspended throughout the migration process, resulting in downtime that equals the migration time. Since no message-based synchronization method is used, the message rate does not impact the migration time.

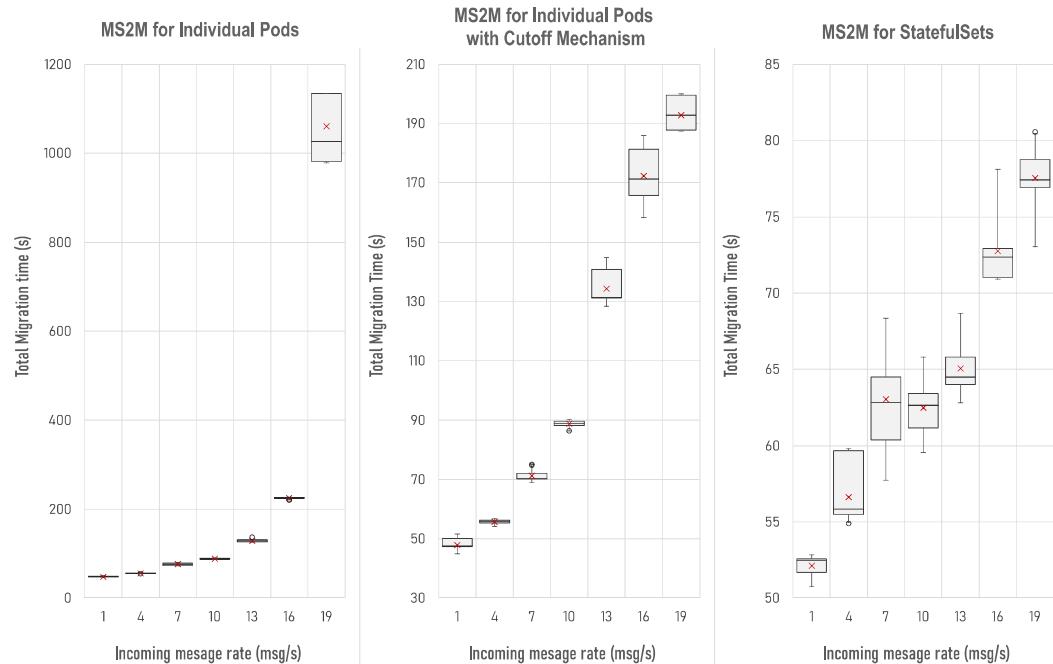


FIGURE 11.5: Distribution of Total Migration Time across test rounds.

Figure 11.5 illustrates the distributions of total migration time for the techniques proposed in this chapter. Consistent with the theoretical analysis of the original MS2M mechanism, the median migration time increases with higher incoming message rates. A new insight, however, is that as the message rate increases, the variance in migration time also grows significantly, making migration durations highly unpredictable. This unpredictability is unacceptable in industrial contexts that require clear service windows. In contrast, the Cutoff mechanism significantly reduced total migration time while also controlling variance more effectively than the original MS2M method. The adapted procedure for StatefulSets is similarly predictable, as its duration is determined by the process of stopping and restoring Pods in Kubernetes rather than depending on incoming message rate.

Figure 11.6 shows the downtime distributions of the proposed techniques. The left-most plot demonstrates that the standard MS2M approach achieves consistently

low downtime (1.36–1.52s) across all message rates by confining service interruption to the brief checkpoint phase, effectively decoupling downtime from message rate and total migration time. While higher loads introduce slight variance, overall downtime remains minimal.

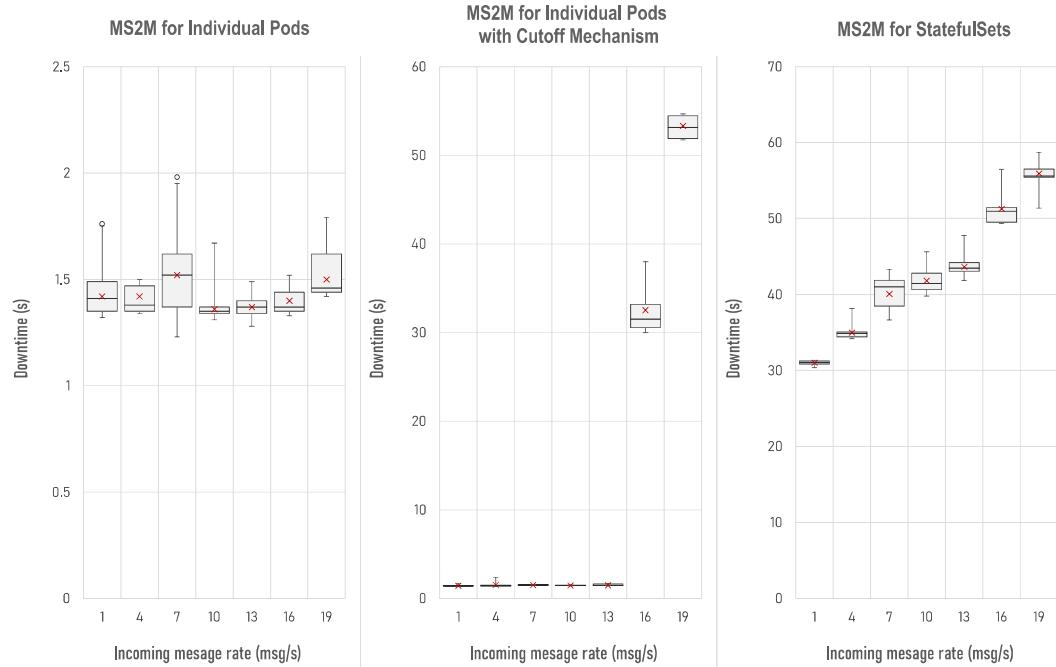


FIGURE 11.6: Distribution of Downtime across test rounds.

In contrast, the Cutoff mechanism demonstrates low downtime at low and moderate message rates, similar to MS2M, but downtime increases significantly at higher message rates (32.53 s at 16 msg/s, 54.46 s at 19 msg/s). This illustrates the trade-off introduced by the cutoff mechanism: to guarantee predictable total migration time, the framework proactively stops the source service, introducing a secondary downtime period proportional to message replay time and thus the message rate.

For StatefulSets, downtime is consistently higher than of MS2M and Cutoff mechanism across most message rates, reflecting Kubernetes' architectural constraint: the source Pod must terminate before the target Pod starts, causing unavailability during restoration and message replay. Thus, downtime is intrinsically linked to migration time. While still improving over cold migrations, this result highlights the unavoidable trade-off between preserving a stable network identity and increased service downtime.

### Comparative Performance Analysis Against Baseline

For MS2M applied to individual Pods, Figure 11.7 show a significant reduction in downtime. From 19.92% in the initial proof-of-concept (Chapter 10) to up to 97.30% in the Kubernetes environment (from 52.01 s to 1.40 s at 16 msg/s). This improvement is due not to changes in the MS2M procedure but to a different baseline: while the standalone prototype presented in previous chapter used a simple local Podman API call via Python script, Kubernetes stop-and-copy involves complex orchestration with

the API server, scheduler, and `kubelet`, dominated by slow operations such as image pulling and volume attachment.

However, the Total Migration Time exposes a critical scalability limitation. The total migration time increases approximately exponentially as the message rate approaches the processing limit. At 16 msg/s, migration time reaches 225.08 seconds, and at 19 msg/s, it reaches 1,060.61 seconds, which is over 21 times longer than the stop-and-copy baseline.

The results show that, while the standard MS2M strategy effectively reduces service downtime, it introduces the severe drawback of unpredictable and potentially unbounded migration time. This limits its suitability to scenarios with predictable, low-to-moderate traffic. For systems operating near processing capacity, MS2M lacks the needed predictability.

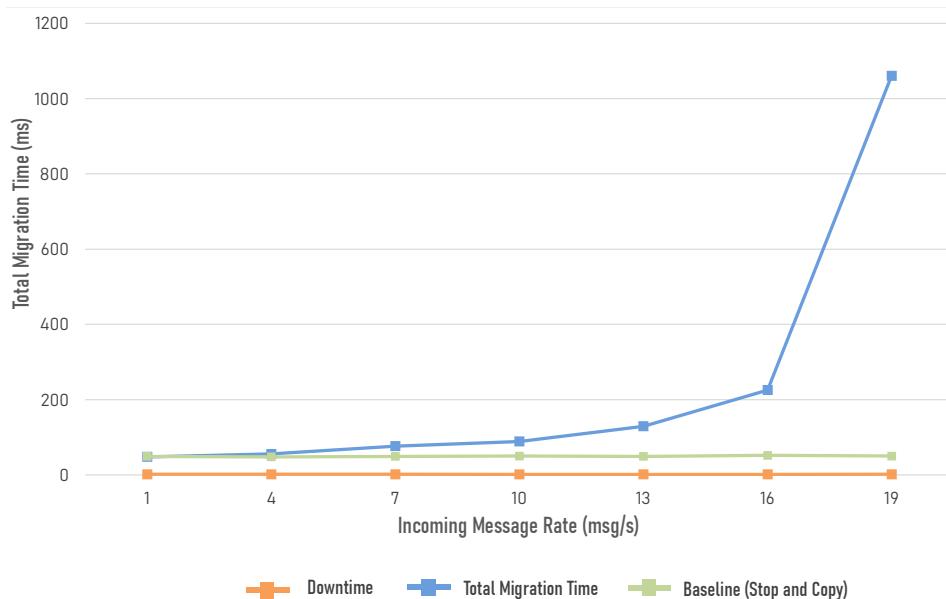


FIGURE 11.7: Comparison of MS2M for Individual Pods against Stop-and-Copy.

For MS2M applied to individual Pods with the Threshold-based Cutoff Mechanism (Figure 11.8), migration time increases more gradually as message rate rises, demonstrating improved handling of high-traffic conditions compared to the standard MS2M approach. At the highest rate of 19 msg/s, the total migration time is approximately 192.8 seconds, an 81.81% reduction compared to original MS2M at the same load. This confirms that the Cutoff mechanism effectively transforms the migration process into a predictable and bounded operation, validating the queuing theory-based model used to guide its optimization.

However, this predictability comes with a clear trade-off. Downtime (orange line) is no longer constant or minimal. While still low at lower message rates (similar to MS2M, as  $t_{\text{replay}} < T_{\text{replay\_max}}$ ), it increases to 55.90 seconds at the highest load (with  $T_{\text{cutoff}} = 120\text{s}$ ). This is because the Cutoff Mechanism introduces a planned secondary downtime by proactively stopping the source service to cap the message replay queue. By adjusting  $T_{\text{cutoff}}$ , system operators can directly tune performance behavior: decreasing  $T_{\text{cutoff}}$  reduces total migration time at the cost of increased downtime, and vice versa.

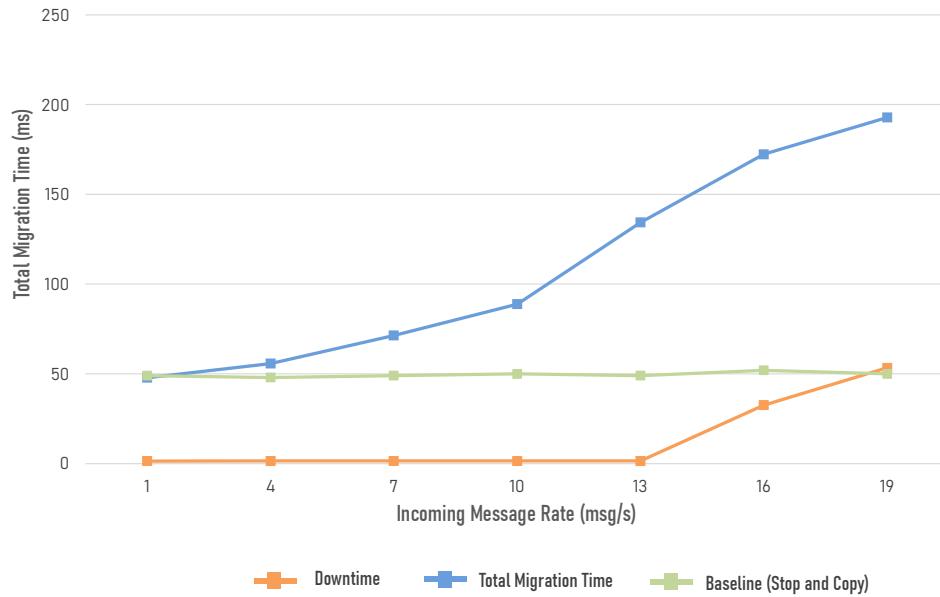


FIGURE 11.8: Comparison of MS2M for Individual Pods with Cutoff mechanism against Stop-and-Copy.

Finally, MS2M for StatefulSet Pods shows a moderate increase in both migration time and downtime as message rates rise (Figure 11.9). The most important observation is the close alignment between Total Migration Time (blue line) and Downtime (orange line). Unlike the live migration of individual Pods, where downtime and migration time were largely decoupled, they move almost in parallel for StatefulSets. This behavior stems from Kubernetes' constraint that the source Pod must be terminated before a new Pod with the same identity can be created on the target node. Therefore, the only difference between total migration time and downtime is the brief checkpoint transfer period, which is independent of the message rate. As a result, the two values appear closely correlated.

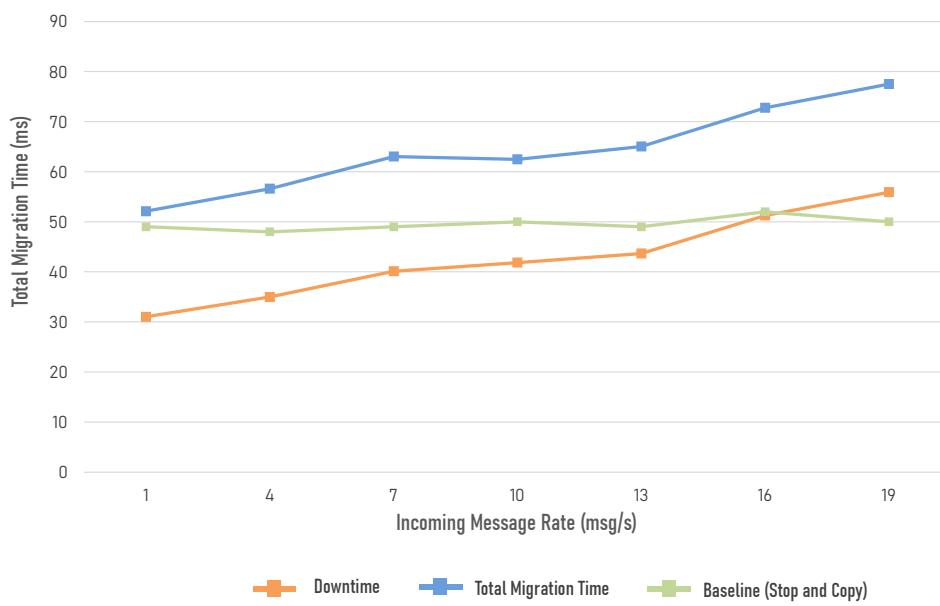


FIGURE 11.9: Comparison of MS2M for StatefulSet Pods against Stop-and-Copy.

At low-to-moderate message rates (1–13msg/s), MS2M for StatefulSets achieved consistently lower downtime than the stop-and-copy baseline, with reductions of 36.69% at 1msg/s and 10.86% at 13msg/s. However, at higher rates (16–19msg/s), a performance crossover occurred, and MS2M downtime surpasses the stop-and-copy baseline. As message rates increase, the replay phase also lengthens.

These results suggest that the adapted procedure offers the greatest benefit under low to moderate traffic conditions, where it provides clear improvements over the traditional stop-and-copy method. In high-traffic scenarios, however, the benefits diminish, suggesting that alternative strategies—such as temporarily scaling up the application or employing blue-green deployments with database replication—may be more effective options.

### Analysis of Migration Bottlenecks Across Phases

To better understand the factors contributing to total migration time across the different strategies, an additional set of evaluations were conducted to analyze how latency is distributed among the migration sub-processes. Each strategy was tested under three representative load conditions: low (4 msg/s, 20% of maximum processing capacity), medium (10 msg/s, 50%), and high (16 msg/s, 80%) traffic levels, relative to the consumer service's maximum rate of 20 msg/s. This setup made it possible to observe how the internal dynamics of each migration approach evolve as the system progresses near saturation. The plots show the time spent in each phase, as explained in Figure 11.1, Figure 11.3, and Figure 11.4. Overall, as the message rate increases from 4 to 16 msg/s, the proportion of time spent on message replay grows significantly across all strategies, particularly for individual Pods without the cutoff mechanism.

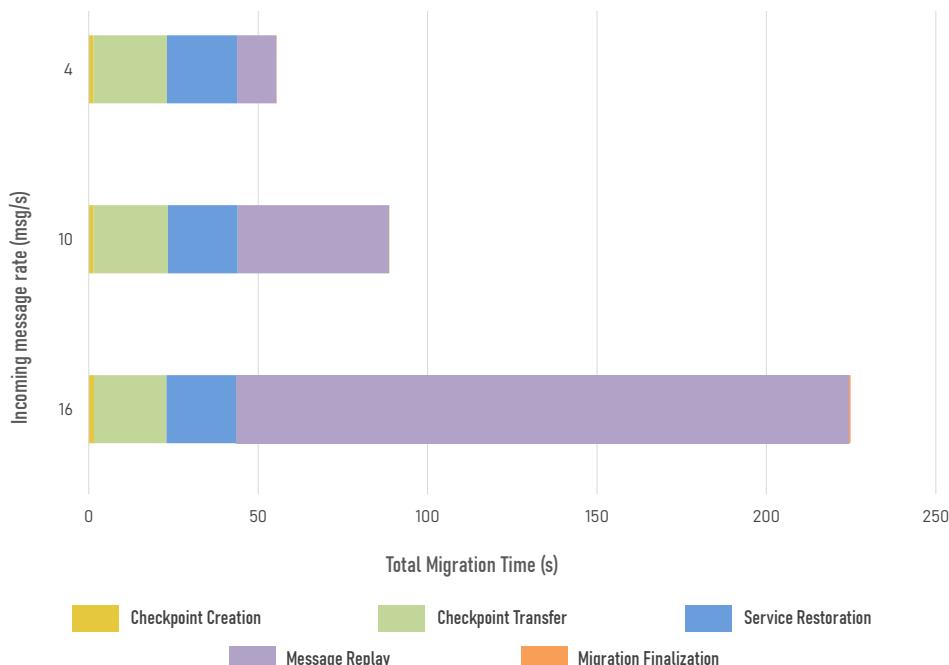


FIGURE 11.10: Latency Analysis of MS2M for Individual Pods.

A clear improvement is observed with the adoption of Forensic Container Checkpointing (FCC). As shown in the previous chapter, checkpoint and restore operations accounted for 88.04% of the total migration time when CRIU was invoked directly. In the Kubernetes integration, the use of FCC, which is an optimized, CRIU-based mechanism, significantly reduces this overhead. At low message rates, the combined checkpoint and restore phases make up only 36.30% of the total migration time when the cutoff mechanism is applied. As message rates increase, this proportion becomes even smaller, since the overall latency is increasingly dominated by the message replay phase.

For MS2M applied to individual Pods, Figure 11.10 shows the message replay phase becomes more dominant as the message rate increase. This again provides empirical evidence that the unbounded message replay problem is the primary scalability bottleneck of the standard MS2M framework. As the incoming message rate approaches the service’s processing capacity, the time required to synchronize the target instance via message replay grows exponentially.

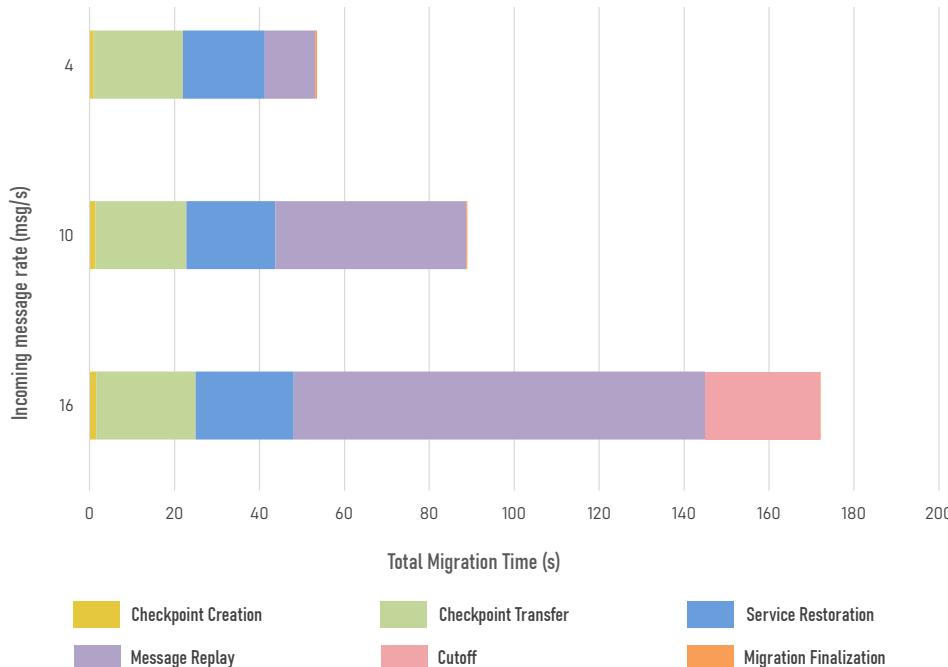


FIGURE 11.11: Latency Analysis of MS2M for Individual Pods with Cutoff Mechanism.

Figure 11.11 shows identical performance at low and moderate message rate, when the cutoff mechanism is not activated. At high message rate, the effect of the optimization is clear. The Message Replay phase (purple bar), which previously took over 180.74 seconds, is now capped at approximately 96.84 seconds. The remaining time is now accounted for by the new Cutoff phase (pink bar), which represents the secondary downtime period (27.05s). The two, when combined, are still 31.45% faster than original MS2M’s replay time.

Finally, Figure 11.12 shows a completely different performance profile compared to the individual Pod migrations, which is dominated by orchestration overhead rather than message replay. At all load levels (low, medium, and high) the Service Restoration phase (blue bar) consistently consumes a large portion of the total migration time.

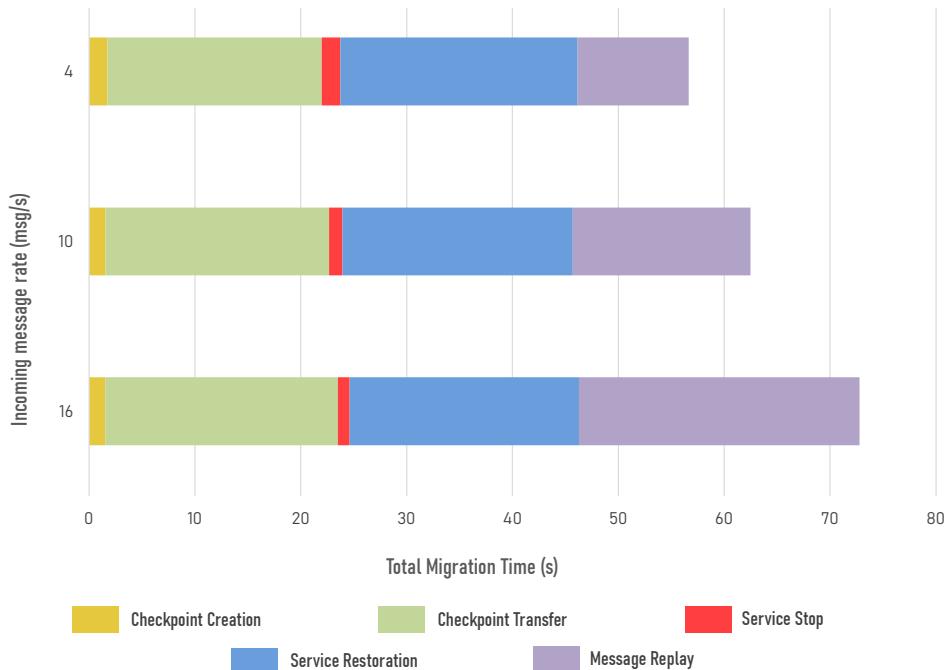


FIGURE 11.12: Latency Analysis of MS2M for StatefulSet Pods.

This phase represents the time Kubernetes takes to fully terminate the original Pod, release its identity, and then schedule, create, and start the new Pod on the target node. This is a complex orchestration process with inherent latencies. The Message Replay phase (purple bar), while still growing with the message rate, is no longer the prominent bottleneck it was in the standard MS2M strategy. Even at a high load of 16 msg/s, it only accounts for approximately 36.4% of the total migration time, a stark contrast to the over 80% seen previously. This chart demonstrates that for StatefulSet migration, the primary performance bottleneck is not the MS2M procedure itself, but the inherent operational overhead of the Kubernetes control plane when managing identity-bound workloads.

## 11.6 Chapter Summary

This chapter addressed the key operational challenges of the original MS2M framework. The issue of *unbounded message replay*, which is responsible for unpredictable migration times under high load, was resolved through the introduction of the *Threshold-Based Cutoff Mechanism*. In addition, the framework was integrated into Kubernetes, utilizing its native *Forensic Container Checkpointing* (FCC) feature to reduce the latency previously associated with manual checkpoint and restore operations using CRIU.

The analysis of standard MS2M for individual Pods confirmed that, although the framework can reduce service downtime, the total migration time increases sharply as message rates grow. This unbounded replay behavior makes the approach unsuitable for high-traffic industrial workloads, highlighting the need for a mechanism that can control the state synchronization phase more effectively.

One of those mechanisms is the Cutoff mechanism, which effectively transforms the migration into a predictable process. It enables operators to make informed, data-driven trade-offs between availability and performance:

- *Prioritize Availability*: Apply a higher cutoff threshold  $T_{\text{cutoff}}$  to minimize downtime at the cost of a longer total migration time.
- *Prioritize Predictability/Speed*: Apply a lower cutoff threshold  $T_{\text{cutoff}}$  to ensure a fast migration completion, accepting a longer period of controlled downtime.

The evaluation of `StatefulSet` migration further exposed an inherent limitation in Kubernetes itself: downtime remains tightly coupled with orchestration latency, as Pods with the same identities cannot coexist during migration. While the current integration improves performance under low to moderate traffic, its benefits diminish at higher message rates. Addressing this constraint would likely require a deeper integration effort, such as developing a custom Kubernetes controller specifically optimized for stateful microservices.

Finally, the analysis presented in this chapter highlights the need to ensure application-level consistency without resorting to architecturally invasive methods. The evaluated sandboxed replay model, though functional, remains fragile and imposes a high implementation burden. Future research should therefore focus on designing an *idempotency-aware migration framework*, where services are inherently capable of consistent state reconstruction without specialized replay logic.



## **Part VI**

# **Managing Scale: Resource Coordination across the Continuum**



# 12 Service Affinity and Centralized Resource Management

## 12.1 Introduction

Earlier chapters of this dissertation established the foundational principles for engineering modern industrial systems. Part III introduced the CRACI reference architecture, designed for the Industrial Compute Continuum. Part IV presented evaluation results focusing on the development and deployment aspects of the proposed architecture. Later, in Part V, the focus shifted to one of the key operational challenges, which is service mobility, through the development of the MS2M framework and its integration into Kubernetes. The results demonstrated that live migration via message replaying is technically feasible, allowing stateful industrial services to be migrated with minimal downtime.

However, the capability to migrate services raises a more strategic question: *when should a service be moved, and to where?* Although MS2M enables efficient migrations, it does not include the decision logic required for optimal service placement.

This chapter addresses this decision-making gap by introducing the concept of *Service Affinity* (SA), a multi-dimensional metric that captures the relationships, data flows, privacy constraints, and functional dependencies among industrial microservices. To validate this concept, the Service Affinity Graph-based Approach (SAGA) is proposed. SAGA collects data from the system's messaging infrastructure and orchestration layer, applies graph theory principles to represent the microservice cluster, and reformulates the placement task as a minimum-weight  $k$ -cut optimization problem.

The contributions of this chapter build upon previously published work [506], are twofold:

- *A Conceptual Framework for Affinity-Driven Decision-Making:* Service Affinity is defined and formalized as the core decision model guiding the migration mechanisms developed in Part V. Its impact on service latency is evaluated quantitatively.
- *Implementation and Evaluation of a Prototype:* The design and evaluation of SAGA are presented as the first implementation of this decision-making logic. Deployed in a standard Kubernetes environment, the prototype demonstrates the feasibility of an affinity-aware orchestrator.

The remainder of this chapter is organized as follows. Section 12.2 reviews related work on service placement methods. The algorithms for SA calculation, as well as the  $k$ -way extension of the Kernighan-Lin algorithm are explained in Section 12.3. Section 12.4 presents the prototype implementation of SAGA within a Kubernetes environment, while Section 12.5 presents both quantitative and qualitative evaluation results. Lastly, Section 12.6 summarizes the key findings and outlines potential directions for future research.

## 12.2 Related Work

Service placement within the Compute Continuum presents a complex optimization challenge. As presented in the Taxonomy, service placement involves determining optimal hosting locations to satisfy multiple objectives such as minimizing latency, balancing workloads, enhancing cost efficiency, and ensuring high availability [507]. The distributed nature of these systems and the underlying NP-hard combinatorial optimization further increase the complexity of this task. While traditional deterministic algorithms can address specific cases, they often lack the scalability necessary for these environments, leading to a growing interest in dynamic, adaptive methodologies. This section reviews representative works in the field, structured by methodological approach.

*Exact methods* for solving service allocation problems typically involve approaches that guarantee finding the optimal solution, albeit often at the cost of high computational complexity. For example, Liu and Fan [508] address multi-cloudlet resource allocation using a two-stage Mixed Integer Linear Programming (MILP) strategy. The first stage selects the optimal cloudlet by evaluating latency and reward, while the second allocates resources within the chosen cloudlet to optimize overall system reward and resource usage. Their MILP formulations focus on maximizing mean reward and minimizing latency, with simulation results indicating enhanced system performance.

In contrast, *heuristic methods* trade optimality for scalability, providing practical and computationally efficient solutions. For example, Fan et al. [509] propose the *Iterative Greedy-and-Search-based (IGS)* algorithm to optimize resource allocation and pricing in cloud/edge environments for mobile blockchain applications. Framing the problem as a Stackelberg game between Cloud/Edge Service Providers (CESP) and users, they present a mixed-integer programming formulation and prove equilibrium existence. The algorithm decouples the problem into resource allocation under given pricing and pricing under the resulting allocation, utilizing a greedy-and-search strategy and golden section search, respectively. Simulation results demonstrate that IGS effectively increases CESP and user revenues while reducing task execution delays, particularly as user revenue parameters ( $\alpha_i$ ) and server computing power increase. Another example is the work by Soumplis et al. [317], who use a Mixed Integer Linear Programming (MILP) model to minimize latency and cost. To handle MILP complexity in large systems, they introduce a greedy best-fit heuristic and a multi-agent rollout mechanism inspired by Approximate Dynamic Programming and Reinforcement Learning. This hybrid approach achieves near-optimal results (within

4% of MILP) with improved scalability and execution time, illustrating the trade-off between optimality and efficiency, typically found in heuristic methods.

Complementing these approaches, *mathematical models* provide structured frameworks for balancing multiple objectives in service placement. Bento et al. [510] propose a *bi-objective optimization model* to maximize service availability while minimizing costs under infrastructure and SLA constraints. Using a Pareto front approach, the model identifies optimal configurations, with case studies demonstrating improved availability and cost efficiency, reflecting a shift from reactive, localized optimization to proactive, global strategies. Wei et al. [511] propose CSRPEs, a stochastic algorithm integrating horizontal and vertical resource sharing to maximize revenue under dynamic demand and nonlinear pricing. Combining fast approximation methods with Particle Swarm Optimization, Simulated Annealing, and Differential Evolution, the method iteratively refines a base solution, achieving more than 30% revenue gains in resource-constrained environments.

*Graph-theoretical* approaches provide specialized techniques for handling the inherent network structures in service placement problems. Devi and Murugaboopathi [512] propose the *Cloud Load Balancer* (CLB) algorithm, incorporating *Cluster Data Center Clustering* (CDCC) and *Client Cluster Assignment* (CCA) to improve clustering and load distribution in multi-cloud environments. By selecting initial cluster centers based on node degree and achieving balanced client allocation, CLB outperforms k-means and k-spanning tree algorithms in reducing clustering time and enhancing service delivery performance.

*Machine learning-based* approaches are also gaining traction. Afachao et al. [513] propose BAMPP, a Reinforcement Learning (RL) algorithm for dynamic microservice deployment in CC. Built on Advantage Actor-Critic with enhanced stability via state-value loss ratios, BAMPP reduces network usage by up to 8%, energy consumption by up to 16.5%, and migration delays. Evaluations on EdgeSimPy show that BAMPP outperforms ILP, PPO, and A2C baselines, demonstrating RL's potential in optimizing service placement under latency and resource constraints.

There are also other works focusing on actual techniques for service migration within cloud-native environments, which not only leads to more efficient use of resources [514] but also helps overcome challenges like network congestion or server downtime, enabling uninterrupted service availability. The MS2M mechanism proposed and evaluated in the previous part [491, 505] is an example. This approach reduces downtime and allow service availability during migration. Another worth mentioning work is *Sledge* [515], which introduces a *Lightweight Container Registry* (LCR) mechanism for end-to-end image migration and employs *Dynamic Context Loading* (DCL) for management context migration. Sledge's approach significantly reduces both downtime and total migration time compared to previously proposed solutions.

Although these approaches provide valuable contributions, this literature review indicates a common limitation: they are primarily resource-centric. Most existing approaches focus on optimizing infrastructure-level metrics, such as CPU, memory, network routes, and cost. However, they lack a deep understanding of the application's internal logic. In other words, they optimize how resources are consumed, but not why services interact the way they do. This creates a critical gap: an orchestrator

might perfectly balance CPU load across two edge nodes but inadvertently separate two services that have a high-frequency, low-latency communication requirement, which degrades overall application performance.

The Service Affinity Graph-based Approach (SAGA) framework addresses this gap by introducing *Service Affinity*, a multi-dimensional metric that models contextual service relationships, including data flows, functional dependencies, and operational patterns. By using this application-aware metric, SAGA constructs a graph representation of the system to guide placement decisions that align with the application's internal logic, achieving orchestration that respects both infrastructure constraints and functional requirements.

## 12.3 The Service Affinity Concept

At the core of the SAGA framework is the concept of Service Affinity (SA), a multi-dimensional metric designed to quantify the *closeness* or *relatedness* of microservices based on their operational and functional characteristics. Unlike traditional resource-centric metrics (CPU, memory), SA provides a deeper, application-aware understanding of the system, enabling more intelligent placement decisions. This section formalizes the SA concept, details its key components, and explains how it is used to construct a weighted graph model of the microservice cluster, effectively translating the placement task into a graph partitioning problem.

Let  $G(V, E)$  be an undirected graph, where each vertex  $v \in V$  represents a microservice instance within a CC environment, and each edge  $e \in E$  represents a connection between two services, existing only when data is exchanged between them. Let  $k$  denote the number of hosts in the cluster. For any pair of nodes  $u, v \in V$ , the affinity value  $a_{u,v}(\Delta t)$  during the time window  $\Delta t$  is calculated as:

$$a_{u,v}(\Delta t) = \sum_{i \in \{d,p,c,f,o\}} \alpha_i \cdot a_i(u, v, \Delta t) \quad (12.1)$$

where:

- $a_{u,v}(\Delta t)$  represents the overall affinity value between  $u$  and  $v$  during the time window  $\Delta t$ .
- $a_i(u, v, \Delta t)$  represents the specific affinity value between  $u$  and  $v$  for type  $i$ , with  $i$  being data ( $d$ ), coupling ( $c$ ), functional ( $f$ ), operational ( $o$ ), and security & privacy ( $p$ ) during the time window  $\Delta t$ .
- $\alpha_i$  is the weight assigned to each affinity type. These values are adjustable according to the specific environment, allowing fine-grained control over the overall affinity calculation.

The dimensions of this affinity are further defined and explained in the following sections.

### 12.3.1 Data Affinity ( $d$ )

Data Affinity measures the volume of data exchanged between two microservices, calculated as the normalized ratio of bytes exchanged between them to the total data

exchanged across the application:

$$d_{u,v}(\Delta t) = \frac{b_{u,v}(\Delta t)}{B(\Delta t)}$$

This metric is important for managing network bandwidth on the factory floor. It captures high-volume data exchange such as a video streams (for example, the control service in MAIA use case, mentioned in Chapter 7) or high resolution images. A high Data Affinity value indicates a strong relationship data dependency services, encouraging their co-location to reduce network congestion and data transfer overhead.

### 12.3.2 Coupling Affinity (*c*)

Coupling Affinity measures the frequency of interaction between two microservices, regardless of the amount of data exchanged. It is calculated as the normalized count of exchanged messages or API calls between them.

$$c_{u,v}(\Delta t) = \frac{m_{u,v}(\Delta t)}{M(\Delta t)}$$

While Data Affinity captures large data exchanges, Coupling Affinity focuses on time-sensitive, high-frequency interactions. For instance, a real-time control loop between a PLC controller and a supervisory service may exchange thousands of small heartbeat messages per minute. Each of those message is lightweight in size but collectively significant in terms of latency. A high Coupling Affinity value indicates a strong functional or operational dependency, encouraging their co-location to minimize latency and maintain system responsiveness.

### 12.3.3 Functional Affinity (*f*)

Functional Affinity reflects how closely related two services are in terms of their roles within a business process. It is expressed as a weighted similarity score, typically derived from an architectural knowledge graph or defined by domain experts.

$$\text{Sim}_{func}(u, v) \in [0, 1]$$

This metric provides detailed insight into how services relate within industrial workflows. For example, in a predictive maintenance pipeline, the `vibration_data_ingestor` and `anomaly_prediction_model` services have high functional affinity because they operate in a tightly linked sequence. In contrast, the `maintenance_ticket_creator` is less directly related. Such insight enables the orchestrator to co-locate closely related components, improving overall system coherence.

### 12.3.4 Operational Affinity (*o*)

Operational Affinity is a hybrid metric that evaluates how services relate to hardware resources. It calculates a single score that balances a service's static hardware

requirements (*attractive force*) with its dynamic runtime contention for those resources (*repulsive force*). This affinity dimension builds on insights from earlier evaluations in this dissertation, particularly from unikernels vs containers and microservice implementations comparisons. This affinity score is defined as:

$$o_{u,v} = (1 - \gamma_o) \cdot Sim_{op}(u, v) - \gamma_o \cdot Cont(u, v)$$

where  $\gamma_o$  is the *Operational Balance Factor* ( $0 \leq \gamma_o \leq 1$ ), a configurable parameter tuning the balance between static similarity and dynamic contention.

- Operational Similarity ( $Sim_{op}$ ): Measures how similar two services are in terms of their declared hardware requirements. Each service is represented as a profile vector (e.g., [GPU: Required, RAM: High]), with binary or normalized values for each resource type. Cosine similarity is then applied to these vectors, producing a score between 0 (completely dissimilar) and 1 (identical needs). A high score indicates that both services would benefit from being placed on the same specialized node.
- Resource Contention ( $Cont$ ): Measures how the degree to which two services compete for limited runtime resources. Using live telemetry data, CPU contention is modeled as  $Cont_{cpu}(u, v) = Usage_{cpu}(u) \times Usage_{cpu}(v)$ , where usage is normalized between 0 and 1. This ensures contention is only high when both services are heavily utilizing the same resource. The approach extends to other dimensions—such as memory or I/O—via a weighted sum: such as  $Cont(u, v) = \beta_{cpu} \cdot Cont_{cpu}(u, v) + \beta_{mem} \cdot Cont_{mem}(u, v) + \dots$ , allowing the orchestrator to prioritize which resource contentions to avoid.

This unified Operational Affinity is particularly valuable for managing heterogeneous industrial environments. The *attractive force* component ensures that services requiring specialized hardware are co-located on suitable nodes, maximizing utilization of scarce resources. Meanwhile, the *repulsive force* component mitigates the *noisy neighbor problem* [516], where resource-intensive workloads interfere with one another. For example, if two CPU-heavy simulation services were deployed on the same node, their high contention would generate a strong repulsive signal, prompting the orchestrator to separate them and prevent delays.

### 12.3.5 Security & Privacy Affinity ( $p$ )

Security & Privacy Affinity measures how closely aligned two services are in terms of their security and privacy requirements. It encourages co-location of services with similar sensitivity levels or compliance domains to maintain policy enforcement.

$$p_{u,v} = \begin{cases} 0 & \text{if group of } u = \text{group of } v \\ -P_{violation} & \text{if group of } u \neq \text{group of } v \end{cases}$$

where:

- *group of  $x$*  refers to the security group that the tag of service  $x$  belongs to.

- $P_{violation}$  is a large negative penalty that acts as a hard constraint, effectively discouraging or preventing cross-group placements.

### 12.3.6 Weighted-graph representation of the microservices cluster

The normalized edge weight  $w_{u,v}(\Delta t)$  for any pair of nodes  $u, v \in V$  is calculated by normalizing the affinity value  $a_{u,v}(\Delta t)$  based on the minimum and maximum affinity values observed across all edges in  $E$  during the time window  $\Delta t$ :

$$w_{u,v}(\Delta t) = \frac{a_{u,v}(\Delta t) - \min_{(u,v) \in E} a_{u,v}(\Delta t)}{\max_{(u,v) \in E} a_{u,v}(\Delta t) - \min_{(u,v) \in E} a_{u,v}(\Delta t)} \quad (12.2)$$

## 12.4 The Partitioning Algorithm

In this section, the concept and design of the modified  $k$ -way Kernighan–Lin algorithm are introduced, developed to optimize microservice deployment across the CC. The main objective is to partition the graph into  $k$  balanced subsets, where each subset represents a cluster of microservices with high internal affinity and minimal inter-cluster affinity. The detailed algorithmic procedure is presented below.

Using formula (12.1) and (12.2), an undirected, weighted graph  $G(V, E)$ , is constructed to represent the microservice cluster. The objective is to cluster the vertices into  $k$  distinct subsets, minimizing the total edge weight between them, which forms a minimum-weight  $k$ -cut problem. This problem is NP-hard and can be approached through different algorithmic strategies depending on the graph's size and complexity. For smaller graphs, exact optimization techniques such as Integer Linear Programming (ILP) are feasible. For medium-sized graphs, approximation methods including the Greedy or Kernighan–Lin algorithms and spectral partitioning are more practical. For large-scale graphs, heuristic or stochastic approaches such as simulated annealing, genetic algorithms, or randomized methods are often adopted to achieve near-optimal results within reasonable computational limits.

In this work, the Kernighan–Lin algorithm was selected for its strong balance between solution quality and computational efficiency. Although originally developed for two-way graph partitioning, it was adapted in this dissertation to support  $k$ -way clustering, allowing the formation of  $k$  distinct groups of microservices. The adaptation follows a straightforward *recursive bipartitioning* strategy: the algorithm first divides the complete set of nodes into two optimal subsets, and then recursively applies the same partitioning process to the largest subset until the desired number of clusters is obtained. The modified pseudocode is provided in Algorithm 1.

## 12.5 The SAGA Framework

Following the introduction of the Service Affinity concept, which transform the service placement into a graph partitioning problem, this section presents the initial centralized implementation of the  $k$ -way Kernighan–Lin algorithm and its integration with Kubernetes.

**Algorithm 1**  $k$ -way Kernighan–Lin Algorithm

---

```

1: procedure KERNIGHAN-LIN-K( $G(V, E)$ ,  $k$ )
2:   Initialize a list  $S$  containing all vertices
3:   while length of  $S < k$  do
4:     Identify the largest subset  $L$  in  $S$ 
5:     Partition vertices in  $L$  into balanced sets  $A$  and  $B$ 
6:     repeat
7:       Compute  $D$  values for all  $a \in A$  and  $b \in B$ 
8:       Let  $gv$ ,  $av$ , and  $bv$  be empty lists
9:       for  $n := 1$  to  $|L|/2$  do
10:        Find  $a \in A$  and  $b \in B$  to maximize  $g = D[a] + D[b] - 2 \times c(a, b)$ 
11:        Remove  $a$  and  $b$  from further consideration in this pass
12:        Add  $g$  to  $gv$ ,  $a$  to  $av$ , and  $b$  to  $bv$ 
13:        Update  $D$  values for the elements of  $A = A \setminus \{a\}$  and  $B = B \setminus \{b\}$ 
14:      end for
15:      Find  $t$  which maximizes  $g_{\max}$ , the sum of  $gv[1], \dots, gv[t]$ 
16:      if  $g_{\max} > 0$  then
17:        Exchange  $av[1], av[2], \dots, av[t]$  with  $bv[1], bv[2], \dots, bv[t]$ 
18:      end if
19:      until  $g_{\max} \leq 0$ 
20:      Replace  $L$  in  $S$  with two new subsets  $A$  and  $B$ 
21:    end while
22:    return  $S$                                  $\triangleright$  List of  $k$  subsets
23: end procedure

```

---

**12.5.1 Architecture Overview**

The Service Affinity Graph-based Approach (SAGA) is a centralized integration into Kubernetes. This prototype consists of three services as follows:

1. *SAGA Telemetry Collector (STC)*: This service continuously collects runtime data from cloud infrastructures, including communication fabrics and cloud orchestrators. It primarily tracks and analyzes message flows among microservices, capturing metadata such as the sender, receiver, message count, size, and frequency. Additionally, STC collects other operational metrics through cloud orchestrator APIs.
2. *SAGA Affinity-based clusterer (SAC)*: Based on the data collected by STC, SAC groups microservices according to their calculated affinities over the selected time window  $\Delta t$ . It reconstructs the graph representation  $G$ , computes the pairwise affinity values  $a$ , and implements the  $k$ -way Kernighan–Lin algorithm to partition the graph into  $k$  subsets that minimize cross-cluster affinity.
3. *SAGA Migration Manager (SMM)*: SMM enforces the placement decisions determined by SAC by orchestrating microservice migrations. It orchestrates the entire migration process, serving the role of Migration Manager as defined in MS2M procedure. Once the service reallocation is complete, it signals STC to reset its database and start collecting new metric data for the subsequent time window.

While the STC runs continuously, both SAC and SMM can operate on a set schedule or be triggered on demand, such as in response to increases in system load. Figure 12.1 illustrates the high-level view of these three services in the Kubernetes cluster.

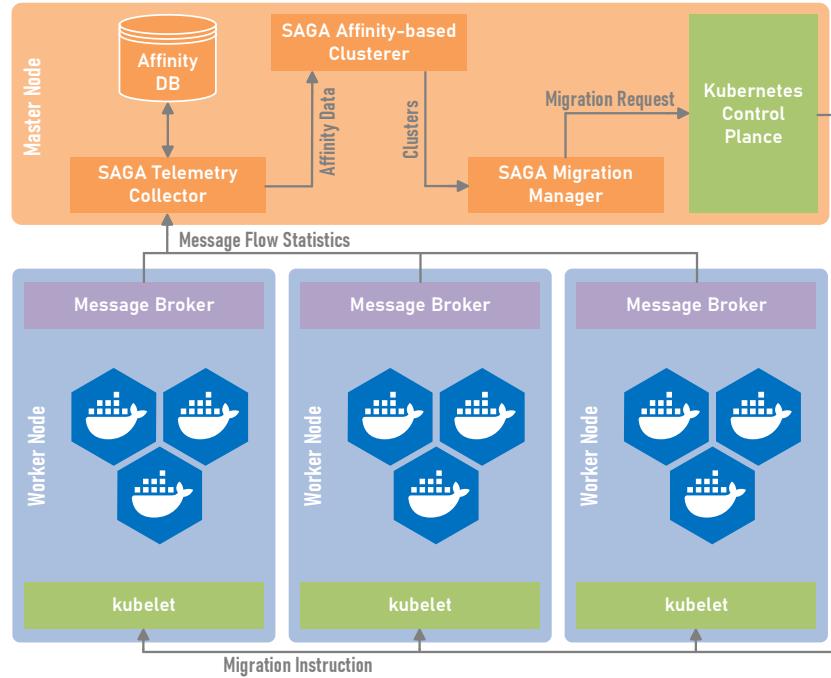


FIGURE 12.1: Overview of SAGA Framework prototype integrated within Kubernetes cluster.

## 12.6 Evaluation and Discussion

The implemented prototype was first analyzed theoretically to determine the time complexity of the  $k$ -way Kernighan–Lin algorithm. This was complemented by a runtime evaluation of the prototype to assess its performance. The results of both evaluations will be presented in this section.

### 12.6.1 Algorithm Complexity Evaluation

The time complexity of the original Kernighan–Lin bipartitioning run on a graph of size  $n$  is  $O(n^2 \log n)$ . The  $k$ -way partitioning approach proposed in Algorithm 1 extends this by applying the algorithm recursively. The proposed  $k$ -way partitioning approach, as outlined in Algorithm 1, employs a recursive strategy. It begins by splitting the entire graph of  $n$  vertices, an operation with a complexity of  $O(n^2 \log n)$ . Subsequent iterations operate on progressively smaller subgraphs (e.g., of size  $n/2, n/4$ , etc.), each causing a significantly lower computational cost. As the cost of the first partition on the full graph dominates the total cost, the overall estimated time complexity for this recursive  $k$ -way partitioning method remains  $O(n^2 \log n)$ .

### 12.6.2 Algorithm Runtime Analysis

As the theoretical time complexity analysis suggests, the runtime performance of the algorithm depends on the number of microservices  $n$  and the number of nodes  $k$ . Therefore, the evaluation was set up where the number of microservices varies from 100 to 500, and the number of nodes ranges from 10 to 50. The algorithm is

implemented in Java, with the source code publicly available<sup>1</sup>. All tests were conducted on a dedicated server equipped with an Intel® Core™ i7-14700K (20 cores, 5.6 GHz max turbo) and 64 GB DDR4 RAM.

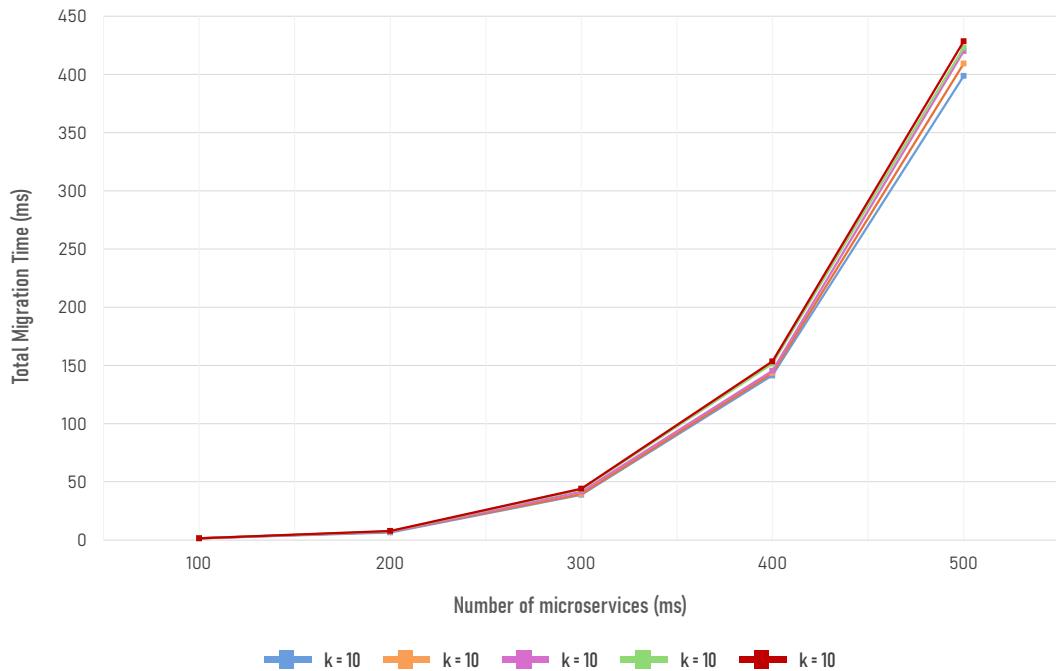


FIGURE 12.2: Extended  $k$ -way Kernighan–Lin algorithm performance analysis.

The runtime performance analysis of the  $k$ -way Kernighan–Lin algorithm, shown in Figure 12.2, demonstrates a strong dependence on the number of microservices ( $n$ ), with a comparatively smaller influence from the cluster count ( $k$ ). For instance, the results indicate that in a smaller application with 100 microservices, the algorithm’s runtime ranges from 343.32 ms ( $k = 10$ ) to 474.83 ms ( $k = 50$ ). In contrast, for a larger application with 500 microservices, the runtime increases significantly, from 398.05 s ( $k = 10$ ) to 428.00 s ( $k = 50$ ). Across different values of  $k$ , there was only a minor runtime variation recorded. This behavior can be explained by the recursive nature of the algorithm, where each subsequent run operates on a smaller subset of the graph, particularly in later iterations. Overall, the findings suggest that the size of the microservice architecture is the dominant factor limiting the scalability of the algorithm.

### 12.6.3 Prototype Evaluation

The SAGA framework was evaluated using Microsoft’s open-source .NET Core eShop application [517], demonstrating basic microservice architecture and employing RabbitMQ for messaging [518]. The whole application was deployed on a Kubernetes cluster in the Google Kubernetes Engine (GKE) [519]. The cluster consisted of one master and three worker nodes distributed across multiple zones to ensure high availability. Each node was provisioned with 8 vCPUs and 32 GB of memory on Google

<sup>1</sup>The source code is publicly available at: <https://github.com/haidinhuan/k-way-Kernighan-Lin-algorithm>

Cloud Platform [520]. All SAGA components were implemented in Java using the Spring Framework [521].

The evaluation focused on measuring service latency for typical user operations such as order placement, payment processing, and item retrieval. A dataset of 200 requests was used to analyze end-to-end latency, defined as the time between a request being sent and the corresponding response received. Latency measurements were collected, similar to previous evaluations, using *distributed tracing* techniques [522], allowing for detailed tracking and analysis of request handling within the system.

Immediate evaluation results are shown in Figure 12.3. Before migration, the data showed a broader range of latency values, with minimum and maximum latencies at 24.81 ms and 52.08 ms, respectively. The mean latency was 31.80 ms, with a standard deviation of 6.50 ms, indicating a moderate variation. The interquartile range was from 28.20 ms to 33.54 ms, suggesting some instances of higher latency were recorded.

After migration, there was a significant improvement in latency performance. The highest recorded latency was significantly lower at 33.91 ms, and the mean latency reduced to 24.36 ms with a smaller standard deviation of 4.30 ms. The interquartile range also narrowed to 20.98 ms to 27.67 ms.

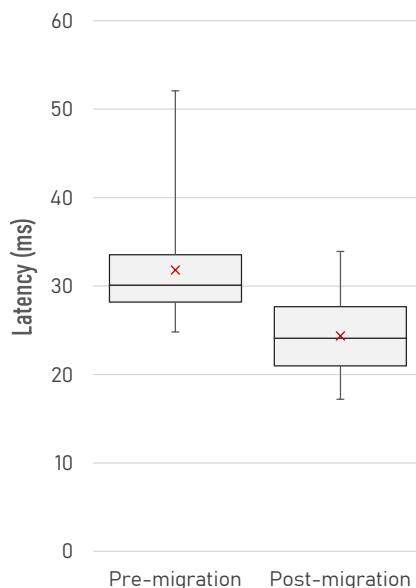


FIGURE 12.3: Box plots of the latency performance before and after migration.

As shown in Figure 12.3, the pre-migration box plot displays a wider spread and longer whiskers, indicating higher latency variability. Post-migration, the box becomes more compact, with shorter whiskers, indicating more stable and predictable latency behavior.

Overall, the results show a clear improvement after migration, with a 23.4% reduction in mean latency. The latency performance was also more predictable after migration. This demonstrates that the proposed affinity-driven placement mechanism not only lowers average latency but also enhances overall system stability and responsiveness.

## Chapter Summary

The evolution toward cloud-native and distributed architectures in industrial systems requires new strategies to maintain performance and service quality. This chapter addressed that challenge through the introduction of the SAGA framework, which bases service placement on the application-aware concept of Service Affinity (SA). By representing the microservice cluster as a weighted graph and transforming placement into a minimum-weight  $k$ -cut problem, SAGA offers an alternative approach to resource-driven optimization.

The implementation of SAGA within a standard Kubernetes environment, which includes the SAGA Telemetry Collector, SAGA Affinity-based Clusterer, and SAGA Migration Manager, served as a concrete validation of the concept. The evaluation has two main results.

First, the prototype achieved a 23.4% reduction in average request latency for a representative microservice application. This outcome confirms that utilizing a holistic, affinity-based understanding of application behavior enables more efficient communication patterns and improved overall system responsiveness. By incorporating application-level metadata into orchestration decisions, SAGA provides a more complete foundation for deployment optimization than conventional resource-centric methods.

The second result highlights a key limitation. The runtime analysis of the modified Kernighan–Lin algorithm revealed a strong dependence on the number of microservices. This finding suggests scalability concerns, as the centralized design introduces both a performance bottleneck and a single point of failure. Although SAGA successfully demonstrates the feasibility of the Service Affinity concept, its centralized architecture remains unsuitable for large-scale, highly dynamic industrial systems.

From these findings, clear directions for future development can be drawn. While the partitioning algorithm itself can be optimized further, the main limitation is in SAGA's centralized architecture. Given that the CC is inherently a distributed computing model, relying on a single centralized entity for optimization contradicts its fundamental principles. Moreover, as discussed in the taxonomy earlier in this dissertation, decentralization has been identified as one of the key driving trends for research in CC.

Motivated by these results, the next chapter introduces DREAMS, a fully decentralized framework that implements the Service Affinity in an autonomous, collaborative, and consensus-driven decision-making architecture across the Industrial Compute Continuum.

# 13 Decentralized Resource Management across the Continuum

## 13.1 Introduction

The previous chapter introduced the SAGA framework and presented the Service Affinity concept as a multi-dimensional metric used for optimizing microservice placement. Through an implemented prototype, it was shown that an affinity-aware approach can significantly reduce service latency. However, its centralized design becomes a bottleneck in the large-scale ecosystems envisioned by Industry 4.0. A single orchestrator cannot manage the needs from thousands of services across a global supply chain. Moreover, the dependence on global knowledge and the use of algorithms with limited scalability creates a single point of failure.

To overcome these limitations, this chapter presents DREAMS (Decentralized Resource and Affinity Management System), a framework that extends the Service Affinity concept into a decentralized, fault-tolerant, and scalable architecture. Instead of relying on a central orchestrator, DREAMS adopts a *multi-agent model*, where each compute domain is governed by an autonomous *Local Domain Manager* (LDM). These LDMs collect local metrics, evaluate placement improvements based on service affinity, and collaborate through a consensus-driven protocol to make decentralized placement decisions. This decentralized approach not only preserves domain autonomy and data privacy compliance, but also enables collaborative optimization through coordinated local interactions.

The contributions of this chapter are therefore threefold:

1. A decentralized decision-making framework for collaborative service management across the continuum.
2. The design and implementation of a reusable Local Domain Manager (LDM), which is capable of autonomous operation and coordination through consensus mechanisms.
3. A thorough evaluation demonstrating the feasibility and scalability of the proposed architecture, including metrics on LDM registration convergence and migration coordination latency under increasing system size.

The remainder of this chapter is structured as follows. Section 13.2 reviews related work, emphasizing key contributions and their limitations. Section 13.3 presents the design of the LDM. Section 13.4 details the technical implementation of the

underlying decentralized decision-making logic. Section 13.5 presents the evaluation setup and discusses the results. Finally, Section 13.6 concludes the chapter and outlines directions for future research.

## 13.2 Related Work

The previous chapter has reviewed various research addressing service placement optimization. While valuable, these approaches share a common limitation in their centralized model. However, the very nature of the CC, as a unified infrastructure spanning from centralized Cloud to Edge and IoT devices, implies a massive scale and a high degree of distribution. When combined with the proliferation of cloud-native services as proposed in the CRACI architecture, centralized solutions face significant limitations. These include single points of failure, limited scalability, and privacy concerns in multi-domain environments. Therefore, recent research increasingly favors decentralized approaches for their scalability, resilience, and local autonomy.

To achieve decentralization, several strategies have been proposed in the literature. One of them is to *decompose global mathematical models*. A well-known example is the Alternating Direction Method of Multipliers (ADMM) [523], which breaks down a large optimization problem into smaller sub-problems. Coordination is achieved through a structured, iterative process where agents solve their local problems and then communicate their results to update a shared, global *consensus* variable, progressively forcing convergence through a system of dynamic penalties. Nedic, Olshevsky, and Shi [524] further explored decentralized methods, establishing how algorithms like decentralized ADMM can achieve efficient convergence using only local communications with immediate neighbors. Similarly, the work by Liu et al. on HVAC energy optimization [525] also followed the same principle. Their work decomposed an HVAC system into multiple subsystems, where each component (e.g., a chiller or pump) acts as an agent. Each agent makes local optimization decisions based only on its own state and limited information from its neighbors, achieving near-optimal energy savings without global knowledge. While mathematically rigorous for convex optimization problems, these approaches assume a formally defined global objective and often rely on synchronous, iterative communication, which can introduce significant overhead in the highly dynamic environments of the CC.

A more flexible alternative is the use of *decentralized heuristics*, which are particularly effective due to their lightweight nature. These methods are typically realized through *multi-agent systems* where a global solution results from local interactions. A notable example is COHDA (A Combinatorial Optimization Heuristic for Distributed Agents) [526], a self-organizing heuristic where each agent manages a partial solution and uses local interactions to collectively approximate a global target. This model was later extended by Bremer and Lehnhoff to COHDAGo [527], adapting the protocol for continuous, non-linear optimization. These works show that decentralized, gossip-based agent coordination can be effectively applied to both discrete and continuous problems, highlighting the versatility of decentralized heuristic frameworks. More recent is another work by Nezami et al. [528], which proposes EPOS Fog, a decentralized multi-agent framework for IoT service placement.

In this proposal, the authors envision a scenario where each node hosts a local agent that autonomously generates service placement plans using local information (e.g., resource capacity, proximity). Plans are shared and optimized through a self-organized tree topology, enabling coordinated, decentralized decisions. EPOS Fog reduces execution delay by up to 25% and improves load balancing by 90% over centralized and heuristic baselines, demonstrating its scalability and resilience in dynamic IoT settings. However, the coordination mechanisms in these approaches rely on an implicit convergence model, which lacks formal guarantees on consensus. This is problematic in the Industrial Compute Continuum, where multiple stakeholders are involved.

The DREAMS framework presented in this chapter directly addresses this gap. Similar to COHDA, COHDAGo, or EPOS Fog, it is an *agent-based heuristic framework* where each Local Domain Manager (LDM) acts as an autonomous agent. The core heuristic logic is driven by the Service Affinity concept [506], which provides a holistic, multi-dimensional, application-aware rule for an LDM to decide when and where to propose a migration. However, DREAMS's key novelty is how it replaces implicit coordination with an explicit and consensus-driven coordination mechanism. Rather than relying on probabilistic convergence, DREAMS integrates a Raft-based quorum voting mechanism to achieve deterministic, fault-tolerant agreement on migration decisions. This design combines the lightweight nature of heuristics with formal consensus, making DREAMS well-suited for managing critical industrial services that demand both flexible autonomy and reliable coordination across the continuum.

## 13.3 Concept and Design

### 13.3.1 Design Principles

The proposed approach implements a multi-agent model, where each domain hosts an autonomous LDM that monitors local Service Affinities and collaborates with peer domains to make decentralized placement decisions. The process is executed in two phases: first, LDMs propose migrations based on estimated local QoS improvements; second, they evaluate the potential global impact through a distributed voting mechanism to reach consensus. Approved migrations are executed with rollback support for fault tolerance and consistency. The design of DREAMS follows these principles:

- *Self-Governed Decentralization*: Each LDM operates autonomously within its domain.
- *Collaborative Optimization*: LDMs coordinate for global goals while maintaining local autonomy.
- *Privacy-Preserving*: Metrics are processed locally to minimize data exposure outside of the domain.
- *Heuristic-Driven Placement*: Decisions are driven by an extensible heuristic model.

- *Fault Tolerance:* Coordination is resilient to LDM failures and supports rollback for consistency.

These principles are implemented following a *Domain-Driven Design (DDD)* approach, which identifies the key challenges of the system and decomposes the domain into bounded contexts, each responsible for a specific functional concern. Within these contexts, entities, aggregates, and services are defined, and domain events are used to enable event-driven communication. Integration patterns are applied to maintain modularity and scalability across system components. This process leads to the structures of LDM, consisting of *Modules* and *Repositories*, which will be presented in the next sections.

### 13.3.2 Modules

The LDM is structured as a composition of modules aligned with the defined contexts, as illustrated in Figure 13.1.

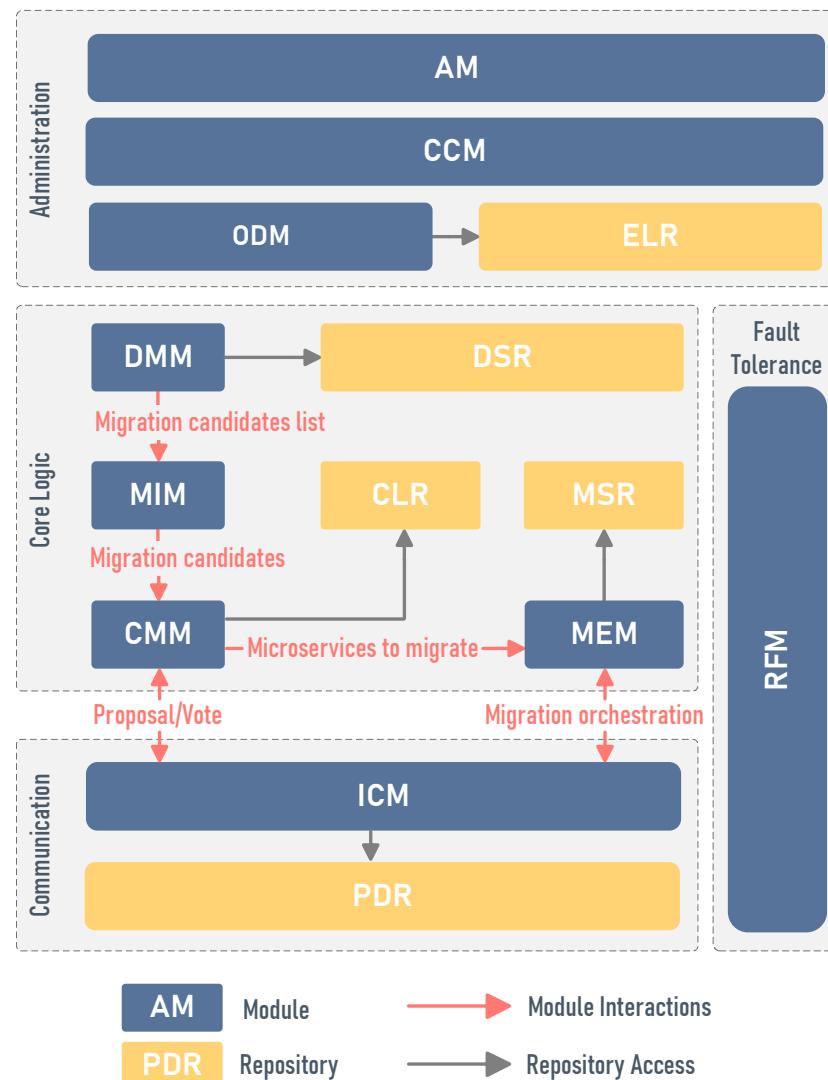


FIGURE 13.1: Overview of Modules and Repositories in LDM System Architecture.

**Administrative Module (AM) - Admin/Policy Context:** Provides tools for administrators to monitor and manage the LDM. This module provides several domain services as follows.

- *Administrative Dashboard:* Provides an interface for monitoring the local domain's health and migration activity.
- *Policy Manager:* Allows administrators to define and update migration policies.
- *Visualizer and Report:* Generates visualizations for dependencies, affinities, and performance metrics.

**Configuration Control Module (CCM) - Configuration Context:** Manages all configuration settings required for the LDM to operate and adapt to changing conditions. Key domain services are:

- *Configuration Repository Service:* Acts as the central storage for all configuration data.
- *Dynamic Configuration Updater:* Manages real-time updates to configuration settings and propagates changes.
- *Configuration Validator Service:* Guarantees that new or updated configurations meet predefined constraints and dependencies.

**Observability and Diagnostics Module (ODM) - Observability Context:** Monitors the system's performance and provides visibility into operations. Its main domain services are:

- *Metrics Aggregator:* Aggregates statistics for migration success rates and system resource usage.
- *Event Logger:* Logs key events for debugging and diagnostics.

**Domain Monitoring Module (DMM) - Monitoring Context:** Responsible for collecting, normalizing, and maintaining runtime metrics and service interaction data within a local domain. Key domain services are:

- *Service Health Monitor:* Aggregates CPU, memory, and traffic metrics; performs unit conversion, filtering, and outlier removal. Stores data in the Domain State Repository.
- *Service Affinity Calculator:* Calculates service affinity scores based on communication patterns and resource usage, using the formulas defined in [506]. Scores are stored in the Domain State Repository for intra-domain affinities, and in the Peer Domain Repository for inter-domain affinities.

**Migration Intelligence Module (MIM) - Optimization/Decision Context:** Manages migration decisions in both leader and follower roles. This module provides several domain services as follows.

- *Migration Eligibility Evaluator:* In the leader role, it selects migration candidates using service metrics, affinity, latency, and resource data. It uses information from both the Domain State Repository and the Peer Domain Repository to select optimal candidates and forwards them to the Proposal Manager in the Consensus Management Module.

- *Cost-Benefit Analyzer*: In the follower role, it evaluates proposals based on expected QoS, resource, and latency impact to cast a vote.

**Consensus Management Module (CMM) - Consensus Context:** Manages the decentralized voting process on migration proposals via a Raft-based protocol, ensuring system-wide consistency through log replication. Key domain services are:

- *Proposal Manager*: Constructs migration proposals based on MIM inputs.
- *Consensus Voting Engine*: Distributes proposals, collects votes, and triggers replication through the Consensus Log Service.
- *Leader Coordinator*: Controls leader election and term management.
- *Consensus Fault Recovery*: Handles incomplete consensus and restores state from logs.
- *Consensus Log Service*: Stores consensus-related logs in the Consensus Log Repository.

**Migration Execution Module (MEM) - Execution Context:** Executes and validates service migrations with rollback mechanisms to maintain consistency. This module provides several domain services as follows.

- *Migration Orchestrator*: Executes migrations post-consensus, checks resource availability and coordinates rollback if needed.
- *Rollback Manager*: Performs rollback transactions for failed migrations.
- *Health Assurance Validator*: Verifies service health before and after migration.

**Inter-Domain Communication Module (ICM) - Communication Context:** Enables inter-LDM communication and awareness of neighboring LDMs' operational status. This module provides several domain services as follows.

- *Inter-domain Monitoring and Health Exchange Service (IMHES)*: Periodically measures latency and exchanges health data with peer LDMs. Data is stored in the Peer Domain Repository.
- *Inter-domain Migration Coordinator*: Coordinates migration-specific messaging with remote LDMs.
- *LDM Discovery Service*: Handles registration and discovery of LDMs, updating their records in the Peer Domain Repository.
- *Inter-domain Communication Service*: Provides end-to-end communication channels among LDMs.

**Recovery and Fault Tolerance Module (RFM) - Recovery Context:** Guarantees quick recovery in case of crashes or failures. This module provides several domain services as follows.

- *State Persistence Engine*: Logs migration actions in a fault-tolerant store for traceability and recovery.
- *Crash Recovery Manager*: Resumes interrupted processes after crashes.
- *Checkpoint Manager*: Saves periodic checkpoints for faster recovery.

### 13.3.3 Repositories

These modules persist data to dedicated repositories, each of which serves a specific role in maintaining either local or distributed system state. The system includes five such repositories:

**Event Log Repository (ELR)** provides observability by storing system health metrics (CPU, memory, and network usage), migration events (successes and failures), error traces, and historical service performance data. It is primarily populated by the ODM’s *Event Logger*.

**Domain State Repository (DSR)** maintains localized information about the computational domain. It stores intra-domain service affinity scores, topology, resource availability, and configuration settings. This repository is primarily populated and updated by the services of the DMM.

**Consensus Log Repository (CLR)** guarantees distributed consistency by storing logs related to the Raft consensus protocol, including Raft messages, committed migration proposals, leader election history, and replication metadata. Utilized by the CMM, it supports consensus operations and ensures all LDMs maintain a consistent view of agreed actions. It also enables recovery from incomplete or failed rounds.

**Migration State Repository (MSR)** tracks the progress of service migrations. It stores details about ongoing and completed migrations, including source and target LDMs, progress checkpoints, and the last known execution step. The repository is used and populated by the *MEM* to ensure that migrations can resume from their last known state in case of failure, prevent conflicting or duplicate migrations, and support rollback operations.

**Peer Domain Repository (PDR)** captures information about other LDMs in the system, including their health status, availability, measured inter-domain latency, membership, and registration details. It also stores affinity scores for inter-domain service interactions. Populated by the ICM’s IMHES, the PDR is used by the Migration Eligibility Evaluator and the Cost-Benefit Analyzer within the MIM to select suitable migration candidates and evaluate incoming proposals.

Together, these *Modules* and *Repositories* form the LDM’s architecture, the implementation details of which are presented in the following section.

## 13.4 Implementation

The LDM was implemented following the principles of *Hexagonal Architecture*, promoting modularity and long-term maintainability. The LDM was developed in Java 17 using the cloud-native Quarkus<sup>1</sup> framework with GraalVM<sup>2</sup> support for optimized runtime efficiency. Inter-module and inter-LDM communication is asynchronous and event-driven, built on Apache Pekko<sup>3</sup> (Akka fork) with built-in clustering and service discovery. Messages are serialized using Protobuf<sup>4</sup>. Decentralized consensus is achieved via the Raft algorithm, implemented using Apache Ratis<sup>5</sup>. Two in-memory

<sup>1</sup><https://github.com/quarkusio/quarkus>

<sup>2</sup><https://github.com/graalvm>

<sup>3</sup><https://github.com/apache/pekko>

<sup>4</sup><https://github.com/protocolbuffers/protobuf>

<sup>5</sup><https://github.com/apache/ratis>

caches using *Caffeine*<sup>6</sup> accelerate access to latency metrics and service states. The system is deployed on *Kubernetes* (GKE), integrating with its API for resource awareness and service migration.

### 13.4.1 Migration candidate selection

To identify the optimal microservice for migration, each LDM executes a local decision-making procedure based on affinity analysis and latency evaluation. The process is structured as follows:

1. **Filtering Phase:** Eliminate from consideration all microservices that are:

- explicitly marked as non-migratable, or
- already located in the cluster with their highest affinity.

This step avoids unnecessary computation by immediately eliminating services that are either administratively fixed or have already been placed in their optimal location, allowing the LDM to focus only on viable candidates.

2. **Cluster Affinity Computation ( $A_c$ ):** For each remaining candidate microservice  $m$ , and for each cluster  $c$  in the system, the *Cluster Affinity Score* is calculated as:

$$A_c(m) = \sum_{v \in \mathcal{N}_c(m)} a_{m,v}(\Delta t)$$

where  $\mathcal{N}_c(m)$  is the set of microservices in cluster  $c$  connected to  $m$ , and  $a_{m,v}(\Delta t)$  denotes the affinity between  $m$  and  $v$  over the time window  $\Delta t$ . This score,  $A_c(m)$ , represents the total affinity between  $m$  and all the microservices currently residing within that specific cluster  $c$ . This step quantifies the *closeness* between the candidate microservice and each cluster, providing the basis for comparing possible migration destinations.

3. **Affinity Scores:** The intra-cluster and inter-cluster affinities are then defined as

$$A_{\text{intra}} = A_{c_{\text{current}}}(m), \quad A_{\text{inter}} = \max_{c \neq c_{\text{current}}} A_c(m)$$

where  $A_{\text{intra}}$  represents local affinity within the current cluster and  $A_{\text{inter}}$  denotes the strongest affinity to any other cluster.

4. **Affinity Gain ( $\Delta A$ ):** Compute the potential affinity gain from migrating the microservice to the most favorable external cluster:

$$\Delta A = A_{\text{inter}} - A_{\text{intra}}$$

A positive  $\Delta A$  indicates that another cluster has stronger overall affinity to the microservice than its current cluster, suggesting potential QoS improvement through migration.

---

<sup>6</sup><https://github.com/ben-manes/caffeine>

5. **Latency Penalty ( $L$ ):** A penalty is introduced to reflect the additional latency cost of migrating to a more distant cluster, scaled according to the affinity gain:

$$L = \begin{cases} \frac{\ell}{1 + \exp\left(\frac{\Delta A}{\gamma_{\text{proposal}}}\right)}, & \text{if } \Delta A > 0 \\ \ell, & \text{otherwise} \end{cases}$$

where  $\ell$  is the latency to the target cluster and  $\gamma_{\text{proposal}}$  is the *Affinity Gain Sensitivity Coefficient* controlling how strongly the system reacts to affinity changes. This step introduces the trade-off between improved affinity and added latency, so that only migrations with meaningful gains justify the cost.

6. **QoS Improvement Score ( $Q$ ):** The overall benefit of migration is then estimated as:

$$Q = \Delta A - L = A_{\text{inter}} - A_{\text{intra}} - L$$

By combining the affinity gain with the latency penalty, it produces a single metric representing the overall improvement, allowing all potential migrations to be ranked on a common scale.

7. **Candidate Selection:** Identify the microservice with the highest  $Q$ . If  $Q > \theta_{\text{proposal}}$ , where  $\theta_{\text{proposal}}$  is the proposal threshold, the migration is proposed.

This procedure enables each LDM to identify migration candidates based on a balance between improved service affinity and increased communication latency (if any). The decision to propose a migration is governed by two tunable parameters that influence the system's responsiveness and migration selectivity. The *Affinity Gain Sensitivity Coefficient*  $\gamma_{\text{proposal}}$  shapes the steepness of the sigmoid function used in the latency penalty calculation. A smaller  $\gamma_{\text{proposal}}$  value results in a sharper transition, allowing only migrations with significant affinity gains to justify the latency overhead, thus favoring conservative and stable migration behavior. Conversely, a larger value makes the penalty curve flatter, allowing more aggressive migrations even for modest affinity improvements, which can help the system respond quickly to dynamic changes but may increase system instability. The second parameter, the *proposal threshold*  $\theta_{\text{proposal}}$ , sets the minimum acceptable net QoS gain. It ensures that only candidates offering meaningful net gains (after accounting for both affinity improvement and latency penalty) are proposed for system-wide evaluation. Together, these parameters enable fine-grained control over the proposal process.

### 13.4.2 Voting Procedure for Migration Approval

After the leader LDM proposes a migration, participating LDMs independently evaluate the proposal based on local impact and system-wide trade-offs. The decision-making procedure for each voting LDM proceeds as follows:

1. **Local Impact Score ( $I_{\text{local}}$ ):** Each LDM first quantifies how strongly its local microservices depend on the migrating microservice  $m$ :

$$I_{\text{local}} = \sum_{v \in \mathcal{N}_{\text{local}}(m)} a_{m,v}(\Delta t)$$

Before proceeding any further, the LDM must quantify its own dependency on the service being migrated.

2. **Non-Affected Clusters:** If  $I_{\text{local}} = 0$ , the LDM immediately casts a *positive vote*, as the migration does not affect local performance. The *positive-vote-by-default* policy ensures that unaffected domains remain cooperative participants and, importantly, shortens convergence time to reach quorum.
3. **Affinity Normalization ( $\tilde{I}_{\text{local}}$ ):** The raw impact score is normalized to produce a measure that is comparable across LDMs:

$$\tilde{I}_{\text{local}} = \frac{I_{\text{local}} - I_{\min}}{I_{\max} - I_{\min} + \epsilon}$$

where  $I_{\min}$  and  $I_{\max}$  are historical extremes, and  $\epsilon$  is a small constant to avoid division by zero.

A raw affinity score is context-dependent and difficult to compare. Normalizing the score transforms it into a dimensionless measure of impact (e.g., a value from 0 to 1). For example, an  $\tilde{I}_{\text{local}}$  close to 1 indicates strong local dependency, whereas a value near 0 indicates minimal sensitivity to the migrating service. This step is essential for ensuring the subsequent weighting logic is applied consistently across all voters.

4. **Latency Difference ( $\Delta\ell$ ):** The change in communication latency from the perspective of the voting LDM is calculated as:

$$\Delta\ell = \ell(c_{\text{target}}, c_v) - \ell(c_{\text{source}}, c_v)$$

This step quantifies the direct cost (or benefit) of the proposed migration for the voter. A positive  $\Delta\ell$  means the service is moving farther away, increasing the latency. A negative value means it is moving closer, reducing the latency.

5. **Local Impact Penalty Weight ( $W_{\text{aff}}$ ):** The LDM then applies a penalty weight based on the normalized local impact:

$$W_{\text{aff}} = \frac{1}{1 + \exp\left(-\frac{\tilde{I}_{\text{local}}}{\gamma_{\text{vote}}}\right)}$$

This mechanism allows each LDM to adjust its sensitivity to latency changes. If its dependency is high ( $\tilde{I}_{\text{local}} \approx 1$ ), the weight is high ( $W_{\text{aff}} \approx 1$ ). If its dependency is low ( $\tilde{I}_{\text{local}} \approx 0$ ), the weight is low, and it becomes more permissive, effectively prioritizes global benefits.

6. **Scaled Latency Penalty ( $P_{\text{lat}}$ ):** The weighted latency cost is normalized against the maximum possible latency in the system:

$$P_{\text{lat}} = \frac{\Delta\ell \cdot W_{\text{aff}}}{\ell_{\max}}$$

This step normalizes the weighted penalty against the maximum possible latency in the system, converting it into a proportional score. A  $P_{\text{lat}}$  of 0.1 represents a 10% penalty relative to the worst-case latency. This enables a single, system-wide voting threshold to be applied consistently.

7. **Voting Decision:** If  $P_{\text{lat}} < \theta_{\text{vote}}$ , the LDM casts a *positive vote*; otherwise, it casts a *negative vote*. This prevents small, localized penalties from blocking changes that yield broader system-level benefits.

The voting procedure was designed to ensure that migration proposals are evaluated in a decentralized yet coordinated manner, balancing local performance considerations with system-wide objectives. Two tunable parameters govern this evaluation. The *Local Impact Sensitivity Coefficient*  $\gamma_{\text{vote}}$  controls how strongly each LDM reacts to its own dependency on the migrating service. It adjusts the sigmoid function that weighs the local impact when computing the scaled latency penalty. A smaller value makes LDMs highly sensitive to even small dependencies, effectively favoring local stability. In contrast, a larger  $\gamma_{\text{vote}}$  softens this response, promoting more global cooperation even at some local cost. The *voting threshold*  $\theta_{\text{vote}}$  determines the upper limit of the acceptable latency penalty. If the scaled penalty exceeds this threshold, the LDM rejects the migration proposal to protect local performance. These two parameters allow each LDM to balance its autonomy with collective optimization goals, enabling a flexible consensus process in a multi-stakeholder continuum.

## 13.5 Evaluation

Due to the complexity of the design, the evaluation is conducted using both qualitative and quantitative approaches. The results of each are presented in the following sections.

### 13.5.1 Qualitative evaluation

DREAMS is designed to support decentralized coordination across a globally distributed compute continuum, where nodes operate independently but must sometimes reach agreement on critical service migration decisions. This section discusses the reasoning behind the main architectural choices in DREAMS: consensus, peer discovery, and consistency. It also highlights how each contributes to the overall scalability, responsiveness, and resilience of the design.

**Consensus Mechanism** The *Raft consensus algorithm* is used to coordinate migration decisions in a fault-tolerant manner. Although Raft is not Byzantine Fault Tolerant (BFT) and assumes crash-only failures, it provides a practical balance between simplicity, performance, and correctness in environments where nodes are

cooperative but may experience failures or temporary disconnections. Compared to more complex alternatives like BFT-based protocols or Paxos, Raft offers lower latency, better scalability, and easier implementation, making it suitable for resource-constrained scenarios in the Compute Continuum [529]. It tolerates up to  $\lfloor (N - 1)/2 \rfloor$  node failures and provides log replication as well as quorum-based agreement without excessive coordination overhead.

**Peer Discovery and Monitoring** Effective decentralized coordination also requires each LDM to discover and monitor peers in real time. Various options exist for this task, including centralized registries (e.g., etcd, Consul), static bootstrapping, distributed hash tables (DHTs), and overlay libraries like libp2p. Centralized solutions simplify discovery but violate decentralization principles and introduce single points of failure. Static configurations are too inflexible for changing environments, while DHTs and overlays add unnecessary protocol complexity. To avoid these limitations, DREAMS uses a lightweight *gossip-based protocol* to exchange peer health and availability information. This approach improves resilience and scalability under network instability or partial failures, which aligns with the system's distributed design.

**Consistency Model** Given the wide-area deployment of LDMs and the varied rate at which domains evolve, enforcing strong consistency at all times would significantly degrade responsiveness. Therefore, DREAMS adopts a *hybrid consistency model*. For non-critical operations such as health monitoring and affinity estimation, modules like the Inter-domain Monitoring and Health Exchange Service (IMHES) and the Service Affinity Calculator operate under *eventual consistency*. This allows decisions to be made using slightly outdated data, which is acceptable for trend-based optimization and improves system responsiveness. In contrast, strong consistency is enforced only during migration coordination, where the Raft-based consensus guarantees agreement and system-wide correctness. This selective use of strong consistency maintains reliability without sacrificing scalability or latency elsewhere in the system.

To evaluate the system behavior and the global optimization capabilities of the LDMs, three clusters were deployed across geographically distributed Google Cloud regions: *us-east4* (Northern Virginia, USA), *europe-west3* (Frankfurt, Germany), and *asia-southeast1* (Jurong West, Singapore). Each LDM was hosted on an *e2-standard-4* virtual machine (4 vCPUs, 16 GB RAM), running Ubuntu 22.04 LTS with identical configurations. Each LDM managed a simulated network of microservices with pre-defined inter-service affinity values. These affinities were assigned between each pair of services to reflect realistic interaction patterns, as illustrated in Figure 13.2. The optimization problem can be formalized as follows: given an undirected graph  $G = (V, E)$  where edge weights  $w(e)$  denote affinity scores, the objective is to minimize the sum of affinities across domain boundaries:

$$\min_{\substack{u \in V_i, v \in V_j \\ i \neq j}} \sum w(u, v)$$

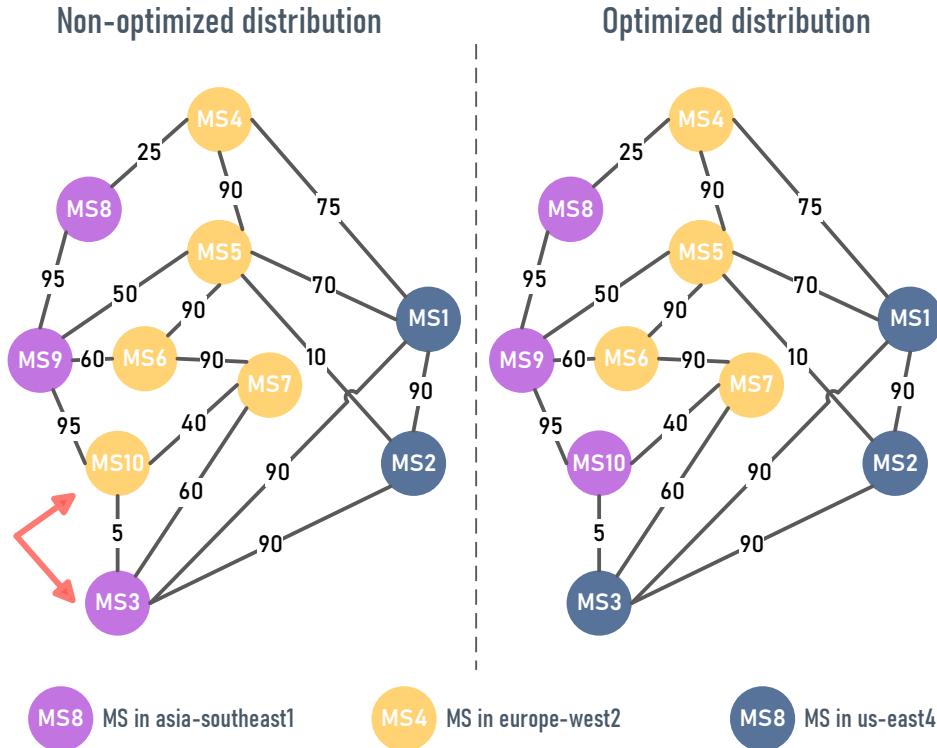


FIGURE 13.2: Microservice distribution before (left) and after (right) optimization.

In the initial (non-optimized) distribution, the total inter-domain affinity value was 630. An offline global optimization produced a lower cost of 395, achieved by migrating MS3 from `asia-southeast1` to `us-east4` and MS10 from `europe-west3` to `asia-southeast1`.

During evaluation, the system gradually converged toward this optimal state. In normal operational scenario, the LDMs in `asia-southeast1` and `europe-west3` successfully identified the correct migration candidates and independently initiated the appropriate proposals. To test fault tolerance, a leader LDM failure was simulated. As expected in a Raft-based setup, the remaining nodes entered a brief leader election phase, during which optimization paused until a new leader was elected. Once the election completed, the system automatically resumed operation, and the failed LDM later rejoined the cluster without manual intervention. These results confirm the framework's resilience and its ability to recover and maintain coordination under failure conditions.

### 13.5.2 Quantitative evaluation

To complement the qualitative analysis above, the evaluation also included quantitative measurements focusing on two main procedures that influence system scalability. The first is *LDM registration*, which is handled via gossip-based mechanism for cluster membership and service discovery. The second is the *migration voting process*, which combines asynchronous message exchange for proposal evaluation with consensus and decision logging through the Raft protocol.

### LDM registration

LDM registration time was measured by synchronizing the system clocks of all nodes using `chrony`. Each existing LDM monitors cluster membership events using Pekko's `ClusterEvent.MemberUp` event. The registration time is calculated as the time elapsed between the LDM's startup and the moment it is recognized as a member by all peers. Experiments were conducted with cluster sizes ranging from 3 to 20 LDMs, including 2 seed nodes. Each configuration was repeated three times to obtain average registration times.

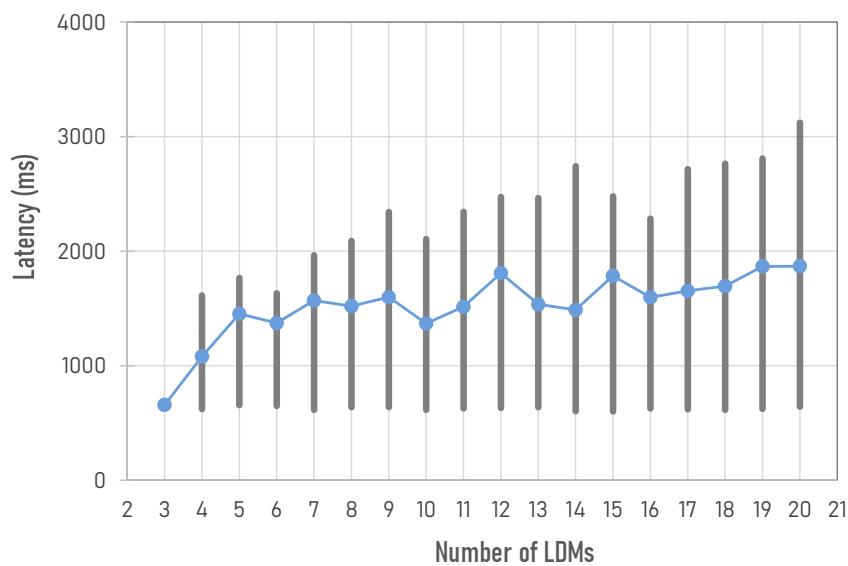


FIGURE 13.3: LDM registration time across cluster sizes.

The results are illustrated in Figure 13.3. The vertical bars represent the range between the minimum and maximum registration times observed across all existing LDMs, while the blue line indicates the mean registration time. The minimum registration time (*best-case scenario*) remains relatively stable across all configurations (average 623.9 ms). This consistency is expected since the seed nodes, which are typically contacted first, recognize a new member almost immediately. Indeed, those values are recorded on either one of the two seed nodes.

The maximum registration time, however, reflects the convergence latency: the time needed for the *last node* in the cluster to become aware of the new member through the gossip protocol. This value increases with the number of LDMs, from 556 ms (3 nodes) to 3,121 ms (20 nodes). As the cluster grows, the number of random hops the information must take to reach every peer can increase, leading to this higher worst-case latency. However, the trend is not strictly linear or logarithmic due to the probabilistic nature of the gossip protocol, in which each node randomly selects peers for periodic state exchange. Importantly, the average registration time shows a clear sub-linear growth trend, indicating that the process scales efficiently. The time required to register a new LDM does not grow proportionally with cluster size, confirming that gossip-based membership remains effective even in larger deployments.

## Migration Voting

The performance of the migration voting process was measured as the time elapsed between the initiation of a migration proposal by the leader LDM and the point at which the final decision was committed to the replicated log. This duration captures the complete lifecycle of the decentralized coordination process, including (1) the propagation of the proposal to all participating LDMs via Apache Pekko's asynchronous messaging, (2) the local evaluation and submission of votes based on affinity and latency metrics, and (3) the aggregation of votes by the leader and the subsequent consensus and replication using Apache Ratis. The actual execution of the migration was excluded from this measurement to isolate the coordination overhead. Once a quorum of positive votes was reached, the leader appended the decision to the Raft log.

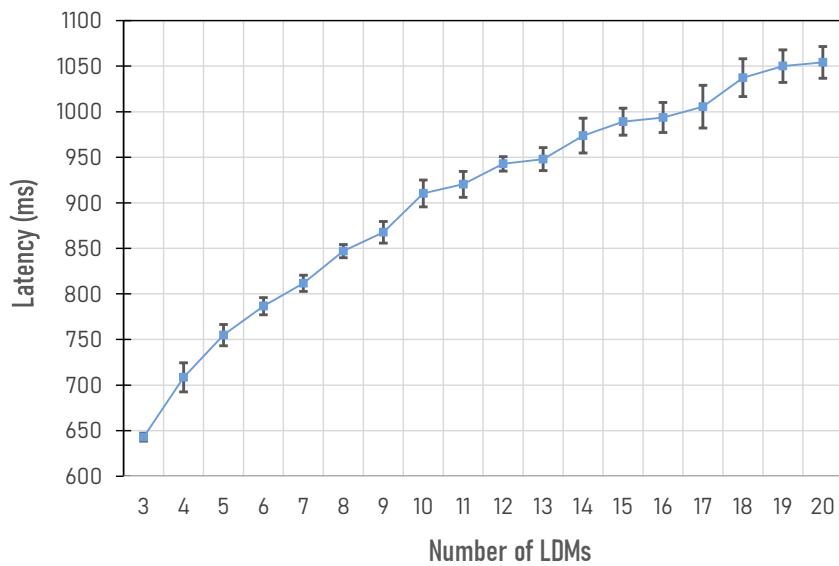


FIGURE 13.4: Migration voting latencies across cluster sizes.

As shown in Figure 13.4, the average latency from proposal to log commitment increased from 642.8 ms with 3 nodes to 1,054.1 ms with 20 nodes. Each point represents the mean of five independent runs, and the low standard deviations across configurations indicate that the protocol behaves consistently. The increase in latency follows a sub-linear pattern, which aligns with the expected behavior of quorum-based systems. This can be explained by the composition of the workflow:

1. *Proposal propagation via Pekko*:  $O(1)$  (asynchronous and concurrent).
2. *Local evaluation and vote submission*:  $O(N)$  total messages, but parallel execution at each LDM.
3. *Vote aggregation*:  $O(N)$ , with early termination upon reaching quorum ( $N/2 + 1$ ).
4. *Log replication via Raft*: The leader sends `AppendEntries` to all followers in parallel ( $O(N)$  messages). However, it only needs to wait for acknowledgments from a quorum to commit the entry. Therefore, the commit latency is governed by the network RTT to the median follower and is largely independent of the cluster size  $N$ , i.e., it exhibits  $O(1)$  latency with respect to  $N$ .

Although the theoretical complexity suggests constant-time performance, an upward trend appears in the evaluation results due to small increases in leader workload and network variation as the cluster expands. Nonetheless, the overall growth remains sub-linear, confirming the scalability of the decision process. The results demonstrate that DREAMS' decentralized voting mechanism provides reliable and efficient coordination, making it suitable for large-scale, distributed deployments across the Compute Continuum.

## 13.6 Chapter Summary

This part of the dissertation has presented the development of affinity-based service management, following a progression from the validation of the core concept in a centralized setting to the design of a fully decentralized framework.

In the first stage, the SAGA framework confirmed that, *Service Affinity* concept is effective in improving service placement and can achieve measurable performance gains. However, this centralized approach also demonstrates its inherent architectural limitations, which are its poor scalability and incompatibility with the decentralized, multi-domain nature of the CC.

Building upon these findings, the next step introduced DREAMS, a framework that extends the idea of Service Affinity into decentralized systems. DREAMS replaces centralized decision-making with a network of autonomous Local Domain Managers that coordinate through a lightweight consensus mechanism. By combining affinity-based heuristics with quorum voting and Raft consensus, the system achieves fault-tolerant coordination and hybrid consistency, while maintaining local data processing for privacy and autonomy. A prototype implementation demonstrated the practicality of this design. Experimental evaluation showed that LDMs were able to collaboratively reach optimal placement decisions. The sub-linear scalability achieved in these tests further indicates that DREAMS can support large-scale, dynamic Industry 4.0 environments.

Together, SAGA and DREAMS form a complete narrative: SAGA established *what* should be optimized (Service Affinity), and DREAMS defined *how* this optimization can be achieved in a decentralized and resilient manner. These contributions provide a strong foundation for advancing intelligent and decentralized resource management in future industrial compute continuum.

However, several areas remain open for further investigation. First, the current implementation assumes cooperative domains and relies on a crash-tolerant consensus model, which does not account for Byzantine behavior. Incorporating Byzantine fault tolerance could enhance robustness in adversarial settings (e.g., in a supply chain with competing commercial partners), although at the cost of increased complexity. Second, the leader election currently uses a simple round-robin mechanism, fair but suboptimal in dynamic environments, where a more adaptive, capability-aware policy could improve responsiveness. Third, the system allows only one active migration proposal per leader term; future work could explore batching, prioritization, or parallel proposal handling to better manage bursts in service reallocation demands. Finally, key parameters (e.g.,  $\gamma_{\text{proposal}}$ ,  $\gamma_{\text{vote}}$ ,  $\theta_{\text{vote}}$ ) could be optimized automatically using machine learning to enhance adaptability.

## **Part VII**

# **Dissertation Conclusion**



# 14 Contribution Summary

## 14.1 Novel Architecture and Frameworks for the Industrial Compute Continuum

This final part of the dissertation provides a overall summary of the entire work and its main contributions. To summarize the central focus of this research, the work was developed around a single research question:

*Main Research Question:* How can a unified cloud-native reference architecture be designed and implemented within the Compute Continuum to meet the changing demands of Industry 4.0?

Structured around this central question, each part of this dissertation addresses a different aspect of the research problem. The research follows an engineering-oriented approach that covers both conceptual development and practical implementation. As mentioned throughout this work, the dissertation is organized into six stages, with each stage corresponding to one part:

- *Motivating (Part I):* This dissertation began with an introduction to the challenges facing modern industrial systems. Through a review of the historical evolution of the industrial revolutions, paralleled by the development of various computing models, the dissertation argues that Industry 4.0 and cloud-native technologies represent the latest advancements in Operational Technology (OT) and Information Technology (IT), respectively. Together, they offer an opportunity to merge both domains into a unified design to build highly-efficient modern industrial systems. After motivating the research problem, this part defined the main research question and three sub-questions, which were addressed through an iterative Design Science Research process.
- *Understanding (Part II):* The dissertation continued with a systematic review of state-of-the-art research on the Compute Continuum. Based on this foundation, it proposed a taxonomy that maps the current research landscape, synthesized a definition of the Compute Continuum, and identified the main gaps for future work. Based on an analysis of 291 studies, the review showed that the Compute Continuum has become an actively growing research area. Most of the work so far has focused on engineering solutions to address its many technical challenges. Several key themes have been identified and were also reflected throughout this dissertation itself: the move toward decentralization, the use of cloud-native technologies, the need for low-latency processing, concerns about security and privacy, and the challenge of achieving full interoperability among components.

- *Designing (Part III):* Subsequently, this dissertation presented one of its central contributions, which is the *Cloud-Native Reference Architecture for the Compute Continuum in Industry 4.0* (CRACI). This architecture was designed using state-of-the-art principles, enabling the convergence of IT and OT domains by incorporating the latest developments from both. One of CRACI's main objectives was to overcome the limitations of existing designs, particularly the balance between abstraction and granularity as well as the rigidity of hierarchical structures. By applying cloud-native and data-driven principles to address the challenges found in modern manufacturing systems, CRACI provides a novel approach to designing the Industrial Compute Continuum. The entire architecture was built upon four foundational pillars (Trust, Governance, Observability, and Lifecycle Management), establishing a structured foundation to support the required capabilities across the continuum. The core principles of CRACI were further evaluated through a concrete industrial analytics use case, the *Microservices-based Architecture for Industrial Data Analytics* (MAIA), which provided evidence-based validations of the design rationale.
- *Implementing (Part IV):* Building upon this foundation, the dissertation provided in-depth analyses to evaluate different approaches for realizing the CRACI architecture. This evaluation was conducted from two perspectives: *service development* and *service deployment*. From the development perspective, a comparative analysis was conducted across several microservice development frameworks. The comparison analyzed not only their design philosophies and features but also their performance, measured through a common data analytics pipeline. From the deployment perspective, the dissertation presented a detailed comparison between Unikernels and Containers, analyzing their behavior using different workload characteristics, implemented in different programming models, and executed across multiple environments. The results from both analyses demonstrated the complexity involved in selecting suitable approaches. There is no single optimal solution, each option has its own strengths and weaknesses, often sophisticated and context-dependent, requiring an in-depth understanding to make informed decisions. This dissertation advocates for a *polyglot approach*, combining multiple tools and technologies according to the specific requirements of each environment or service.
- *Operating (Part V):* One of the key challenges in distributed computing models such as the Compute Continuum is to maintain system flexibility, particularly through service portability. While this is straightforward for *stateless* services, it becomes significantly more challenging for *stateful* services. This dissertation addressed this issue by proposing the *Message-based Stateful Microservice Migration* (MS2M) framework. The main idea behind MS2M was to utilize the existing communication fabric, which already exists in many cloud-native implementations, to support state synchronization during migration. By shifting the approach from *memory transfer* to *state reconstruction*, MS2M enables live, minimal-downtime migration of stateful services while maintaining an architectural harmony with the message-driven and event-driven nature of the CRACI architecture. Although the first MS2M

prototype has limitations, particularly in handling the *unbounded message replay* problem, the second prototype incorporated several improvements. Besides its integration with Kubernetes, the second prototype also introduced the *Threshold-based Cutoff Mechanism*, which formalizes the cutoff threshold  $T_{\text{cutoff}}$  and provides a flexible way to balance total migration time and downtime according to system requirements. Overall, the insights from this part demonstrated the advantages and potential of an *application-aware approach* to service management. Rather than focusing only on resource consumption, understanding the dynamics among components in a distributed system can lead to more effective and context-aware optimization strategies.

- *Optimizing (Part VI):* While MS2M provided a message-driven mechanism for service portability, it did not include a method for deciding *when* and *where* a service should be migrated. Therefore, this part complemented the MS2M proposal by addressing this placement challenge using the concept of *Service Affinity*. It is defined as a composite value that captures the relationships among microservices, effectively modeling a microservice cluster as a weighted graph. By doing that, the service placement problem was transformed into a graph partitioning problem, which was solved using a  $k$ -way Kernighan–Lin algorithm (an extended version of the original algorithm). Similar to MS2M, two prototypes were designed and implemented to apply Service Affinity for microservice migration decisions. The first prototype, named the *Service Affinity Graph-based Approach* (SAGA), is a centralized model, fully integrated into Kubernetes’s control plane. Although SAGA demonstrated that Service Affinity concept can effectively optimize microservice placement, its centralized nature creates several limitations. To address this, the dissertation also introduced a decentralized framework named DREAMS (Decentralized Resource Allocation and Service Management). Instead of relying on a single centralized decision-maker, DREAMS adopts a *multi-agent model*, where Local Domain Managers (LDMs) manage and represent different computational domains, collaboratively reach consensus on migration decisions. This part further emphasized decentralization as a key direction in the evolution of the Industrial Compute Continuum.

The summary of this dissertation also revisits the main contributions of the research. Although this dissertation does not formally define the six-stage model it follows, the model reflects the author’s own logic in approaching and solving the research problems throughout the doctoral journey. There are certainly alternative approaches to addressing the identified research challenge, and this dissertation represents only one perspective within that broader landscape. However, the six-stage model itself can also be seen as a secondary contribution of this work, as it provides a systematic approach to designing and engineering modern industrial systems within the context of the Compute Continuum. Next, the sub-research questions are revisited, and the corresponding answers are presented in detail.

## 14.2 Answers to Sub-Questions

The main research question was further broken down into three specific sub-questions, which are detailed below.

### RQ1: Foundational Architectural Principles

*RQ1: What are the architectural principles and design patterns that form the foundation of a decoupled, and event-driven system capable of overcoming the structural limitations in traditional hierarchical models such as ISA-95?*

Part III of this dissertation addressed this question directly through the design of the CRACI architecture. The analysis of three existing model categories (standards, industry practices, and academic proposals) suggested that the rigidity of the ISA-95 pyramid can be mitigated by applying cloud-native principles. By adopting message-driven and event-driven principles, CRACI enables more flexible communication channels among services. Instead of a strictly *hierarchical communication model*, its publish–subscribe model enables *cross-layer communication*. In addition, two architectural patterns address limitations of ISA-95: a data-centric Hub-and-Spoke model (implemented by the Asset Representation Services) and an asynchronous Event-Driven Architecture (implemented by the Core Platform Services’ event bus). Together, these patterns break down data silos and enable flexible, non-hierarchical communication across the continuum.

Moreover, CRACI distributes decision-making capabilities across the continuum, allowing functions to be deployed dynamically according to system requirements: low-latency services can run at the Edge, while computationally intensive tasks are handled in the Cloud. Unlike ISA-95, which primarily focuses on vertical data flows within a single organization, CRACI also enables horizontal data exchange across organizational boundaries. This is achieved through its *External and User-Facing Services*, which extend from Edge to Cloud and support integration at any layer.

Finally, the evaluation with MAIA pointed the importance of treating non-functional requirements as fundamental architectural principles. This is realized through the formalization of four Foundational Pillars—Trust, Governance, Observability, and Lifecycle Management—which embed the disciplines necessary to ensure that distributed systems remain secure, compliant, and adaptive by design.

### RQ2: Implementation and Deployment Trade-offs

*RQ2: What are trade-offs in performance, resource efficiency, and scalability between key implementation (microservices frameworks) and deployment paradigms (containers vs. unikernels) when realizing this architecture?*

Through a series of systematically designed evaluations, several valuable insights were derived in Part IV of this dissertation, directly addressing RQ2.

- *Development Frameworks:* When it comes to choosing a development framework, the evaluation showed a series of trade-offs that need to be carefully considered. The first concerned the decision of whether to rely on an existing framework or to build microservices entirely from scratch. With the latter approach,

organizations retain full control over the architecture and its components, but the complexity of distributed systems require steep learning curve, and could lead to maintenance challenges later on. In contrast, adopting a framework accelerated the initial development process, as all the reviewed frameworks provided essential functionalities for microservice-based systems. However, reliance on a framework also introduced certain constraints, in some cases, the framework itself proved to be short-lived or reached end-of-life status, as in the case of Lagom. This dependence is always a risk for organizations maintaining a large-scale system for a long time horizon. Regarding runtime performance, further trade-offs were also identified. Mature, JVM-based frameworks such as Spring Boot demonstrated to be useful in complex enterprise environments due to their rich ecosystems and robust integration capabilities. However, this convenience entailed higher memory consumption and slower startup times, making them less suitable for deployment on resource-constrained edge devices. In contrast, lightweight frameworks such as Go Micro delivered excellent runtime performance and a smaller footprint, though they required significantly more effort to integrate new capabilities.

- *Deployment Paradigms:* The trade-offs involved in selecting deployment paradigms were similar, if not even more complex. Many previous publications have highlighted the advantages of unikernels over containers in terms of security and performance. On one hand, the results confirmed the benefits of unikernels in terms of startup speed and deployment footprint, making them particularly advantageous for on demand, latency-sensitive applications. On the other hand, the analysis also showed that performance trade-offs were more nuanced, being highly dependent on the application's runtime environment and resource constraints. For ahead-of-time (AOT) compiled languages such as Go, unikernels achieved better performance on resource-constrained devices by minimizing operating system overhead. In contrast, for just-in-time (JIT) environments such as Node.js, a *performance crossover* was observed under heavy memory pressure, where the mature memory management of Docker's Linux kernel provided a clear advantage. In well-provisioned, multi-core environments, Docker further benefited from utilizing the host's optimized kernel for parallel workloads, especially for services written in languages like Go. Overall, while Docker containers remain the practical default for most use cases due to its extensive ecosystems, unikernels represent a promising alternative for specialized edge functions, especially when paired with AOT-compiled applications.

The conclusions drawn from Part III demonstrated that there are numerous possible approaches to implement the CRACI architecture. Choosing a specific approach depends on a variety of factors, requiring in-depth knowledge. Nevertheless, the variety of alternatives is itself an advantage, both of cloud-native systems in particular and distributed systems in general. This dissertation advocates for a *polyglot approach* to implement the architecture, meaning applying different technical stacks depending on the concrete context. To achieve this, beyond the insights provided in this work, system architects can follow the same six-stage

process adopted in this dissertation to derive their own context-specific insights and design decisions.

### RQ3: Dynamic Management of Stateful Services

*RQ3: How can stateful services be dynamically managed and migrated across the compute continuum by utilizing application-level semantics and decentralized coordination to ensure both high availability and operational efficiency?*

First, to address the challenge of stateful service migration, Part V introduced the Message-based Stateful Microservice Migration (MS2M) framework. This work demonstrated that, by utilizing the application's existing communication fabric, it is possible to reconstruct service state through orchestrated message replay instead of direct memory synchronization. The evaluation confirmed that this approach reduces service downtime compared to traditional methods, enabling minimal-downtime migration suitable for high-availability industrial systems.

Second, to address the need for service management, Part VI introduced the SAGA and DREAMS frameworks. These frameworks demonstrated that, resource management can be made more intelligent by using Service Affinity, a multi-dimensional model of inter-service relationships. Conceptually, Service Affinity extends conventional optimization approaches from relying solely on resource consumption metrics to including application-level semantics. With a deeper understanding of the dependencies among microservices, optimization strategies can be designed not only to improve latency and resource utilization but also to ensure policy compliance and maintain functional dependencies. For small- and medium-scale systems, a centralized approach can effectively use Service Affinity to determine optimal service placement. However, such an approach faces inherent limitations, such as single points of failure, limited scalability, and potential privacy and fairness concerns. These are better addressed in a decentralized model, which is also the focus of the DREAMS framework. By combining a vote-based consensus mechanism, a hybrid consistency model, and a quorum-based decision-making process, DREAMS provides a scalable yet robust solution for decentralized service management across the continuum.

# 15 Discussion and Conclusion

## 15.1 Key Insights

Throughout this dissertation, a number of recurring themes were identified. They reflect the key insights gained from this research and outline the main challenges and design considerations relevant to compute continuum systems.

### 15.1.1 The Complexity of Trade-off Decisions

Like any engineering problem, this research shows that every decision involves a trade-off. However, within the continuum environment, these trade-offs are rarely binary, such as “option A is better than option B,” or even “option A is better than option B in context X.” Instead, the decision-making complexity goes much deeper. Throughout this work, a wide range of trade-offs had to be considered: whether to use a framework or not, which programming language to adopt, containers versus unikernels, how to configure MS2M parameters, and many more. For example, Part III highlighted this clearly, with concrete conclusions such as: “Nanos Unikernel combined with an AOT-compiled language outperforms Docker containers for CPU-bound processes in resource-constrained environments.” Such conclusions underline the necessity of a systematic approach to system-level decisions. The structure of this dissertation, organized into six phases of Motivating, Understanding, Designing, Implementing, Operating, and Optimizing along with its foundation in the Design Science Research methodology, reflects this systematic approach. Deeply analyzing and understanding system requirements at multiple levels is a prerequisite for making sound architectural choices.

This leads to an important question: Is CRACI suitable for industrial systems? As established earlier, no single architectural approach fits all scenarios. CRACI was not intended to be a one-size-fits-all solution. Rather, it serves as an *architectural navigation system*, offering directions and structures that industrial architects can adapt based on their unique needs. CRACI presented a *spectrum of decisions*, such as the *latency spectrum*, which clearly differentiates Operational Intelligence Services (designed to run at the edge, optimized for low latency) from Strategic Intelligence Services (executed in the cloud, optimized for compute power). Or the *integration spectrum*, where it distinguishes between Brownfield Assets (legacy devices) and Greenfield Assets (modern equipment). This dissertation does not aim to provide a single, definitive solution. Instead, it offers a series of in-depth trade-off analyses intended to guide system architects toward more informed decision-making. The contribution lies not only in the specific technical results, but also in the systematic approach to reasoning about complex systems within the Compute Continuum, which is referred to as the six-stage model.

However, CRACI and the concepts proposed in this dissertation are particularly aligned with the needs of *large-scale smart factories and integrated supply chain ecosystems*. These environments are characteristically complex and heterogeneous, spanning the whole continuum. They demand multi-layered analytics, high availability, and rapid reconfiguration, all within a multi-stakeholder ecosystem. CRACI, MS2M or DREAMS were designed to precisely support these requirements. On the other hand, the proposed architecture may be *over-engineered* for scenarios such as isolated operations, static and repeated processes, or systems with extremely strict real-time constraints. In such cases, low-level, monolithic, purpose-built embedded systems remain more appropriate than a highly distributed, cloud-native architecture like CRACI.

### 15.1.2 The Importance of Flexibility

The previous conclusion naturally leads to another insight: flexibility is essential for realizing the full potential of CC systems. This principle is of particular relevance in traditional industrial systems, which are typically built on rigid, closed architectures and often rely on proprietary protocols, vendor-specific technologies, and business models based on closed ecosystems.

These limitations were observed in practice, for example in the case of accessing data from an industrial Laser Powder Bed Fusion (LPBF) 3D printer. The manufacturer deliberately restricted direct access to the machine's telemetry data, offering instead a paid optimization and maintenance service package. As long as such practices persist, building truly interoperable, connected industrial systems remains a substantial challenge.

While this dissertation acknowledges the influence of non-technical factors, it places a strong emphasis on an open and flexible approach to system design. To that end, the proposed architecture maintains a balance: it is abstract enough to be applied across different contexts, yet specific enough to provide practical guidance for industrial deployment based on cloud-native principles.

Importantly, this work refrains from prescribing any single technology stack, framework, or application. In contrast, this dissertation actually advocates a polyglot approach. The case of Lagom, which is a highly opinionated framework that was eventually discontinued by its own maintainers, serves as a cautionary example. Technology evolves rapidly, and the tools used in this dissertation may become obsolete in the near future. Therefore, the true contribution of this work lies in the underlying principles and methodology, which offer greater longevity and reusability.

Across all system designs and methods proposed in this dissertation, a wide range of options are intentionally made available to accommodate diverse requirements and evolving technological landscapes. The concepts behind CRACI, MS2M, SAGA, or DREAMS can also be reused in different contexts and implemented using different technical stacks.

### 15.1.3 The Significance of Application-Aware Management

One of the key insights of this dissertation is the realization that the modern software architecture paradigms, which are built on microservices, event-driven

patterns, and message-based communication, enables new possibilities for managing services in Industry 4.0 environments. Most importantly, it enables the design and implementation of novel techniques that are collectively referred to in this dissertation as *application-aware management*. As discussed in the related work section, existing approaches often focus on optimizing generic parameters such as resource utilization, throughput, or latency. In contrast, this dissertation introduces methods that are closely aligned with the architecture's underlying principle, as illustrated in the following examples:

- *MS2M framework for stateful migration (Part V)*: This framework introduces a fundamentally new approach to service state management during migration. That state reconstruction mechanism was made possible by the event-driven nature of cloud-native systems. In addition, message replay inherently creates an auditable trace, making migrations easier to debug and monitor. If an error occurs after migration, engineers can retrace the message stream to pinpoint the fault. This improves debuggability in complex systems and reinforces the *Observability* pillar of the CRACI architecture. The framework also enables control over the trade-off between application downtime and total migration time, balancing availability with resource overhead.
- *SAGA framework for application-driven placement (Part VI)*: This framework demonstrates that resource management becomes more effective when it goes beyond CPU and memory metrics. By introducing the concept of *Service Affinity*, the dissertation advocates for a more complex, multi-dimensional approach that can capture the dependencies and interaction patterns among microservices.

In other words, within the Compute Continuum, the most effective management strategies are those that bridge the gap between the application and the infrastructure layer. These strategies must not only exhibit a deep understanding of the services themselves, but also provide platform-agnostic solutions. Only then can they be truly applicable within the highly heterogeneous and dynamic environment of the continuum.

#### 15.1.4 The Hidden Overhead of Cloud-Native Architecture

While the adoption of cloud-native technologies is the key argument of this dissertation, several analyses demonstrated that these technologies come with hidden overhead when used without proper consideration. The evaluation of containerization in Part III demonstrated this most clearly: a poorly optimized setup can cause a significant increase in artifact size and memory usage, offsetting the theoretical efficiency gains.

This observation highlights the importance of the Governance and Lifecycle Management pillar proposed in the CRACI architecture. These aspects should be prioritized and integrated across the entire service lifecycle, from deployment to operation and optimization. More broadly, this finding reflects the overall intention of the dissertation: it does not claim that cloud-native technologies are the optimal solution for all industrial systems. Instead, it provides a detailed perspective for

practitioners interested in this paradigm, outlining how such systems can be designed, operated, and optimized while addressing practical engineering challenges.

By discussing both the strengths and limitations of cloud-native technologies and by presenting a transparent evaluation methodology, this dissertation seeks to provide a neutral and balanced view. The goal is not only to inform but also to encourage stakeholders to critically assess and experiment with these technologies in their own contexts, ultimately enabling decisions that best fit their specific industrial environments.

## 15.2 Limitations and Future Work

While this dissertation addressed many aspects of cloud-native systems in industrial contexts, its findings are subject to several limitations.

- *Validation scopes:* The evaluations, while carefully designed, were conducted in a controlled, simulated environment on public cloud infrastructure. The complex and often unpredictable network conditions, hardware heterogeneity, and electromagnetic interference of a live factory floor could introduce performance characteristics not captured in this study. A full-scale deployment in a physical production facility would be the ultimate validation.
- *Assumptions of the Proposed Models:* The frameworks presented here operate under specific assumptions. The DREAMS framework, for instance, relies on a crash-fault tolerance model and assumes cooperative, non-malicious actors. It is therefore not resilient to Byzantine faults, which may be a significant limitation in multi-stakeholder, zero-trust industrial ecosystems where commercial interests differ among involved organizations.
- *Technological Dependencies:* The implementations used a specific set of modern cloud-native technologies (e.g., Kubernetes, Java with Spring Boot, Pekko). While the underlying architectural principles are technology-agnostic, the evaluation needed to be conducted on specific technology stacks. As such, the results and trade-off analyses are influenced by these choices. Employing alternative technologies such as different programming languages (e.g., Rust), container runtimes, or messaging systems, would likely lead to different quantitative outcomes.

From those limitations, several potential directions for further development are listed below.

- *Towards a Zero-trust, Byzantine-Tolerant Continuum:* The next logical step is to extend the proposed frameworks (such as DREAMS) to handle adversarial environments by incorporating Byzantine Fault Tolerance (BFT). This would require examining how stronger, though more complex, consensus protocols (e.g., Practical Byzantine Fault Tolerance (PBFT) and its variants) can be adapted for use across wide-area networks with acceptable efficiency. At the communication layer, there is a need for secure yet lightweight protocols, particularly suited for low-power edge devices. In multi-stakeholder settings,

interoperable identity management would further strengthen trust and accountability.

- *Automating the Foundational Pillars with AIOps:* The enforcement of the four Foundational Pillars in CRACI architecture could be automated using AI and Machine Learning (AIOps). This could involve developing predictive monitoring models capable of anticipating service failures or SLA violations before they occur. In addition, with the advent of Large Language Models (LLMs), a potential direction is the automatic translation of regulatory constraints, expressed in natural language, into executable policies, configurations, or compliance rules.
- *Optimizing the Communication Fabric:* The evaluation results in the MAIA case study clearly identified the inter-service communication layer as the primary performance bottleneck at scale. This finding points to the need for further research on optimizing this communication fabric. Areas for improvement include the development of high-performance message brokers, more efficient messaging protocols, particularly for resource-constrained industrial edge devices, and improved serialization techniques. These improvements could significantly enhance the efficiency of communication-intensive architectures in the Compute Continuum.
- *Expanding the Concept to New Domains:* This dissertation evaluated the proposed framework in the context of Industry 4.0, a domain that remains underexplored in current literature. However, as pointed out in the taxonomy, there are also other areas where the potential of this paradigm has not been thoroughly investigated. A promising direction for future work would be to apply and adapt the CRACI architecture and its associated frameworks to other domains. Examples include large-scale scientific computing, which poses significant challenges in data orchestration and pipeline management, and healthcare, where privacy, compliance, and reliability are of critical importance. Exploring these domains would help further validate the generality, flexibility, and robustness of the proposed concepts.

Finally, the convergence of operational and information technologies is transforming the industrial landscape, leading to systems of greater scale and complexity than ever before. This dissertation argues that building such systems demands a systematic approach (such as the applied *six-stage process*). Cloud-native could offer significant benefits, but their successful adoption depends on a *polyglot approach* and a deep understanding of the underlying engineering trade-offs. Furthermore, new methods, such as the *message-based state reconstruction* for service portability and the *decentralized models for resource and service management*, can further enhance industrial systems and help realize the full potential of *Industry 4.0*.



# Bibliography

- [1] Precedence Research. *Industry 4.0 Market Size, Share and Trends 2025 to 2034*. <https://www.precedenceresearch.com/industry-4-0-market>. Report Code: 3375, last updated 23 June 2025. 2025.
- [2] Fortune Business Insights. *Internet of Things (IoT) Market Size, Share & Industry Analysis, By Component, Deployment, Enterprise Type, Industry Regional Forecast, 2024–2032*. Tech. rep. Fortune Business Insights, 2024.
- [3] MarketsandMarkets. *Industrial Edge Market Size, Share & Trends – Global Forecast to 2030*. Tech. rep. MarketsandMarkets, Feb. 2025.
- [4] W. Bank. *World Development Indicators*. Accessed: 2025-01-04. 2024. URL: <https://databank.worldbank.org/source/world-development-indicators#>.
- [5] European Comission. *Advanced manufacturing*. [https://research-and-innovation.ec.europa.eu/research-area/industrial-research-and-innovation/advanced-manufacturing\\_en](https://research-and-innovation.ec.europa.eu/research-area/industrial-research-and-innovation/advanced-manufacturing_en). Accessed: 29.12.2024. 2024.
- [6] A. Dresch, D. P. Lacerda, J. A. V. Antunes Jr, A. Dresch, D. P. Lacerda, and J. A. V. Antunes. *Design science research*. Springer, 2015.
- [7] K. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee. “A Design Science Research Methodology for Information Systems Research”. In: *Journal of Management Information Systems* 24 (2007), pp. 45–77. URL: <https://api.semanticscholar.org/CorpusID:17511997>.
- [8] J. Eekels and N. F. Roozenburg. “A methodological comparison of the structures of scientific research and engineering design: their similarities and differences”. In: *Design studies* 12.4 (1991), pp. 197–203.
- [9] S. T. March and G. F. Smith. “Design and natural science research on information technology”. In: *Decision support systems* 15.4 (1995), pp. 251–266.
- [10] J. F. Nunamaker Jr, M. Chen, and T. D. Purdin. “Systems development in information systems research”. In: *Journal of management information systems* 7.3 (1990), pp. 89–106.
- [11] A. R. Hevner, S. T. March, J. Park, and S. Ram. “Design science in information systems research”. In: *MIS quarterly* (2004), pp. 75–105.
- [12] M. J. Page, J. E. McKenzie, P. M. Bossuyt, I. Boutron, T. C. Hoffmann, C. D. Mulrow, L. Shamseer, J. M. Tetzlaff, E. A. Akl, S. E. Brennan, et al. “The PRISMA 2020 statement: an updated guideline for reporting systematic reviews”. In: *bmj* 372 (2021).
- [13] I. F. Alexander and N. Maiden. *Scenarios, stories, use cases: through the systems development life-cycle*. John Wiley & Sons, 2005.

- [14] R. J. Wieringa. *Design science methodology for information systems and software engineering*. Springer, 2014.
- [15] J. C. Licklider. “Memorandum for members and affiliates of the intergalactic computer network”. In: *M. a. A. ot IC Network (Ed.). Washington DC: KurzweilAI. ne* (1963).
- [16] S. L. Garfinkel. “The Cloud Imperative”. In: *MIT Technology Review* (Oct. 2011). Accessed: 2025-08-28. URL: <https://www.technologyreview.com/2011/10/03/190237/the-cloud-imperative/>.
- [17] S. Srinivasan. “Cloud computing evolution”. In: *Cloud Computing Basics*. Springer, 2014, pp. 1–16.
- [18] Salesforce. *The History of Salesforce: Here's the Salesforce History over the years, including newsworthy highlights and a timeline of milestones since 1999*. <https://www.salesforce.com/news/stories/the-history-of-salesforce/>. Accessed: 25.06.2024.
- [19] A. Zahariev. “Google app engine”. In: *Helsinki University of Technology* (2009), pp. 1–5.
- [20] H. Cloud. “The nist definition of cloud computing”. In: *National institute of science and technology, special publication 800.2011* (2011), p. 145.
- [21] N. Leavitt. “Is cloud computing really ready for prime time”. In: *Growth* 27.5 (2009), pp. 15–20.
- [22] L. Wang, J. Tao, M. Kunze, A. C. Castellanos, D. Kramer, and W. Karl. “Scientific cloud computing: Early definition and experience”. In: *2008 10th ieee international conference on high performance computing and communications*. Ieee. 2008, pp. 825–830.
- [23] W. Tsai, X. Bai, and Y. Huang. “Software-as-a-service (SaaS): perspectives and challenges”. In: *Science China Information Sciences* 57 (2014), pp. 1–15.
- [24] S. R. Goniwada. *Cloud native architecture and design: a handbook for modern day architecture and design with enterprise-grade examples*. Springer, 2022.
- [25] L. Qian, Z. Luo, Y. Du, and L. Guo. “Cloud computing: An overview”. In: *IEEE international conference on cloud computing*. Springer. 2009, pp. 626–631.
- [26] B. Williams. *The economics of cloud computing*. Cisco Press, 2012.
- [27] J. Ren, G. Yu, Y. He, and G. Y. Li. “Collaborative cloud and edge computing for latency minimization”. In: *IEEE Transactions on Vehicular Technology* 68.5 (2019), pp. 5031–5044.
- [28] B. Charyyev, E. Arslan, and M. H. Gunes. “Latency comparison of cloud datacenters and edge servers”. In: *GLOBECOM 2020-2020 IEEE Global Communications Conference*. IEEE. 2020, pp. 1–6.
- [29] Akamai. *20 Years of Edge Computing*. <https://www.akamai.com/blog/news/20-years-of-edge-computing>. Accessed: 25.06.2024. 2020.
- [30] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli. “Fog computing and its role in the internet of things”. In: *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. 2012, pp. 13–16.

- [31] M. Iorga, L. Feldman, R. Barton, M. Martin, N. Goren, and C. Mahmoudi. *Fog Computing Conceptual Model*. en. 2018. DOI: <https://doi.org/10.6028/NIST.SP.500-325>.
- [32] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie. “Mobile edge computing: A survey”. In: *IEEE Internet of Things Journal* 5.1 (2017), pp. 450–465.
- [33] ETSI. *Multi-access Edge Computing (MEC); Framework and Reference Architecture*. ETSI Group Specification ETSI GS MEC 003 V3.1.1. Available at ETSI website. ETSI Industry Specification Group (ISG) Multi-access Edge Computing (MEC), 2022. URL: [https://www.etsi.org/deliver/etsi\\_gs/MEC/001\\_099/003/03.01.01\\_60/gs\\_mec003v030101p.pdf](https://www.etsi.org/deliver/etsi_gs/MEC/001_099/003/03.01.01_60/gs_mec003v030101p.pdf).
- [34] T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta, and D. Sabella. “On multi-access edge computing: A survey of the emerging 5G network edge cloud architecture and orchestration”. In: *IEEE Communications Surveys & Tutorials* 19.3 (2017), pp. 1657–1681.
- [35] S. Ketu and P. K. Mishra. “Cloud, fog and mist computing in IoT: an indication of emerging opportunities”. In: *IETE Technical Review* 39.3 (2022), pp. 713–724.
- [36] P. Galambos. “Cloud, fog, and mist computing: Advanced robot applications”. In: *IEEE Systems, Man, and Cybernetics Magazine* 6.1 (2020), pp. 41–45.
- [37] Y. Wang. “Cloud-dew architecture”. In: *International Journal of Cloud Computing* 4.3 (2015), pp. 199–210.
- [38] P. P. Ray. “An introduction to dew computing: definition, concept and implications”. In: *IEEE Access* 6 (2017), pp. 723–737.
- [39] A. Rindos and Y. Wang. “Dew computing: The complementary piece of cloud computing”. In: *2016 IEEE International Conferences on Big Data and Cloud Computing (BDCloud), Social Computing and Networking (SocialCom), Sustainable Computing and Communications (SustainCom)(BDCloud-SocialCom-SustainCom)*. IEEE. 2016, pp. 15–20.
- [40] Y. Wang. “Definition and categorization of dew computing”. In: *Open Journal of Cloud Computing (OJCC)* 3.1 (2016), pp. 1–7.
- [41] S. Svorobej, M. Bendechache, F. Griesinger, and J. Domaschka. “Orchestration from the Cloud to the Edge”. In: *The Cloud-to-Thing Continuum: Opportunities and Challenges in Cloud, Fog and Edge Computing* (2020), pp. 61–77.
- [42] M. S. Aslanpour, S. S. Gill, and A. N. Toosi. “Performance evaluation metrics for cloud, fog and edge computing: A review, taxonomy, benchmarks and standards for future research”. In: *Internet of Things* 12 (2020), p. 100273.
- [43] OpenFog Consortium Architecture Working Group. *OpenFog Reference Architecture for Fog Computing*. [https://www.iiconsortium.org/pdf/OpenFog\\_Reference\\_Architecture\\_2\\_09\\_17.pdf](https://www.iiconsortium.org/pdf/OpenFog_Reference_Architecture_2_09_17.pdf). Accessed: 01.07.2024.
- [44] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies. “The case for vm-based cloudlets in mobile computing”. In: *IEEE pervasive Computing* 8.4 (2009), pp. 14–23.
- [45] M. Satyanarayanan. “A brief history of cloud offload: A personal journey from odyssey through cyber foraging to cloudlets”. In: *GetMobile: Mobile Computing and Communications* 18.4 (2015), pp. 19–23.

- [46] J. Shropshire. "Extending the cloud with fog: Security challenges & opportunities". In: (2014).
- [47] H. El-Sayed, S. Sankar, M. Prasad, D. Puthal, A. Gupta, M. Mohanty, and C.-T. Lin. "Edge of things: The big picture on the integration of edge, IoT and the cloud in a distributed computing environment". In: *ieee access* 6 (2017), pp. 1706–1717.
- [48] M. Villari, M. Fazio, S. Dustdar, O. Rana, and R. Ranjan. "Osmotic computing: A new paradigm for edge/cloud integration". In: *IEEE Cloud Computing* 3.6 (2016), pp. 76–83.
- [49] X. Wang, L. T. Yang, X. Xie, J. Jin, and M. J. Deen. "A cloud-edge computing framework for cyber-physical-social services". In: *IEEE Communications Magazine* 55.11 (2017), pp. 80–85.
- [50] J. Pan and J. McElhannon. "Future edge cloud and edge computing for internet of things applications". In: *IEEE Internet of Things Journal* 5.1 (2017), pp. 439–449.
- [51] M. Satyanarayanan. "The emergence of edge computing". In: *Computer* 50.1 (2017), pp. 30–39.
- [52] J. Ren, Y. Pan, A. Goscinski, and R. A. Beyah. "Edge computing for the internet of things". In: *IEEE Network* 32.1 (2018), pp. 6–7.
- [53] V. C. Pujol, P. K. Donta, A. Morichetta, I. Murturi, and S. Dustdar. "Edge intelligence—research opportunities for distributed computing continuum systems". In: *IEEE Internet Computing* 27.4 (2023), pp. 53–74.
- [54] D. M. A. D. Silva and R. C. Sofia. "A Discussion on Context-Awareness to Better Support the IoT Cloud/Edge Continuum". In: *IEEE Access* 8 (2020), pp. 193686–193694. DOI: 10.1109/ACCESS.2020.3032388.
- [55] J. Zhang, B. Chen, Y. Zhao, X. Cheng, and F. Hu. "Data security and privacy-preserving in edge computing paradigm: Survey and open issues". In: *IEEE access* 6 (2018), pp. 18209–18237.
- [56] B. Neha, S. K. Panda, P. K. Sahu, K. S. Sahoo, and A. H. Gandomi. "A systematic review on osmotic computing". In: *ACM Transactions on Internet of Things* 3.2 (2022), pp. 1–30.
- [57] M. Villari, A. Celesti, and M. Fazio. "Towards osmotic computing: Looking at basic principles and technologies". In: *Complex, Intelligent, and Software Intensive Systems: Proceedings of the 11th International Conference on Complex, Intelligent, and Software Intensive Systems (CISIS-2017)*. Springer. 2018, pp. 906–915.
- [58] A. Corsaro. *Fluid Computing: Unifying Cloud, Fog, and Mist Computing*. Accessed: 2024-08-17. 2016. URL: <http://www.embedded-computing.com/embedded-computing-design/fluidcomputing-unifying-cloud-fog-and-mist-computing>.
- [59] K. Fu, W. Zhang, Q. Chen, D. Zeng, and M. Guo. "Adaptive resource efficient microservice deployment in cloud-edge continuum". In: *IEEE Transactions on Parallel and Distributed Systems* 33.8 (2021), pp. 1825–1840.

- [60] A. Orive, A. Agirre, H.-L. Truong, I. Sarachaga, and M. Marcos. “Quality of Service Aware Orchestration for Cloud–Edge Continuum Applications”. In: *Sensors* 22.5 (2022). ISSN: 1424-8220. DOI: 10 . 3390 / s22051755. URL: <https://www.mdpi.com/1424-8220/22/5/1755>.
- [61] K. Fu, W. Zhang, Q. Chen, D. Zeng, X. Peng, W. Zheng, and M. Guo. “Qos-aware and resource efficient microservice deployment in cloud-edge continuum”. In: *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE. 2021, pp. 932–941.
- [62] R. Boutaba. “The cloud to things continuum: infrastructure and management”. In: *Proceedings of the 22nd International Middleware Conference: Extended Abstracts*. Middleware ’21. Virtual Event, Canada: Association for Computing Machinery, 2021, p. 7. ISBN: 9781450391924. DOI: 10 . 1145 / 3501255.3501407. URL: <https://doi.org/10.1145/3501255.3501407>.
- [63] A. Ullah, T. Kiss, J. Kovács, F. Tusa, J. Deslauriers, H. Dagdeviren, R. Arjun, and H. Hamzeh. “Orchestration in the Cloud-to-Things compute continuum: taxonomy, survey and future directions”. In: *Journal of Cloud Computing* 12.1 (2023), p. 135.
- [64] T. Lynn, J. G. Mooney, B. Lee, and P. T. Endo. “The cloud-to-thing continuum: opportunities and challenges in cloud, fog and edge computing”. In: (2020).
- [65] F. Tusa and S. Clayman. “End-to-end slices to orchestrate resources and services in the cloud-to-edge continuum”. In: *Future Generation Computer Systems* 141 (2023), pp. 473–488.
- [66] A. Ullah, H. Dagdeviren, R. C. Ariyattu, J. DesLauriers, T. Kiss, and J. Bowden. “Micado-edge: Towards an application-level orchestrator for the cloud-to-edge computing continuum”. In: *Journal of Grid Computing* 19.4 (2021), p. 47.
- [67] A. Marchese and O. Tomarchio. “Sophos: A Framework for Application Orchestration in the Cloud-to-Edge Continuum.” In: *CLOSER*. 2023, pp. 261–268.
- [68] M. Guo, L. Li, and Q. Guan. “Energy-efficient and delay-guaranteed workload allocation in IoT-edge-cloud computing systems”. In: *IEEE Access* 7 (2019), pp. 78685–78697.
- [69] Z. Hong, W. Chen, H. Huang, S. Guo, and Z. Zheng. “Multi-hop cooperative computation offloading for industrial IoT–edge–cloud computing environments”. In: *IEEE transactions on parallel and distributed systems* 30.12 (2019), pp. 2759–2774.
- [70] T. Cui, R. Yang, C. Fang, and S. Yu. “Deep reinforcement learning-based resource allocation for content distribution in IoT-edge-cloud computing environments”. In: *Symmetry* 15.1 (2023), p. 217.
- [71] C. Pahl, N. El Ioini, S. Helmer, and B. Lee. “An architecture pattern for trusted orchestration in IoT edge clouds”. In: *2018 third international conference on fog and mobile edge computing (FMEC)*. IEEE. 2018, pp. 63–70.

- [72] L. Bittencourt, R. Immich, R. Sakellariou, N. Fonseca, E. Madeira, M. Curado, L. Villas, L. DaSilva, C. Lee, and O. Rana. “The internet of things, fog and cloud continuum: Integration and challenges”. In: *Internet of Things* 3 (2018), pp. 134–155.
- [73] D. Rosendo, P. Silva, M. Simonin, A. Costan, and G. Antoniu. “E2clab: Exploring the computing continuum through repeatable, replicable and reproducible edge-to-cloud experiments”. In: *2020 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE. 2020, pp. 176–186.
- [74] S. Dustdar, V. C. Pujol, and P. K. Donta. “On distributed computing continuum systems”. In: *IEEE Transactions on Knowledge and Data Engineering* 35.4 (2022), pp. 4092–4105.
- [75] P. K. Donta, I. Murturi, V. Casamayor Pujol, B. Sedlak, and S. Dustdar. “Exploring the potential of distributed computing continuum systems”. In: *Computers* 12.10 (2023), p. 198.
- [76] P. K. Donta and S. Dustdar. “The promising role of representation learning for distributed computing continuum systems”. In: *2022 IEEE International Conference on Service-Oriented System Engineering (SOSE)*. IEEE. 2022, pp. 126–132.
- [77] P. Mantoux. *The industrial revolution in the eighteenth century: An outline of the beginnings of the modern factory system in England*. Routledge, 2013.
- [78] A. Sharma and B. J. Singh. “Evolution of industrial revolutions: a review”. In: *International Journal of Innovative Technology and Exploring Engineering* 9.11 (2020), pp. 66–73.
- [79] J. Mokyr and R. H. Strotz. “The second industrial revolution, 1870–1914”. In: *Storia dell'economia Mondiale* 21945.1 (1998), pp. 219–245.
- [80] C. Nardinelli. “Industrial revolution and the standard of living”. In: *David R. Henderson, editor* (1993).
- [81] I. Sidhu. *The Digital Revolution: How Connected Digital Innovations are Transforming Your Industry, Company & Career*. FT Press, 2015.
- [82] E. Brynjolfsson and A. McAfee. *Race against the machine: How the digital revolution is accelerating innovation, driving productivity, and irreversibly transforming employment and the economy*. Brynjolfsson and McAfee, 2012.
- [83] H. Kagermann, W. Lukas, and W. Wahlster. “Industrie 4.0: Mit dem Internet der Dinge auf dem Weg zur 4. industriellen Revolution”. In: *VDI nachrichten* 13 (2011), p. 2011.
- [84] H. Lasi, P. Fettke, H. Kemper, T. Feld, and M. Hoffmann. “Industry 4.0”. In: *Business & Information Systems Engineering* 6.4 (2014), p. 239.
- [85] H. Kagermann, J. Helbig, A. Hellinger, and W. Wahlster. *Recommendations for Implementing the strategic initiative INDUSTRIE 4.0: securing the future of German manufacturing industry; final report of the Industrie 4.0 working group*. Forschungsunion, 2013.
- [86] W. MacDougall. *Industrie 4.0: Smart manufacturing for the future*. Germany Trade & Invest, 2014.

- [87] S. J. Oks, M. Jalowski, M. Lechner, S. Mirschberger, M. Merklein, B. Vogel-Heuser, and K. M. Mösllein. "Cyber-physical systems in the context of industry 4.0: A review, categorization and outlook". In: *Information Systems Frontiers* 26.5 (2024), pp. 1731–1772.
- [88] P. Engineering. *Brave new world: Industry 4.0 technology*. [Online; accessed 17 May 2017]. 2013. URL: <http://processengineering.co.uk/article/2017121/brave-new-world-indu>.
- [89] R. Baheti and H. Gill. "Cyber-physical systems". In: *The impact of control technology* 12 (2011), pp. 161–166.
- [90] E. A. Lee. "Cyber-physical systems—are computing foundations adequate". In: *Position paper for NSF workshop on cyber-physical systems: research motivation, techniques and roadmap*. Vol. 2. 2006, pp. 1–9.
- [91] C. Witchalls and J Chambers. "The internet of things business index: A quiet revolution gathers pace". In: *The Economist Intelligence Unit*. Retrieved from <http://www.economistinsights.com/analysis/internet-things-business-index> (2013).
- [92] E. A. Lee. "Cyber physical systems: Design challenges". In: *Object Oriented Real-Time Distributed Computing (ISORC), 2008 11th IEEE International Symposium on*. IEEE. 2008, pp. 363–369.
- [93] M. Kumar, M. Vetripriya, A. Brigitte, A. Akila, and D. Keerthana. "Analysis on Internet of Things and Its Application". In: *International Journal of Scientific Research in Science* (2016).
- [94] E. Glaessgen and D. Stargel. "The digital twin paradigm for future NASA and US Air Force vehicles". In: *53rd AIAA/ASME/ASCE/AHS/ASC structures, structural dynamics and materials conference 20th AIAA/ASME/AHS adaptive structures conference 14th AIAA*. 2012, p. 1818.
- [95] F. Tao, B. Xiao, Q. Qi, J. Cheng, and P. Ji. "Digital twin modeling". In: *Journal of Manufacturing Systems* 64 (2022), pp. 372–389.
- [96] F. Pires, A. Cachada, J. Barbosa, A. P. Moreira, and P. Leitão. "Digital twin in industry 4.0: Technologies, applications and challenges". In: *2019 IEEE 17th international conference on industrial informatics (INDIN)*. Vol. 1. IEEE. 2019, pp. 721–726.
- [97] I. C. Ehie and M. A. Chilton. "Understanding the influence of IT/OT Convergence on the adoption of Internet of Things (IoT) in manufacturing organizations: An empirical investigation". In: *Computers in Industry* 115 (2020), p. 103166.
- [98] G. Murray, M. N. Johnstone, and C. Valli. "The convergence of IT and OT in critical infrastructure". In: (2017).
- [99] Cloud Native Computing Foundation. *CNCF Cloud Native Definition (DEFINITION.md)*. <https://github.com/bloomberg/cncf-toc/blob/master/DEFINITION.md>. Accessed: 2024-07-06. 2025.
- [100] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. "Service-oriented computing: a research roadmap". In: *International Journal of Cooperative Information Systems* 17.02 (2008), pp. 223–255.

- [101] N. Kratzke and P.-C. Quint. "Understanding cloud-native applications after 10 years of cloud computing—a systematic mapping study". In: *Journal of Systems and Software* 126 (2017), pp. 1–16.
- [102] M. Fowler and J. Lewis. *Microservices a definition of this new architectural term*. [Online; accessed 23 June 2017]. 2014. URL: <https://martinfowler.com/articles/microservices.html>.
- [103] A. Krylovskiy, M. Jahn, and E. Patti. "Designing a smart city internet of things platform with microservice architecture". In: *Future Internet of Things and Cloud (FiCloud), 2015 3rd International Conference on*. IEEE. 2015, pp. 25–30.
- [104] P. H. Salus. *A quarter century of UNIX*. Addison-Wesley Reading, 1994.
- [105] S. Newman. *Building microservices*. " O'Reilly Media, Inc.", 2015.
- [106] C. Richardson. *Pattern: Microservice Architecture*. [Online; accessed 17 May 2017]. 2014. URL: <http://microservices.io/patterns/microservices.html>.
- [107] T. Erl. *Service-oriented architecture: concepts, technology, and design*. Pearson Education India, 2005.
- [108] D. I. Savchenko, G. I. Radchenko, and O. Taipale. "Microservices validation: Mjolnirr platform case study". In: *Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2015 38th International Convention on*. IEEE. 2015, pp. 235–240.
- [109] S. Łaskawiec. "The evolution of Java based software architectures". In: *J. Cloud Comput. Res* 2.1 (2016), pp. 1–17.
- [110] A. Cockcroft. "State of the Art in Microservices". In: *DockerCon Europe 14* (2014).
- [111] S. Daya, N. Van Duy, K. Eati, C. M. Ferreira, D. Glozic, V. Gucer, M. Gupta, S. Joshi, V. Lampkin, M. Martins, et al. *Microservices from Theory to Practice: Creating Applications in IBM Bluemix Using the Microservices Approach*. IBM Redbooks, 2016.
- [112] D. Chappell. *Enterprise service bus*. " O'Reilly Media, Inc.", 2004.
- [113] M. Bendechache, S. Svorobej, P. Takako Endo, and T. Lynn. "Simulating resource management across the cloud-to-thing continuum: A survey and future directions". In: *Future Internet* 12.6 (2020), p. 95.
- [114] S. Moreschini, F. Pecorelli, X. Li, S. Naz, D. Hästbacka, and D. Taibi. "Cloud Continuum: The Definition". In: *IEEE Access* 10 (2022), pp. 131876–131886. DOI: [10.1109/ACCESS.2022.3229185](https://doi.org/10.1109/ACCESS.2022.3229185).
- [115] D. Rosendo, A. Costan, P. Valduriez, and G. Antoniu. "Distributed intelligence on the Edge-to-Cloud Continuum: A systematic literature review". In: *Journal of Parallel and Distributed Computing* 166 (2022), pp. 71–94.
- [116] S. K. Mishra, S. K. Sahoo, and C. K. Swain. "A Systematic Review on Federated Learning in Edge-Cloud Continuum". In: *SN Computer Science* 5.7 (2024), pp. 1–21.
- [117] C. Prigent, A. Costan, G. Antoniu, and L. Cudennec. "Enabling federated learning across the computing continuum: Systems, challenges and future directions". In: *Future Generation Computer Systems* (2024).

- [118] B. Oliveira, N. Ferry, H. Song, R. Dautov, A. Barišić, and A. R. Da Rocha. “Function-as-a-Service for the Cloud-to-Thing continuum: a Systematic Mapping Study”. In: *IoTBDS 2023–8th International Conference on Internet of Things, Big Data and Security*. 2023, pp. 82–93.
- [119] P. Gkonis, A. Giannopoulos, P. Trakadas, X. Masip-Bruin, and F. D’Andria. “A survey on IoT-edge-cloud continuum systems: status, challenges, use cases, and open issues”. In: *Future Internet* 15.12 (2023), p. 383.
- [120] R. S-Julian, I. Lacalle, R. Vaño, F. Boronat, and C. E. Palau. “Self-\* capabilities of cloud-edge nodes: A research review”. In: *Sensors* 23.6 (2023), p. 2931.
- [121] M. Gaglianese, J. Soldani, S. Forti, and A. Brogi. “Green Orchestration of Cloud-Edge Applications: State of the Art and Open Challenges”. In: *2023 IEEE International Conference on Service-Oriented System Engineering (SOSE)*. IEEE. 2023, pp. 250–261.
- [122] A. Cavacini. “What is the best database for computer science journal articles?” In: *Scientometrics* 102.3 (2015), pp. 2059–2071.
- [123] L. I. Meho and Y. Rogers. “Citation counting, citation ranking, and h-index of human-computer interaction researchers: a comparison of Scopus and Web of Science”. In: *Journal of the American Society for Information Science and Technology* 59.11 (2008), pp. 1711–1726.
- [124] W. Iqbal, J. Qadir, G. Tyson, A. N. Mian, S.-u. Hassan, and J. Crowcroft. “A bibliometric analysis of publications in computer networking research”. In: *Scientometrics* 119 (2019), pp. 1121–1155.
- [125] A. Valente, M. Holanda, A. M. Mariano, R. Furuta, and D. Da Silva. “Analysis of academic databases for literature review in the computer science education field”. In: *2022 ieee frontiers in education conference (fie)*. IEEE. 2022, pp. 1–7.
- [126] A. Valente, M. Holanda, A. M. Mariano, R. Furuta, and D. Da Silva. “Analysis of Academic Databases for Literature Review in the Computer Science Education Field”. In: *2022 IEEE Frontiers in Education Conference (FIE)*. 2022, pp. 1–7. DOI: 10.1109/FIE56618.2022.9962393.
- [127] M. Nardelli, G. Russo Russo, and V. Cardellini. “Compute Continuum: What Lies Ahead?” In: *European Conference on Parallel Processing*. Springer. 2023, pp. 5–17.
- [128] J. Gallego-Madrid, I. Bru-Santa, A. Ruiz-Rodenas, R. Sanchez-Iborra, and A. Skarmeta. “Machine learning-powered traffic processing in commodity hardware with eBPF”. In: *Computer Networks* 243 (2024), p. 110295.
- [129] S. Ahearne, A. Khalid, M. Ron, and P. Burget. “An AI factory digital twin deployed within a high performance edge architecture”. In: *2023 IEEE 31st International Conference on Network Protocols (ICNP)*. IEEE. 2023, pp. 1–6.
- [130] M. Aazam, S. ul Islam, S. T. Lone, and A. Abbas. “Cloud of things (CoT): Cloud-fog-IoT task offloading for sustainable Internet of Things”. In: *IEEE Transactions on Sustainable Computing* 7.1 (2020), pp. 87–98.

- [131] G. Di Modica, A. Galletta, L. Carnevale, A. Alkhansa, A. Costantini, D. Cesini, P. Bellavista, and M. Villari. “Orchestration of containerized applications in the Cloud continuum”. In: *2023 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops)*. IEEE. 2023, pp. 44–49.
- [132] E. Carmona-Cejudo, A. Betzler, B. Cordero, A. Pino, J. Santa, E. Egea-López, A. Ruz-Nieto, N. Perez-Palma, R. Sanchez-Iborra, A. Skarmeta, et al. “ONOFRE-3: An Architecture for the Secure and Dynamic Management of Cloud-to-Edge Resources in Connected Mobility”. In: *2023 IEEE Future Networks World Forum (FNWF)*. IEEE. 2023, pp. 1–6.
- [133] N. Mehran, D. Kimovski, and R. Prodan. “A two-sided matching model for data stream processing in the cloud–fog continuum”. In: *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. IEEE. 2021, pp. 514–524.
- [134] J. Yuan, H. Xiao, Z. Shen, T. Zhang, and J. Jin. “ELECT: Energy-efficient intelligent edge–cloud collaboration for remote IoT services”. In: *Future Generation Computer Systems* 147 (2023), pp. 179–194.
- [135] D. D. Sanchez-Gallegos, J. Gonzalez-Compean, J. Carretero, and H. Marin-Castro. “A data science pipeline synchronisation method for edge-fog-cloud continuum”. In: *Proceedings of the SC’23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis*. 2023, pp. 2053–2064.
- [136] P. Kochovski, U. Paščinski, V. Stankovski, and M. Ciglarić. “Pareto-optimised fog storage services with novel service-level agreement specification”. In: *Applied Sciences* 12.7 (2022), p. 3308.
- [137] EUCloudEdgeIOT-Building the European Cloud Edge IoT Continuum for Business and Research. <https://eucloudedgeiot.eu/>. Accessed: 29.12.2024.
- [138] C. Wohlin. “Guidelines for snowballing in systematic literature studies and a replication in software engineering”. In: *Proceedings of the 18th international conference on evaluation and assessment in software engineering*. 2014, pp. 1–10.
- [139] H. Yang and M. Tate. “A descriptive literature review and classification of cloud computing research”. In: *Communications of the Association for Information systems* 31.1 (2012), p. 2.
- [140] L. Belcastro, F. Marozzo, A. Orsino, D. Talia, and P. Trunfio. “Using the compute continuum for data analysis: Edge-cloud integration for urban mobility”. In: *2023 31st Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*. IEEE. 2023, pp. 338–344.
- [141] D. Balouek-Thomert, I. Rodero, and M. Parashar. “Evaluating policy-driven adaptation on the edge-to-cloud continuum”. In: *2021 IEEE/ACM HPC for Urgent Decision Making (UrgentHPC)*. IEEE. 2021, pp. 11–20.
- [142] R. Farahani, D. Kimovski, S. Ristov, A. Iosup, and R. Prodan. “Towards sustainable serverless processing of massive graphs on the computing continuum”. In: *Companion of the 2023 ACM/SPEC International Conference on Performance Engineering*. 2023, pp. 221–226.

- [143] A. Iosup, R. Prodan, A.-L. Varbanescu, S. Talluri, G. Magalhaes, K. Hokstam, H. Zwaan, V. Van Beek, R. Farahani, and D. Kimovski. "Graph Greenifier: Towards Sustainable and Energy-Aware Massive Graph Processing in the Computing Continuum". In: *Companion of the 2023 ACM/SPEC International Conference on Performance Engineering*. 2023, pp. 209–214.
- [144] S. Ahmad and A. Aral. "Hierarchical Federated Transfer Learning: A Multi-Cluster Approach on the Computing Continuum". In: *2023 International Conference on Machine Learning and Applications (ICMLA)*. IEEE. 2023, pp. 1163–1168.
- [145] Z. Wang, Y. Wu, and H. Wang. "Intelligent Data Transfer Technique for 6G Cyber Physical Model Enabled Teaching System". In: *Wireless Personal Communications* (2024), pp. 1–18.
- [146] A. Bueno, B. Rubio, C. Martín, and M. Díaz. "Functions as a service for distributed deep neural network inference over the cloud-to-things continuum". In: *Software: Practice and Experience* (2024).
- [147] G. Mittone, G. Malenza, M. Aldinucci, and R. Birke. "Distributed Edge Inference: an Experimental Study on Multiview Detection". In: *Proceedings of the IEEE/ACM 16th International Conference on Utility and Cloud Computing*. 2023, pp. 1–6.
- [148] A. Alqahtani, S. Alsubai, and M. Bhatia. "IoT-Edge-Cloud-Assisted Intelligent Framework for Controlling Dengue". In: *IEEE Internet of Things Journal* (2023).
- [149] S. Manoharan, K. M. Kumar, and N. Vadivelan. "A novel CNN-TLSTM approach for dengue disease identification and prevention using IoT-fog cloud architecture". In: *Neural Processing Letters* 55.2 (2023), pp. 1951–1973.
- [150] A. Pati, M. Parhi, M. Alnabhan, B. K. Pattanayak, A. K. Habboush, and M. K. Al Nawayseh. "An IoT-fog-cloud integrated framework for real-time remote cardiovascular disease diagnosis". In: *Informatics*. Vol. 10. 1. MDPI. 2023, p. 21.
- [151] D. Ciraolo, M. Fazio, R. S. Calabrò, M. Villari, and A. Celesti. "Facial expression recognition based on emotional artificial intelligence for tele-rehabilitation". In: *Biomedical Signal Processing and Control* 92 (2024), p. 106096.
- [152] L. Xiaolin, B. Cardiff, and D. John. "A Two-Stage ECG Classifier for Decentralized Inferencing Across Edge-Cloud Continuum". In: *IEEE Sensors Journal* (2024).
- [153] D. Kimovski, S. Ristov, and R. Prodan. "Decentralized machine learning for intelligent health-care systems on the computing continuum". In: *Computer* 55.10 (2022), pp. 55–65.
- [154] I. Lacalle, S. Cuñat, A. Belsa, R. Vaño, S. Raúl, P. Buschmann, J. Fontalvo-Hernández, K. Pfab, R. Bazan, F. Gramß, et al. "A Novel Approach to Self-\* Capabilities in IoT Industrial Automation Computing Continuum". In: *2023 IEEE 9th World Forum on Internet of Things (WF-IoT)*. IEEE. 2023, pp. 1–6.
- [155] T. Sarantakos, D. M. J. Gutierrez, and D. Amaxilatis. "Olive Leaf Infection Detection Using the Cloud-Edge Continuum". In: *International Symposium on Algorithmic Aspects of Cloud Computing*. Springer. 2023, pp. 25–37.

- [156] M. Amadeo, F. Cicirelli, A. Guerrieri, G. Ruggeri, G. Spezzano, and A. Vinci. “When edge intelligence meets cognitive buildings: The COGITO platform”. In: *Internet of Things* 24 (2023), p. 100908.
- [157] S. K. Mishra, M. L. Sanisetty, A. Z. Shaik, S. L. Thotakura, S. L. Aluru, and D. Puthal. “Lidar-based building damage detection in edge-cloud continuum”. In: *2023 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech)*. IEEE. 2023, pp. 0252–0257.
- [158] U Arul, R GnanaJeyaraman, A Selvakumar, S Ramesh, T. Manikandan, and G Michael. “Integration of IoT and edge cloud computing for smart microgrid energy management in VANET using machine learning”. In: *Computers and Electrical Engineering* 110 (2023), p. 108905.
- [159] N. Tzanis, D. Brodimas, K. Plakas, M. Birbas, and A. Birbas. “Optimal relocation of virtualized PDC in edge-cloud architectures under dynamic latency conditions”. In: *2022 International Conference on Electrical, Computer and Energy Technologies (ICECET)*. IEEE. 2022, pp. 1–6.
- [160] L. Belcastro, F. Marozzo, A. Orsino, D. Talia, and P. Trunfio. “Edge-cloud continuum solutions for urban mobility prediction and planning”. In: *IEEE Access* 11 (2023), pp. 38864–38874.
- [161] M. Moussa, N. Abdennahder, R. Couturier, and G. Di Marzo Serugendo. “Towards a Scalable Compute Continuum Platform Applied to Electrical Energy Forecasting”. In: *European Conference on Parallel Processing*. Springer. 2023, pp. 68–80.
- [162] M. Frincu, M. Penteliuc, and A. Spataru. “A solar radiation forecast platform spanning over the edge-cloud continuum”. In: *Electronics* 11.17 (2022), p. 2756.
- [163] J. Hong, X. Tu, S. Huang, L. Zhang, X. Huang, Z. Hu, and L. Liu. “A BiLSTM-based Industry Abnormal Electricity Consumption Warning Model in the Context of Electricity IoT Edge Cloud”. In: *2022 Asia Power and Electrical Technology Conference (APET)*. IEEE. 2022, pp. 61–66.
- [164] E. I. Fernández, A. J. Jara Valera, and J. T. Fernández Breis. “Embedded machine learning of IoT streams to promote early detection of unsafe environments”. In: *Internet of Things* 25 (2024), p. 101128. ISSN: 2542-6605. DOI: <https://doi.org/10.1016/j.iot.2024.101128>. URL: <https://www.sciencedirect.com/science/article/pii/S2542660524000702>.
- [165] K. Wang, Y. Fu, S. Zhou, R. Zhou, G. Wen, F. Zhou, L. Li, and G. Qi. “Cloud-fog-based approach for smart wildfire monitoring”. In: *Simulation Modelling Practice and Theory* 127 (2023), p. 102791.
- [166] A. Aljumah, T. A. Ahanger, and I. Ullah. “Stochastic Game Network-inspired intelligent framework for quality assessment in logistic industry”. In: *Internet of Things* 26 (2024), p. 101205.

- [167] V. Charpentier, N. Slamnik-Kriještorac, G. Landi, M. Caenepeel, O. Vasseur, and J. M. Marquez-Barja. “Paving the way towards safer and more efficient maritime industry with 5G and Beyond edge computing systems”. In: *Computer Networks* 250 (2024), p. 110499.
- [168] S. Alsubai, A. Alqahtani, A. Alanazi, and M. Bhatia. “Digital Twin-Inspired IoT-Assisted Intelligent Performance Analysis Framework for Electric Vehicles”. In: *IEEE Internet of Things Journal* (2024).
- [169] N. Nikolov, A. Solberg, R. Prodan, A. Soylu, M. Matskin, and D. Roman. “Container-Based Data Pipelines on the Computing Continuum for Remote Patient Monitoring”. In: *Computer* 56.10 (2023), pp. 40–48. DOI: 10.1109/MC.2023.3285414.
- [170] O.-C. Marcu and P. Bouvry. “In support of push-based streaming for the computing continuum”. In: *Asian Conference on Intelligent Information and Database Systems*. Springer. 2023, pp. 339–350.
- [171] S. Laso, J. Berrocal, P. Fernandez, J. M. García, J. Garcia-Alonso, J. M. Murillo, A. Ruiz-Cortés, and S. Dustdar. “Elastic Data Analytics for the Cloud-to-Things Continuum”. In: *IEEE Internet Computing* 26.6 (2022), pp. 42–49. DOI: 10.1109/MIC.2021.3138153.
- [172] D. Roman, R. Prodan, N. Nikolov, A. Soylu, M. Matskin, A. Marrella, D. Kimovski, B. Elvesæter, A. Simonet-Boulogne, G. Ledakis, et al. “Big data pipelines on the computing continuum: tapping the dark data”. In: *Computer* 55.11 (2022), pp. 74–84.
- [173] S. Afzal, Z. N. Samani, N. Mehran, C. Timmerer, and R. Prodan. “Mpec2: multilayer and pipeline video encoding on the computing continuum”. In: *2022 IEEE 21st International Symposium on Network Computing and Applications (NCA)*. Vol. 21. IEEE. 2022, pp. 181–190.
- [174] D. Kimovski, C. Bauer, N. Mehran, and R. Prodan. “Big Data Pipeline Scheduling and Adaptation on the Computing Continuum”. In: *2022 IEEE 46th Annual Computers, Software, and Applications Conference (COMPSAC)*. 2022, pp. 1153–1158. DOI: 10.1109/COMPSAC54236.2022.00181.
- [175] A. Souza, N. Ng, T. Abdelzaher, D. Towsley, and P. Shenoy. “Unlocking Efficiency: Understanding End-to-End Performance in Distributed Analytics Pipelines”. In: *MILCOM 2023 - 2023 IEEE Military Communications Conference (MILCOM)*. 2023, pp. 150–155. DOI: 10.1109/MILCOM58377.2023.10356378.
- [176] I. Zyrianoff, L. Gigli, F. Montori, L. Sciullo, C. Kamienski, and M. Di Felice. “Cache-it: A distributed architecture for proactive edge caching in heterogeneous iot scenarios”. In: *Ad Hoc Networks* 156 (2024), p. 103413.
- [177] I. Zyrianoff, L. Montecchiari, A. Trotta, L. Gigli, C. Kamienski, and M. Di Felice. “Proactive Caching in the Edge-Cloud Continuum with Federated Learning”. In: *2024 IEEE 21st Consumer Communications Networking Conference (CCNC)*. 2024, pp. 234–240. DOI: 10.1109/CCNC51664.2024.10454774.

- [178] B. Sedlak, V. C. Pujol, P. K. Donta, and S. Dustdar. “Controlling Data Gravity and Data Friction: From Metrics to Multidimensional Elasticity Strategies”. In: *2023 IEEE International Conference on Software Services Engineering (SSE)*. 2023, pp. 43–49. DOI: 10.1109/SSE60056.2023.00017.
- [179] G. Mellone, C. G. De Vita, D. D. Sánchez-Gallegos, G. Sánchez-Gallegos, C. A. Torres-Charles, J. García-Blas, J. Carretero Perez, J. Gonzalez-Compean, and G. Laccetti. “A novel approach for large-scale environmental data partitioning on cloud and on-premises storage for compute continuum applications”. In: *Concurrency and Computation: Practice and Experience* 35.25 (2023), e7893.
- [180] Z. Bakhshi, G. Rodriguez-Navas, H. Hansson, and R. Prodan. “Evaluation of Storage Placement in Computing Continuum for a Robotic Application”. In: *Journal of Grid Computing* 22.2 (2024), pp. 1–21.
- [181] I. D. Martinez-Casanueva, L. Bellido, D. González-Sánchez, and D. Lopez. “CANDIL: A federated data fabric for network analytics”. In: *Future Generation Computer Systems* 158 (2024), pp. 98–109.
- [182] P. Plebani, R. I. Kat, F. Pallas, S. Werner, G. Inches, P. Laud, R. Santiago, et al. “TEADAL: Trustworthy, Energy-Aware federated DAta Lakes along the computing continuum”. In: *CEUR Workshop Proceedings*. Vol. 3413. CEUR-WS.org. 2023, pp. 28–35.
- [183] J. Aznar-Poveda, T. Pockstaller, T. Fahringer, S. Pedratscher, and Z. Najafabadi Samani. “SDKV: A Smart and Distributed Key-Value Store for the Edge-Cloud Continuum”. In: *Proceedings of the IEEE/ACM 16th International Conference on Utility and Cloud Computing*. 2023, pp. 1–8.
- [184] D. Carrizales-Espinoza, D. D. Sanchez-Gallegos, J. Gonzalez-Compean, and J. Carretero. “StructMesh: A storage framework for serverless computing continuum”. In: *Future Generation Computer Systems* 159 (2024), pp. 353–369.
- [185] M. Jansen, L. Wagner, A. Trivedi, and A. Iosup. “Continuum: automate infrastructure deployment and benchmarking in the compute continuum”. In: *Companion of the 2023 ACM/SPEC International Conference on Performance Engineering*. 2023, pp. 181–188.
- [186] F. Lordan, D. Lezzi, and R. M. Badia. “Colony: Parallel functions as a service on the cloud-edge continuum”. In: *European Conference on Parallel Processing*. Springer. 2021, pp. 269–284.
- [187] D. Hass and J. Spillner. “Interactive Application Deployment Planning for Heterogeneous Computing Continuums”. In: *International Conference on Advanced Information Networking and Applications*. Springer. 2021, pp. 551–560.
- [188] S. Nastic. “Self-Provisioning Infrastructures for the Next Generation Serverless Computing”. In: *SN Computer Science* 5.6 (2024), p. 678.
- [189] E. Caron and R. Gracia-Tinedo. “The Nanoservices Framework: Co-Locating Microservices in the Cloud-Edge Continuum”. In: *2023 IEEE 31st International Conference on Network Protocols (ICNP)*. IEEE. 2023, pp. 1–6.

- [190] A. Carnero, C. Martín, D. R. Torres, D. Garrido, M. Díaz, and B. Rubio. “Managing and deploying distributed and deep neural models through Kafka-ML in the cloud-to-things continuum”. In: *IEEE Access* 9 (2021), pp. 125478–125495.
- [191] D. R. Torres, C. Martin, B. Rubio, and M. Diaz. “An open source framework based on Kafka-ML for Distributed DNN inference over the Cloud-to-Things continuum”. In: *Journal of Systems Architecture* 118 (2021), p. 102214.
- [192] J. Judvaitis, R. Balass, and M. Greitans. “Mobile iot-edge-cloud continuum based and devops enabled software framework”. In: *Journal of Sensor and Actuator Networks* 10.4 (2021), p. 62.
- [193] M. Vitali, J. Soldani, R. Amadini, A. Brogi, S. Forti, S. Gazza, S. Giallorenzo, P. Plebani, F. Ponce, and G. Zavattaro. “FREEDA: Failure-Resilient, Energy-aware, and Explainable Deployment of Microservice-based Applications over Cloud-IoT Infrastructures.” In: *CAiSE Research Projects Exhibition*. 2024, pp. 69–75.
- [194] I. Cohen, C. F. Chiasserini, P. Giaccone, and G. Scalosub. “Dynamic service provisioning in the edge-cloud continuum with bounded resources”. In: *IEEE/ACM Transactions on Networking* 31.6 (2023), pp. 3096–3111.
- [195] I. Cohen, P. Giaccone, and C. F. Chiasserini. “Distributed asynchronous protocol for service provisioning in the edge-cloud continuum”. In: *2023 International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*. IEEE. 2023, pp. 1–6.
- [196] P. Basaras, E. Vasilopoulos, S. Magklaris, K. V. Katsaros, and A. J. Amditis. “Experimentally Assessing Deployment Tradeoffs for AI-enabled Video Analytics Services in the 5G Compute Continuum”. In: *2023 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*. IEEE. 2023, pp. 99–104.
- [197] H. Song, R. Dautov, N. Ferry, A. Solberg, and F. Fleurey. “Model-based fleet deployment in the IoT–edge–cloud continuum”. In: *Software and Systems Modeling* 21.5 (2022), pp. 1931–1956.
- [198] A. I. Montoya-Munoz, R. A. da Silva, O. M. C. Rendon, and N. L. da Fonseca. “Reliability provisioning for fog nodes in smart farming IoT-fog-cloud continuum”. In: *Computers and Electronics in Agriculture* 200 (2022), p. 107252.
- [199] G. Galante, R. da Rosa Righi, and C. de Andrade. “Extending parallel programming patterns with adaptability features”. In: *Cluster Computing* (2024), pp. 1–22.
- [200] M. Aldinucci, R. Birke, A. Brogi, E. Carlini, M. Coppola, M. Danelutto, P. Dazzi, L. Ferrucci, S. Forti, H. Kavalionak, et al. “A Proposal for a continuum-aware programming model: from workflows to services autonomously interacting in the compute continuum”. In: *2023 IEEE 47th Annual Computers, Software, and Applications Conference (COMPSAC)*. IEEE. 2023, pp. 1852–1857.

- [201] G. Baldoni, J. Loudet, C. Guimarães, S. Nair, and A. Corsaro. “A Data Flow Programming Framework for 6G-Enabled Internet of Things Applications”. In: *2023 IEEE 9th World Forum on Internet of Things (WF-IoT)*. IEEE. 2023, pp. 1–8.
- [202] C. Marcelino and S. Nastic. “CWASI: A WebAssembly Runtime Shim for Inter-Function Communication in the Serverless Edge-Cloud Continuum”. In: *2023 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE. 2023, pp. 158–170.
- [203] J. Ménétrey, M. Pasin, P. Felber, and V. Schiavoni. “WebAssembly as a Common Layer for the Cloud-edge Continuum”. In: *Proceedings of the 2nd Workshop on Flexible Resource and Application Management on the Edge*. FRAME ’22. Minneapolis, MN, USA: Association for Computing Machinery, 2022, 3–8. ISBN: 9781450393102. DOI: 10 . 1145 / 3526059 . 3533618. URL: <https://doi.org/10.1145/3526059.3533618>.
- [204] G. R. Russo, T. Mannucci, V. Cardellini, and F. L. Presti. “Serverledge: Decentralized function-as-a-service for the edge-cloud continuum”. In: *2023 IEEE International Conference on Pervasive Computing and Communications (PerCom)*. IEEE. 2023, pp. 131–140.
- [205] C. Sicari, L. Carnevale, A. Galletta, and M. Villari. “Openwolf: A serverless workflow engine for native cloud-edge continuum”. In: *2022 IEEE intl conf on dependable, autonomic and secure computing, intl conf on pervasive intelligence and computing, intl conf on cloud and big data computing, intl conf on cyber science and technology congress (DASC/PiCom/CBDCom/CyberSciTech)*. IEEE. 2022, pp. 1–8.
- [206] M. A. Uusitalo, P. Rugeland, M. R. Boldi, E. C. Strinati, P. Demestichas, M. Ericson, G. P. Fettweis, M. C. Filippou, A. Gati, M.-H. Hamon, et al. “6G vision, value, use cases and technologies from European 6G flagship project Hexa-X”. In: *IEEE access* 9 (2021), pp. 160004–160020.
- [207] J. Kumar, J. K. Samriya, M. Bolanowski, A. Paszkiewicz, W. Pawłowski, M. Ganzha, K. Wasielewska-Michniewska, B. Solarz-Niesłuchowski, M. Paprzycki, I. L. Úbeda, et al. “Towards 6G-enabled edge-cloud continuum computing—initial assessment”. In: *International Conference on Advanced Communication and Intelligent Systems*. Springer. 2022, pp. 1–15.
- [208] L. Zheng. “Intelligent Language Acquisition Model for Online Student Interaction with Educators Using 6G-Cyber Enhanced Wireless Network”. In: *Wireless Personal Communications* (2024), pp. 1–31.
- [209] T. Laszewski, K. Arora, E. Farr, and P. Zonooz. *Cloud Native Architectures: Design high-availability and cost-effective applications for the cloud*. Packt Publishing Ltd, 2018.
- [210] E. Paraskevoulakou and D. Kyriazis. “Leveraging the serverless paradigm for realizing machine learning pipelines across the edge-cloud continuum”. In: *2021 24th Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)*. IEEE. 2021, pp. 110–117.

- [211] G. Kousiouris and D. Kyriazis. “Functionalities, Challenges and Enablers for a Generalized FaaS based Architecture as the Realizer of Cloud/Edge Continuum Interplay.” In: *CLOSER*. 2021, pp. 199–206.
- [212] I. P. Chochliouros, E. Pages-Montanera, A. Alcázar-Fernández, T. Zahariadis, T.-H. Velivassaki, C. Skianis, R. Rossini, M. Belesioti, N. Drosos, E. Bakiris, et al. “NEMO: building the next generation meta operating system”. In: *Proceedings of the 3rd Eclipse Security, AI, Architecture and Modelling Conference on Cloud to Edge Continuum*. 2023, pp. 1–9.
- [213] M. Singh, E. Fuenmayor, E. P. Hinchy, Y. Qiao, N. Murray, and D. Devine. “Digital twin: Origin to future”. In: *Applied System Innovation* 4.2 (2021), p. 36.
- [214] M. Picone, L. Bedogni, M. Pietri, M. Mamei, and F. Zambonelli. “Digital Twins & Fluid Computing in the Edge-to-Cloud Compute Continuum”. In: *2024 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops)*. IEEE. 2024, pp. 221–226.
- [215] S. Laso, L. Tore-Gálvez, J. Berrocal, C. Canal, and J. M. Murillo. “Deploying Digital Twins Over the Cloud-to-Thing Continuum”. In: *2023 IEEE Symposium on Computers and Communications (ISCC)*. IEEE. 2023, pp. 1–6.
- [216] L. Toro-Gálvez, R. García-Luque, J. Troya, C. Canal, and E. Pimentel. “Towards the integration of digital avatars in urban digital twins on the cloud-to-thing continuum”. In: *International Conference on Web Engineering*. Springer. 2023, pp. 67–74.
- [217] Y. Qu, S. Yu, L. Gao, K. Sood, and Y. Xiang. “Blockchained Dual-Asynchronous Federated Learning Services for Digital Twin Empowered Edge-Cloud Continuum”. In: *IEEE Transactions on Services Computing* (2024).
- [218] S. M. Raza, R. Minerva, N. Crespi, and M. Karech. “Definition Of Digital Twin Network Data Model in The Context of Edge-Cloud Continuum”. In: *2023 IEEE 9th International Conference on Network Softwarization (NetSoft)*. IEEE. 2023, pp. 402–407.
- [219] S. Alsubai, M. Sha, A. Alqahtani, and M. Bhatia. “Hybrid IoT-edge-cloud computing-based athlete healthcare framework: Digital twin initiative”. In: *Mobile Networks and Applications* (2023), pp. 1–20.
- [220] P. Rattanatamrong, J. Srisawat, T. Boonchoo, and J. Haga. “Development of a Digital Twin for Smart Building over Edge-Cloud Continuum”. In: *2022 IEEE 8th World Forum on Internet of Things (WF-IoT)*. IEEE. 2022, pp. 1–6.
- [221] S. S. Cranganore, V. De Maio, I. Brandic, and E. Deelman. “Paving the way to hybrid quantum-classical scientific workflows”. In: *Future Generation Computer Systems* 158 (2024), pp. 346–366.
- [222] H. T. Nguyen, M. Usman, and R. Buyya. “iQuantum: A toolkit for modeling and simulation of quantum computing environments”. In: *Software: Practice and Experience* 54.6 (2024), pp. 1141–1171.

- [223] A. Furutanpey, J. Barzen, M. Bechtold, S. Dustdar, F. Leymann, P. Raith, and F. Truger. “Architectural vision for quantum computing in the edge-cloud continuum”. In: *2023 IEEE International Conference on Quantum Software (QSW)*. IEEE. 2023, pp. 88–103.
- [224] G. Papathanail, L. Mamatas, T. Theodorou, I. Sakellariou, P. Papadimitriou, N. Filinis, D. Spatharakis, E. Fotopoulou, A. Zafeiropoulos, and S. Papavassiliou. “A virtual object stack for iot-enabled applications across the compute continuum”. In: *Proceedings of the IEEE/ACM 16th International Conference on Utility and Cloud Computing*. 2023, pp. 1–6.
- [225] D. Spatharakis, I. Dimolitsas, G. Genovese, I. Tzanettis, N. Filinis, E. Fotopoulou, C. Vassilakis, A. Zafeiropoulos, A. Iera, A. Molinaro, et al. “A Lightweight Software Stack for IoT Interoperability within the Computing Continuum”. In: *2023 19th International Conference on Distributed Computing in Smart Systems and the Internet of Things (DCOSS-IoT)*. IEEE. 2023, pp. 715–722.
- [226] K. Wasielewska-Michniewska, M. Paprzycki, and M. Ganzha. “Semantics for Resource Selection in Next Generation Internet of Things Systems”. In: *International Conference on Big Data Analytics*. Springer. 2023, pp. 289–315.
- [227] I. Korontanis, K. Tserpes, M. Pateraki, L. Blasi, J. Violos, F. Diego, E. Marin, N. Kourtellis, M. Coppola, E. Carlini, et al. “Inter-operability and orchestration in heterogeneous cloud/edge resources: The accordion vision”. In: *Proceedings of the 1st Workshop on Flexible Resource and Application Management on the Edge*. 2020, pp. 9–14.
- [228] M. Vila, V. Casamayor, S. Dustdar, and E. Teniente. “Edge-to-cloud sensing and actuation semantics in the industrial Internet of Things”. In: *Pervasive and Mobile Computing* 87 (2022), p. 101699.
- [229] S. Galantino, E. Albanese, N. Asadov, S. Braghin, F. Cappa, A. Colli-Vignarelli, A. Y. Majid, E. Marin, J. Marino, L. Moro, et al. “Building the Cloud Continuum with REAR”. In: *2024 IEEE 10th International Conference on Network Softwarization (NetSoft)*. IEEE. 2024, pp. 67–72.
- [230] E. Simion, Y. Wang, H.-l. Tai, U. Odyurt, and Z. Zhao. “Towards Seamless Serverless Computing Across an Edge-Cloud Continuum”. In: *Proceedings of the IEEE/ACM 16th International Conference on Utility and Cloud Computing*. 2023, pp. 1–6.
- [231] T. Patki, D. Ahn, D. Milroy, J.-S. Yeom, J. Garlick, M. Grondona, S. Herbein, and T. Scogland. “Fluxion: A scalable graph-based resource model for HPC scheduling challenges”. In: *Proceedings of the SC’23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis*. 2023, pp. 2077–2088.
- [232] B. Sedlak, V. C. Pujol, P. K. Donta, and S. Dustdar. “Designing reconfigurable intelligent systems with Markov blankets”. In: *International Conference on Service-Oriented Computing*. Springer. 2023, pp. 42–50.
- [233] J. Encinas, A. Rodríguez, A. Otero, and E. de la Torre. “Data-driven modeling of reconfigurable multi-accelerator systems under dynamic workloads”. In: *Microprocessors and Microsystems* 107 (2024), p. 105050.

- [234] A. Bauer, H. Pan, R. Chard, Y. Babuji, J. Bryan, D. Tiwari, I. Foster, and K. Chard. “The globus compute dataset: An open function-as-a-service dataset from the edge to the cloud”. In: *Future Generation Computer Systems* 153 (2024), pp. 558–574.
- [235] V. Kashansky, G. Radchenko, and R. Prodan. “Monte carlo approach to the computational capacities analysis of the computing continuum”. In: *International Conference on Computational Science*. Springer. 2021, pp. 779–793.
- [236] A. Márkus and A. Kertész. “Modelling Energy Consumption of IoT Devices in DISSECT-CF-Fog.” In: CLOSER. 2021, pp. 320–327.
- [237] A. Markus, M. Biro, G. Kecskemeti, and A. Kertesz. “Actuator behaviour modelling in IoT-Fog-Cloud simulation”. In: *PeerJ Computer Science* 7 (2021), e651.
- [238] D. Balouek-Thomert, P. Silva, K. Fauvel, A. Costan, G. Antoniu, and M. Parashar. “MDSC: modelling distributed stream processing across the edge-to-cloud continuum”. In: *Proceedings of the 14th IEEE/ACM International Conference on Utility and Cloud Computing Companion*. 2021, pp. 1–6.
- [239] I. Bernabé, A. Fernández, H. Billhardt, and S. Ossowski. “Towards semantic modelling of the edge-cloud continuum”. In: *International Conference on Practical Applications of Agents and Multi-Agent Systems*. Springer. 2022, pp. 71–82.
- [240] J. A. Barriga, J. M. Chaves-González, A. Barriga, P. Alonso, and P. J. Clemente. “Simulate IoT Towards the Cloud-to-Thing Continuum Paradigm for Task Scheduling Assessments.” In: *J. Object Technol.* 22.1 (2023), pp. 1–31.
- [241] S. Baneshi, A.-L. Varbanescu, A. Pathania, B. Akesson, and A. Pimentel. “Estimating the energy consumption of applications in the computing continuum with ifogsim”. In: *International Conference on High Performance Computing*. Springer. 2023, pp. 234–249.
- [242] K. Alwasel, D. N. Jha, F. Habeeb, U. Demirbaga, O. Rana, T. Baker, S. Dustdar, M. Villari, P. James, E. Solaiman, et al. “IoTSim-Osmosis: A framework for modeling and simulating IoT applications over an edge-cloud continuum”. In: *Journal of Systems Architecture* 116 (2021), p. 101956.
- [243] P. Raith, T. Rausch, A. Furutanpey, and S. Dustdar. “faas-sim: A trace-driven simulation framework for serverless edge computing platforms”. In: *Software: Practice and Experience* 53.12 (2023), pp. 2327–2361.
- [244] A. Matricardi, A. Bocci, S. Forti, and A. Brogi. “Simulating FaaS Orchestrations In The Cloud-Edge Continuum”. In: *Proceedings of the 3rd Workshop on Flexible Resource and Application Management on the Edge*. 2023, pp. 19–26.
- [245] W. Valdez, H. Baniata, A. Markus, and A. Kertesz. “Towards a Simulation as a Service Platform for the Cloud-to-Things Continuum”. In: *European Conference on Parallel Processing*. Springer. 2023, pp. 65–75.
- [246] A. Markus, M. Biro, and A. Kertesz. “Analysing IoT Applications with DISSECT-CF-Fog in Simulated Fog and Cloud Environments”. In: (2021).

- [247] F. Habeeb, K. Alwasel, A. Noor, D. N. Jha, D. AlQattan, Y. Li, G. S. Aujla, T. Szydlo, and R. Ranjan. “Dynamic bandwidth slicing for time-critical IoT data streams in the edge-cloud continuum”. In: *IEEE Transactions on Industrial Informatics* 18.11 (2022), pp. 8017–8026.
- [248] D. B. Maciel, E. P. Neto, K. B. Costa, M. P. Lima, V. G. Lopes, A. V. Neto, F. S. D. Silva, and S. C. Sampaio. “Cloud-network slicing MANO towards an efficient IoT-cloud continuum”. In: *Journal of Grid Computing* 19 (2021), pp. 1–25.
- [249] F. Finocchio, N. Tonci, and M. Torquati. “MTCL: a multi-transport communication library”. In: *European Conference on Parallel Processing*. Springer. 2023, pp. 55–67.
- [250] K. Horvath, D. Kimovski, C. Uran, H. Wöllik, and R. Prodan. “MESDD: A Distributed Geofence-Based Discovery Method for the Computing Continuum”. In: *European Conference on Parallel Processing*. Springer. 2023, pp. 125–138.
- [251] H.-L. Truong and K. Magoutis. “Robustness via Elasticity Accelerators for the IoT-Edge-Cloud Continuum”. In: *2022 IEEE/ACM 15th International Conference on Utility and Cloud Computing (UCC)*. IEEE. 2022, pp. 291–296.
- [252] Y. Boloorchi and S. Azizi. “An IoT-Fog-Cloud Framework for Demand Side Management in Smart Grid”. In: *2021 5th International Conference on Internet of Things and Applications (IoT)*. IEEE. 2021, pp. 1–6.
- [253] M. Zanella, F. Sciamanna, and W. Fornaciari. “BarMan: A run-time management framework in the resource continuum”. In: *Sustainable Computing: Informatics and Systems* 35 (2022), p. 100663.
- [254] S. Volpert, G. Eisenhart, and J. Domaschka. “Holistic Life-Cycle Management of Cloud-To-Thing Compute Entities”. In: *2023 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops)*. IEEE. 2023, pp. 50–55.
- [255] Z. Houmani, D. Balouek-Thomert, E. Caron, and M. Parashar. “Enabling microservices management for Deep Learning applications across the Edge-Cloud Continuum”. In: *2021 IEEE 33rd International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*. IEEE. 2021, pp. 137–146.
- [256] Z. Shamsa, A. Rezaee, S. Adabi, A. M. Rahimabadi, and A. M. Rahmani. “A distributed load balancing method for IoT/Fog/Cloud environments with volatile resource support”. In: *Cluster Computing* (2024), pp. 1–40.
- [257] M. Vijarania, S. Gupta, A. Agrawal, M. O. Adigun, S. A. Ajagbe, and J. B. Awotunde. “Energy efficient load-balancing mechanism in integrated IoT-fog-cloud environment”. In: *Electronics* 12.11 (2023), p. 2543.
- [258] N. Grigoropoulos and S. Lalis. “Fractus: Orchestration of Distributed Applications in the Drone-Edge-Cloud Continuum”. In: *2022 IEEE 46th Annual Computers, Software, and Applications Conference (COMPSAC)*. IEEE. 2022, pp. 838–848.

- [259] J. M. Kunkel, C. Boehme, J. Decker, F. Magugliani, D. Pleiter, B. Koller, K. Sivalingam, S. Plana, A. Nikolov, M. Soyturk, et al. “DECICE: Device–Edge–Cloud Intelligent Collaboration Framework”. In: *Proceedings of the 20th ACM International Conference on Computing Frontiers*. 2023, pp. 266–271.
- [260] A. Zafeiropoulos, N. Filinis, E. Fotopoulou, and S. Papavassiliou. “AI-Assisted Synergetic Orchestration Mechanisms for Autoscaling in Computing Continuum Systems”. In: *IEEE Communications Magazine* (2024).
- [261] G. Bisicchia, S. Forti, E. Pimentel, and A. Brogi. “Continuous QoS-compliant orchestration in the Cloud–Edge continuum”. In: *Software: Practice and Experience* (2023).
- [262] A. Spataru and J. Aperribay. “Configuration of Container Deployments on the Compute Continuum using Alien4Cloud”. In: *Scalable Computing: Practice and Experience* 25.1 (2024), pp. 607–613.
- [263] M. Caballer, G. Moltó, A. Calatrava, and I. Blanquer. “Infrastructure manager: A TOSCA-based orchestrator for the computing continuum”. In: *Journal of Grid Computing* 21.3 (2023), p. 51.
- [264] F. Lumpp, F. Barchi, A. Acquaviva, and N. Bombieri. “On the Containerization and Orchestration of RISC-V architectures for Edge–Cloud computing”. In: *Proceedings of the 3rd Eclipse Security, AI, Architecture and Modelling Conference on Cloud to Edge Continuum*. 2023, pp. 21–28.
- [265] D. Hass and J. Spillner. “Workload deployment and configuration reconciliation at scale in kubernetes-based edge–cloud continuums”. In: *2022 21st International Symposium on Parallel and Distributed Computing (ISPDC)*. IEEE. 2022, pp. 121–128.
- [266] C. Sicari, D. Balouek, M. Parashar, and M. Villari. “Event-Driven FaaS Workflows for Enabling IoT Data Processing at the Cloud Edge Continuum”. In: *Proceedings of the IEEE/ACM 16th International Conference on Utility and Cloud Computing*. 2023, pp. 1–10.
- [267] I. Čilić, I. P. Žarko, and M. Kušek. “Towards service orchestration for the cloud-to-thing continuum”. In: *2021 6th International Conference on Smart and Sustainable Technologies (SpliTech)*. IEEE. 2021, pp. 01–07.
- [268] J. Kennedy, V. Sharma, B. Varghese, and C. Reaño. “Multi-tier GPU virtualization for deep learning in cloud-edge systems”. In: *IEEE Transactions on Parallel and Distributed Systems* 34.7 (2023), pp. 2107–2123.
- [269] F. Filippini, R. Cavolini, D. Ardagna, R. Lancellotti, G. Russo Russo, V. Cardellini, and F. Lo Presti. “FIGARO: reinForcement learnInG mAnagement acRoss the computing cOntinuum”. In: *Proceedings of the IEEE/ACM 16th International Conference on Utility and Cloud Computing*. 2023, pp. 1–8.
- [270] P. Soumplis, G. Kontos, A. Kretsis, P. Kokkinos, A. Nanos, and E. Varvarigos. “Security-Aware Resource Allocation in the Edge–Cloud Continuum”. In: *2023 IEEE 12th International Conference on Cloud Networking (CloudNet)*. IEEE. 2023, pp. 161–169.

- [271] D. Balouek and H. Couillon. “Dynamic Adaptation of Urgent Applications in the Edge-to-Cloud Continuum”. In: *European Conference on Parallel Processing*. Springer. 2023, pp. 189–201.
- [272] G. Manogaran and B. S. Rawal. “An efficient resource allocation scheme with optimal node placement in IoT-fog-cloud architecture”. In: *IEEE Sensors Journal* 21.22 (2021), pp. 25106–25113.
- [273] G. R. Russo, V. Cardellini, and F. L. Presti. “A framework for offloading and migration of serverless functions in the Edge–Cloud Continuum”. In: *Pervasive and Mobile Computing* 100 (2024), p. 101915.
- [274] R. C. Sofia, J. Salomon, S. Ferlin-Reiter, L. Garcés-Erice, P. Urbanetz, H. Mueller, R. Touma, A. Espinosa, L. M. Contreras, V. Theodorou, et al. “A Framework for Cognitive, Decentralized Container Orchestration”. In: *IEEE Access* (2024).
- [275] M. Loaiza, C. Savaglio, R. Gravina, D. Chatzopoulos, and S. Lalis. “Agents in the Computing Continuum: the MLSysOps Perspective”. In: *Proceedings of the 2023 INTERNATIONAL CONFERENCE ON EMBEDDED WIRELESS SYSTEMS AND NETWORKS*. 2023, pp. 321–326.
- [276] B. Sedlak, V. C. Pujol, P. K. Donta, and S. Dustdar. “Equilibrium in the Computing Continuum through Active Inference”. In: *Future Generation Computer Systems* (2024).
- [277] A. Zafeiropoulos, E. Fotopoulou, C. Vassilakis, I. Tzanettis, C. Lombardo, A. Carrega, and R. Bruschi. “Intent-driven distributed applications management over compute and network resources in the computing continuum”. In: *2023 19th International Conference on Distributed Computing in Smart Systems and the Internet of Things (DCOSS-IoT)*. IEEE. 2023, pp. 429–436.
- [278] M. Zaccarini, B. Cantelli, M. Fazio, W. Fornaciari, F. Poltronieri, C. Stefanelli, and M. Tortonesi. “VOICE: Value-of-Information for Compute Continuum Ecosystems”. In: *2024 27th Conference on Innovation in Clouds, Internet and Networks (ICIN)*. IEEE. 2024, pp. 73–80.
- [279] R. Sousa, L. Nogueira, F. Rodrigues, and L. M. Pinho. “Global Resource Management in the ELASTIC Architecture”. In: *2022 IEEE 5th International Conference on Industrial Cyber-Physical Systems (ICPS)*. IEEE. 2022, pp. 01–06.
- [280] B. Mutichiro and Y. Kim. “User preference-based QoS-aware service function placement in IoT-Edge cloud”. In: *International Journal of Distributed Sensor Networks* 17.5 (2021), p. 15501477211019912.
- [281] D. Kimovski, R. Mathá, J. Hammer, N. Mehran, H. Hellwagner, and R. Prodan. “Cloud, fog, or edge: Where to compute?” In: *IEEE Internet Computing* 25.4 (2021), pp. 30–36.
- [282] Z. N. Samani, N. Mehran, D. Kimovski, and R. Prodan. “Proactive SLA-aware Application Placement in the Computing Continuum”. In: *2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE. 2023, pp. 468–479.

- [283] A. Markus, J. D. Dombi, and A. Kertesz. “Location-aware task allocation strategies for IoT-fog-cloud environments”. In: *2021 29th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*. IEEE. 2021, pp. 185–192.
- [284] K. Wang, J. Jin, Y. Yang, T. Zhang, A. Nallanathan, C. Tellambura, and B. Jabbari. “Task offloading with multi-tier computing resources in next generation wireless networks”. In: *IEEE Journal on Selected Areas in Communications 41.2* (2022), pp. 306–319.
- [285] H. Chahed, M. Usman, A. Chatterjee, F. Bayram, R. Chaudhary, A. Brunstrom, J. Taheri, B. S. Ahmed, and A. Kassler. “AIDA—A holistic AI-driven networking and processing framework for industrial IoT applications”. In: *Internet of Things 22* (2023), p. 100805.
- [286] B. Miloradović and A. V. Papadopoulos. “Multi-Criteria Optimization of Application Offloading in the Edge-to-Cloud Continuum”. In: *2023 62nd IEEE Conference on Decision and Control (CDC)*. IEEE. 2023, pp. 4917–4923.
- [287] O. M. Al-Tuhafi and E. H. Al-Hemairy. “Adaptive Thresholds for Task Offloading in IoT-Edge-Cloud Networks”. In: *2023 International Conference On Cyber Management And Engineering (CyMaEn)*. IEEE. 2023, pp. 379–385.
- [288] M. Diamanti, E. E. Tsipropoulou, and S. Papavassiliou. “An incentivization mechanism for green computing continuum of delay-tolerant tasks”. In: *ICC 2022-IEEE International Conference on Communications*. IEEE. 2022, pp. 3538–3543.
- [289] J. C. Vicenzi, G. Korol, M. G. Jordan, W. O. de Morais, H. Ali, E. P. De Freitas, M. B. Rutzig, and A. C. S. Beck. “Dynamic Offloading for Improved Performance and Energy Efficiency in Heterogeneous IoT-Edge-Cloud Continuum”. In: *2023 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE. 2023, pp. 1–6.
- [290] D. Saito, S. Hu, and Y. Sato. “A Microservice Scheduler for Heterogeneous Resources on Edge-Cloud Computing Continuum”. In: *2024 IEEE Symposium in Low-Power and High-Speed Chips (COOL CHIPS)*. IEEE. 2024, pp. 1–6.
- [291] B. Mutichiro, M.-N. Tran, and Y.-H. Kim. “QoS-based service-time scheduling in the IoT-edge cloud”. In: *Sensors 21.17* (2021), p. 5797.
- [292] R. Kumar, M. Baughman, R. Chard, Z. Li, Y. Babuji, I. Foster, and K. Chard. “Coding the computing continuum: Fluid function execution in heterogeneous computing environments”. In: *2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE. 2021, pp. 66–75.
- [293] T. Pusztai, S. Nastic, P. Raith, S. Dustdar, D. Vij, and Y. Xiong. “Vela: A 3-Phase Distributed Scheduler for the Edge-Cloud Continuum”. In: *2023 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE. 2023, pp. 161–172.
- [294] S. K. Panda, A. Dhiman, and P. Bhuriya. “Efficient real-time task-based scheduling algorithms for iot-fog-cloud architecture”. In: *2023 14th International Conference on Computing Communication and Networking Technologies (ICCCNT)*. IEEE. 2023, pp. 1–7.

- [295] G. De Palma, S. Giallorenzo, J. Mauro, M. Trentin, and G. Zavattaro. “A declarative approach to topology-aware serverless function-execution scheduling”. In: *2022 IEEE International Conference on Web Services (ICWS)*. IEEE. 2022, pp. 337–342.
- [296] K. M. Matrouk and A. D. Matrouk. “Mobility aware-task scheduling and virtual fog for offloading in IoT-fog-cloud environment”. In: *Wireless Personal Communications* 130.2 (2023), pp. 801–836.
- [297] H. Platter and D. Kimovski. “Preemptive Online Scheduling in the Computing Continuum”. In: *2022 IEEE/ACM 15th International Conference on Utility and Cloud Computing (UCC)*. IEEE. 2022, pp. 321–326.
- [298] A. Orive, A. Agirre, H.-L. Truong, I. Sarachaga, and M. Marcos. “Quality of service aware orchestration for cloud–edge continuum applications”. In: *Sensors* 22.5 (2022), p. 1755.
- [299] A. Taghinezhad-Niar and J. Taheri. “Security, reliability, cost, and energy-aware scheduling of real-time workflows in compute-continuum environments”. In: *IEEE Transactions on Cloud Computing* (2024).
- [300] D. Rosendo, M. Mattoso, A. Costan, R. Souza, D. Pina, P. Valduriez, and G. Antoniu. “ProvLight: Efficient workflow provenance capture on the edge-to-cloud continuum”. In: *2023 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE. 2023, pp. 221–233.
- [301] P. Dazzi, V. Grossi, and R. Trasarti. “Workflows for Bringing Data Science on the Cloud/Edge Computing Continuum.” In: *SEBD*. 2022, pp. 125–132.
- [302] H. Yi. “Efficient learning-driven data transmission algorithm for cloud-to-thing continuum”. In: *Journal of King Saud University-Computer and Information Sciences* 35.10 (2023), p. 101834.
- [303] P. Sowiński, K. Wasielewska-Michniewska, M. Ganzha, W. Pawowski, P. Szmeja, and M. Paprzycki. “Efficient RDF Streaming for the Edge-Cloud Continuum”. In: *2022 IEEE 8th World Forum on Internet of Things (WF-IoT)*. 2022, pp. 1–8. DOI: 10.1109/WF-IoT54382.2022.10152225.
- [304] N. E. Nwogbaga, R. Latip, L. S. Affendey, and A. R. A. Rahiman. “Investigation into the effect of data reduction in offloadable task for distributed IoT-fog-cloud computing”. In: *Journal of Cloud Computing* 10 (2021), pp. 1–12.
- [305] V. Jafari and M. H. Rezvani. “Joint optimization of energy consumption and time delay in IoT-fog-cloud computing environments using NSGA-II metaheuristic algorithm”. In: *Journal of Ambient Intelligence and Humanized Computing* 14.3 (2023), pp. 1675–1698.
- [306] H. Reffad and A. Alti. “Semantic-based multi-objective optimization for qos and energy efficiency in iot, fog, and cloud erp using dynamic cooperative nsga-ii”. In: *Applied Sciences* 13.8 (2023), p. 5218.
- [307] J. Zhang, Y. Deng, H. Zhang, and Y. Fang. “Task Scheduling and Resource Allocation for Compressed Sensing in IoT-Edge-Cloud Systems”. In: *GLOBECOM 2023-2023 IEEE Global Communications Conference*. IEEE. 2023, pp. 3795–3800.

- [308] L. Angelelli, A. A. Da Silva, Y. Georgiou, M. Mercier, G. Mounié, and D. Trystram. “Towards a multi-objective scheduling policy for serverless-based edge-cloud continuum”. In: *2023 IEEE/ACM 23rd International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. IEEE. 2023, pp. 485–497.
- [309] N. Filinis, I. Tzanettis, D. Spatharakis, E. Fotopoulou, I. Dimolitsas, A. Zafeiropoulos, C. Vassilakis, and S. Papavassiliou. “Intent-driven orchestration of serverless applications in the computing continuum”. In: *Future Generation Computer Systems* 154 (2024), pp. 72–86.
- [310] B. Cai, X. Wang, B. Wang, M. Yang, Y. Guo, and Q. Guo. “A self-stabilizing and auto-provisioning orchestration for microservices in edge-cloud continuum”. In: *Computer Networks* 242 (2024), p. 110279.
- [311] F. Smirnov, C. Engelhardt, J. Mittelberger, B. Pourmohseni, and T. Fahringer. “Apollo: Towards an efficient distributed orchestration of serverless function compositions in the cloud-edge continuum”. In: *Proceedings of the 14th IEEE/ACM International Conference on Utility and Cloud Computing*. 2021, pp. 1–10.
- [312] P. Rodis and P. Papadimitriou. “Unsupervised Deep Learning for Distributed Service Function Chain Embedding”. In: *IEEE Access* (2023).
- [313] A. Nanos, A. Kretsis, C. Mainas, G. Ntouskos, A. Ferikoglou, D. Danopoulos, A. Kokkinis, D. Masouros, K. Siozios, P. Soumplis, et al. “Hardware-Accelerated FaaS for the Edge-Cloud Continuum”. In: *2023 IEEE 31st International Conference on Network Protocols (ICNP)*. IEEE. 2023, pp. 1–6.
- [314] S. K. Srichandan, S. K. Majhi, S. Jena, K. Mishra, and D. C. Rao. “Efficient latency-and-energy-aware IoT-fog-cloud task orchestration: novel algorithmic approach with enhanced arithmetic optimization and pattern search”. In: *International Journal of Information Technology* 16.5 (2024), pp. 3311–3324.
- [315] I. Z. Yakubu and M Murali. “An Efficient IoT-Fog-Cloud Resource Allocation Framework Based on Two-Stage Approach”. In: *IEEE Access* (2024).
- [316] I. Z. Yakubu and M Murali. “An efficient meta-heuristic resource allocation with load balancing in IoT-Fog-cloud computing environment”. In: *Journal of Ambient Intelligence and Humanized Computing* 14.3 (2023), pp. 2981–2992.
- [317] P. Soumplis, G. Kontos, P. Kokkinos, A. Kretsis, S. Barrachina-Muñoz, R. Nikbakht, J. Baranda, M. Payaró, J. Mangues-Bafalluy, and E. Varvarigos. “Performance Optimization Across the Edge-Cloud Continuum: A Multi-agent Rollout Approach for Cloud-Native Application Workload Placement”. In: *SN Computer Science* 5.3 (2024), p. 318.
- [318] C. Puliafito, C. Cicconetti, M. Conti, E. Mingozzi, and A. Passarella. “Balancing local vs. remote state allocation for micro-services in the cloud–edge continuum”. In: *Pervasive and Mobile Computing* 93 (2023), p. 101808.
- [319] D. Wang, H. Shen, and H. Tian. “Resource Configuration for Cross-Server Deployment of Application-Oriented Microservices in Cloud-Edge Continuum with SLO Constraints”. In: *2023 IEEE 29th International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE. 2023, pp. 2655–2662.

- [320] I. Sartzetakis, P. Soumplis, P. Pantazopoulos, K. V. Katsaros, V. Sourlas, and E. M. Varvarigos. “Resource allocation for distributed machine learning at the edge-cloud continuum”. In: *ICC 2022-IEEE International Conference on Communications*. IEEE. 2022, pp. 5017–5022.
- [321] C. Bauer, N. Mehran, R. Prodan, and D. Kimovski. “Machine Learning Based Resource Utilization Prediction in the Computing Continuum”. In: *2023 IEEE 28th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*. IEEE. 2023, pp. 219–224.
- [322] D. Gabi, N. M. Dankolo, A. A. Muslim, A. Abraham, M. U. Joda, A. Zainal, and Z. Zakaria. “Dynamic scheduling of heterogeneous resources across mobile edge-cloud continuum using fruit fly-based simulated annealing optimization scheme”. In: *Neural Computing and Applications* 34.16 (2022), pp. 14085–14105.
- [323] O. Almurshed, S. Meshoul, A. Muftah, A. K. Kaushal, O. Almoghamis, I. Petri, N. Auluck, and O. Rana. “A Framework for Performance Optimization of Internet of Things Applications”. In: *European Conference on Parallel Processing*. Springer. 2023, pp. 165–176.
- [324] H. Sedghani, F. Filippini, and D. Ardagna. “A randomized greedy method for AI applications component placement and resource selection in computing continua”. In: *2021 IEEE International Conference on Joint Cloud Computing (JCC)*. IEEE. 2021, pp. 65–70.
- [325] H. Sedghani, F. Filippini, and D. Ardagna. “A random greedy based design time tool for AI applications component placement and resource selection in computing continua”. In: *2021 IEEE International Conference on Edge Computing (EDGE)*. IEEE. 2021, pp. 32–40.
- [326] P. Raith, T. Rausch, S. Dustdar, F. Rossi, V. Cardellini, and R. Ranjan. “Mobility-aware serverless function adaptations across the edge-cloud continuum”. In: *2022 IEEE/ACM 15th International Conference on Utility and Cloud Computing (UCC)*. IEEE. 2022, pp. 123–132.
- [327] R. Brännvall, T. Stark, J. Gustafsson, M. Eriksson, and J. Summers. “Cost Optimization for the Edge–Cloud Continuum by Energy-Aware Workload Placement”. In: *Companion Proceedings of the 14th ACM International Conference on Future Energy Systems*. 2023, pp. 79–84.
- [328] S. Rac and M. Brorsson. “Cost-aware service placement and scheduling in the Edge–Cloud Continuum”. In: *ACM Transactions on Architecture and Code Optimization* 21.2 (2024), pp. 1–24.
- [329] G. M. Almeida, G. Z. Bruno, A. Huff, M. Hiltunen, E. P. Duarte, C. B. Both, and K. V. Cardoso. “RIC-O: Efficient placement of a disaggregated and distributed RAN Intelligent Controller with dynamic clustering of radio nodes”. In: *IEEE Journal on Selected Areas in Communications* (2023).
- [330] C. Guerrero, I. Lera, and C. Juiz. “Distributed genetic algorithm for application placement in the compute continuum leveraging infrastructure nodes for optimization”. In: *Future Generation Computer Systems* (2024).

- [331] F. Filippini, H. Sedghani, and D. Ardagna. “SPACE4AI-R: a runtime management tool for AI applications component placement and resource scaling in computing continua”. In: *Proceedings of the IEEE/ACM 16th International Conference on Utility and Cloud Computing*. 2023, pp. 1–7.
- [332] S. Manihar, R. Patel, and S. Agrawal. “Learning Based Task Placement Algorithm in the IoT Fog-Cloud Environment”. In: *International Journal of Computer Networks and Applications (IJCNA)* 8.5 (2021), pp. 549–565.
- [333] G. Kontos, P. Soumplis, P. C. Kokkinos, and E. A. Varvarigos. “Cloud-Native Applications’ Workload Placement over the Edge-Cloud Continuum.” In: *CLOSER*. 2023, pp. 57–66.
- [334] K. Sheshadri and J Lakshmi. “Qos aware faas for heterogeneous edge-cloud continuum”. In: *2022 IEEE 15th International Conference on Cloud Computing (CLOUD)*. IEEE. 2022, pp. 70–80.
- [335] K. Li and S. Nastic. “AttentionFunc: Balancing FaaS compute across edge-cloud continuum with reinforcement learning”. In: *Proceedings of the 13th International Conference on the Internet of Things*. 2023, pp. 25–32.
- [336] S. Taghizadeh, H. Elbiaze, R. Glitho, and W. Ajib. “CASMaT: Characteristic-Aware SFC Mapping for Telesurgery Systems in Cloud-Edge Continuum”. In: *IEEE Open Journal of the Communications Society* (2024).
- [337] T. Di Riccio, J. Massa, S. Forti, and A. Brogi. “Sustainable placement of VNF chains in Intent-based Networking”. In: *Proceedings of the IEEE/ACM 16th International Conference on Utility and Cloud Computing*. 2023, pp. 1–10.
- [338] Y. Mao, X. Shang, Y. Liu, and Y. Yang. “Joint Virtual Network Function Placement and Flow Routing in Edge-Cloud Continuum”. In: *IEEE Transactions on Computers* (2023).
- [339] S. O. Ali, H. Elbiaze, R. Glitho, and W. Ajib. “CaMP-INC: Components-aware Microservices Placement for In-Network Computing Cloud-Edge Continuum”. In: *GLOBECOM 2022-2022 IEEE Global Communications Conference*. IEEE. 2022, pp. 2116–2121.
- [340] D. Kimovski, N. Mehran, C. E. Kerth, and R. Prodan. “Mobility-Aware IoT Application Placement in the Cloud–Edge Continuum”. In: *IEEE Transactions on Services Computing* 15.6 (2021), pp. 3358–3371.
- [341] G. Yu, P. Chen, Z. Zheng, J. Zhang, X. Li, and Z. He. “Faasdeler: Cost-efficient and qos-aware function delivery in computing continuum”. In: *IEEE Transactions on Services Computing* 16.5 (2023), pp. 3332–3347.
- [342] S. Abdi, M. Ashjaei, and S. Mubeen. “Task Offloading in Edge-Cloud Computing Using a Q-Learning Algorithm.” In: *CLOSER*. 2024, pp. 159–166.
- [343] A. Heidari, N. J. Navimipour, M. A. J. Jamali, and S. Akbarpour. “A green, secure, and deep intelligent method for dynamic IoT-edge-cloud offloading scenarios”. In: *Sustainable Computing: Informatics and Systems* 38 (2023), p. 100859.
- [344] R. Naseri, M. Abdollahi Azgomi, and A. Naghash Asadi. “Task offloading in an optimized power-performance manner”. In: *International Journal of Communication Systems* 37.5 (2024), e5686.

- [345] G. Nieto, I. de la Iglesia, U. Lopez-Novoa, and C. Perfecto. “Deep Reinforcement Learning techniques for dynamic task offloading in the 5G edge-cloud continuum”. In: *Journal of Cloud Computing* 13.1 (2024), p. 94.
- [346] A. Robles-Enciso and A. F. Skarmeta. “Task Offloading in Computing Continuum Using Collaborative Reinforcement Learning”. In: *Global IoT Summit*. Springer, 2022, pp. 82–95.
- [347] Z. Cheng, Z. Gao, M. Liwang, L. Huang, X. Du, and M. Guizani. “Intelligent task offloading and energy allocation in the UAV-aided mobile edge-cloud continuum”. In: *Ieee Network* 35.5 (2021), pp. 42–49.
- [348] G. P. Mattia and R. Beraldì. “Leveraging Reinforcement Learning for online scheduling of real-time tasks in the Edge/Fog-to-Cloud computing continuum”. In: *2021 IEEE 20th International Symposium on Network Computing and Applications (NCA)*. IEEE. 2021, pp. 1–9.
- [349] N. M. D. Dankolo, N. H. M. Radzi, N. H. Mustaffa, M. S. Talib, Z. M. Yunos, and D. Gabi. “Efficient Task Scheduling Approach in Edge-Cloud Continuum based on Flower Pollination and Improved Shuffled Frog Leaping Algorithm”. In: *Baghdad Science Journal* 21.2 (SI) (2024), pp. 0740–0740.
- [350] T. Salehnia, A. Seyfollahi, S. Raziani, A. Noori, A. Ghaffari, A. R. Alsoud, and L. Abualigah. “An optimal task scheduling method in IoT-Fog-Cloud network using multi-objective moth-flame algorithm”. In: *Multimedia Tools and Applications* 83.12 (2024), pp. 34351–34372.
- [351] P. Singh and R. Singh. “Energy-efficient delay-aware task offloading in fog-cloud computing system for IoT sensor applications”. In: *Journal of Network and Systems Management* 30.1 (2022), p. 14.
- [352] M. Abbasi, E. Mohammadi-Pasand, and M. R. Khosravi. “Intelligent workload allocation in IoT–Fog–cloud architecture towards mobile edge computing”. In: *Computer Communications* 169 (2021), pp. 71–80.
- [353] M. Maashi, E. Alabdulkreem, M. Maray, K. Shankar, A. A. Darem, A. Alzahrani, and I. Yaseen. “Elevating Survivability in Next-Gen IoT-Fog-Cloud Networks: Scheduling Optimization With the Metaheuristic Mountain Gazelle Algorithm”. In: *IEEE Transactions on Consumer Electronics* (2024).
- [354] I. Syrigos, D. Kefalas, N. Makris, and T. Korakis. “EELAS: Energy Efficient and Latency Aware Scheduling of Cloud-Native ML Workloads”. In: *2023 15th International Conference on COMmunication Systems & NETworkS (COMSNETS)*. IEEE. 2023, pp. 819–824.
- [355] S. Rac and M. Brorsson. “Cost-effective scheduling for kubernetes in the edge-to-cloud continuum”. In: *2023 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE. 2023, pp. 153–160.
- [356] M. Yu, A. Liu, N. N. Xiong, and T. Wang. “An intelligent game-based offloading scheme for maximizing benefits of IoT-edge-cloud ecosystems”. In: *IEEE Internet of Things Journal* 9.8 (2020), pp. 5600–5616.

- [357] H. Wu, K. Wolter, P. Jiao, Y. Deng, Y. Zhao, and M. Xu. “EEDTO: An energy-efficient dynamic task offloading algorithm for blockchain-enabled IoT-edge-cloud orchestrated computing”. In: *IEEE Internet of Things Journal* 8.4 (2020), pp. 2163–2176.
- [358] S. Abdi, M. Ashjaei, and S. Mubeen. “Cost-aware workflow offloading in edge-cloud computing using a genetic algorithm”. In: *The Journal of Supercomputing* (2024), pp. 1–36.
- [359] N. Mehran, Z. N. Samani, D. Kimovski, and R. Prodan. “Matching-based scheduling of asynchronous data processing workflows on the computing continuum”. In: *2022 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE. 2022, pp. 58–70.
- [360] D. Rosendo, A. Costan, G. Antoniu, M. Simonin, J.-C. Lombardo, A. Joly, and P. Valduriez. “Reproducible performance optimization of complex applications on the edge-to-cloud continuum”. In: *2021 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE. 2021, pp. 23–34.
- [361] I. Syrigos, N. Angelopoulos, and T. Korakis. “Optimization of Execution for Machine Learning Applications in the Computing Continuum”. In: *2022 IEEE Conference on Standards for Communications and Networking (CSCN)*. IEEE. 2022, pp. 118–123.
- [362] G. L. Stavrinides and H. D. Karatza. “Resource allocation and scheduling of real-time workflow applications in an iot-fog-cloud environment”. In: *2022 Seventh International Conference on Fog and Mobile Edge Computing (FMEC)*. IEEE. 2022, pp. 1–8.
- [363] V. Lukaj, A. Catalfamo, F. Martella, M. Fazio, M. Villari, and A. Celesti. “A nosql dbms transparent data encryption approach for cloud/edge continuum”. In: *2023 IEEE Symposium on Computers and Communications (ISCC)*. IEEE. 2023, pp. 430–435.
- [364] A. Catalfamo, A. Celesti, M. Fazio, and M. Villari. “A homomorphic encryption service to secure data processing in a cloud/edge continuum context”. In: *2022 9th International Conference on Future Internet of Things and Cloud (FiCloud)*. IEEE. 2022, pp. 55–61.
- [365] D. Ayed, E. Jaho, C. Lachner, Z. Á. Mann, R. Seidl, and M. Surridge. “FogProtect: Protecting sensitive data in the computing continuum”. In: *Advances in Service-Oriented and Cloud Computing: International Workshops of ESOCC 2020, Heraklion, Crete, Greece, September 28–30, 2020, Revised Selected Papers 8*. Springer. 2021, pp. 179–184.
- [366] H. Xu, S. Qi, Y. Qi, W. Wei, and N. Xiong. “Secure and lightweight blockchain-based truthful data trading for real-time vehicular crowdsensing”. In: *ACM Transactions on Embedded Computing Systems* 23.1 (2024), pp. 1–31.
- [367] S. Ó. Murphy, U. Roedig, C. J. Sreenan, and A. Khalid. “Towards trust-based data weighting in machine learning”. In: *2023 IEEE 31st International Conference on Network Protocols (ICNP)*. IEEE. 2023, pp. 1–6.

- [368] A. Chaudhary, S. K. Peddoju, and V. Chouhan. “Secure authentication and reliable cloud storage scheme for iot-edge-cloud integration”. In: *Journal of Grid Computing* 21.3 (2023), p. 35.
- [369] B.-C. Mocanu, R. Stoleriu, A.-E. Mocanu, C. Negru, E.-G. Drăgotoiu, M.-A. Moisescu, and F. Pop. “NextEDR—Next generation agent-based EDR systems for cybersecurity threats”. In: *2024 32nd Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*. IEEE. 2024, pp. 183–190.
- [370] Y. Li, Z. Zhou, X. Xue, D. Zhao, and P. C. Hung. “Accurate anomaly detection with energy efficiency in IoT–Edge–Cloud collaborative networks”. In: *IEEE Internet of Things Journal* 10.19 (2023), pp. 16959–16974.
- [371] S. Manimurugan. “IoT–Fog–Cloud model for anomaly detection using improved Naïve Bayes and principal component analysis”. In: *Journal of Ambient Intelligence and Humanized Computing* (2021), pp. 1–10.
- [372] M. Barbareschi, V. Casola, and D. Lombardi. “Ensuring End-to-End Security in Computing Continuum Exploiting Physical Unclonable Functions”. In: *2023 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE. 2023, pp. 273–278.
- [373] L. Loffi, C. M. Westphall, L. D. Grüdtner, and C. B. Westphall. “Mutual authentication with multi-factor in IoT–Fog–Cloud environment”. In: *Journal of Network and Computer Applications* 176 (2021), p. 102932.
- [374] M. A. Aleisa, A. Abuhussein, F. S. Alsubaei, and F. T. Sheldon. “Novel security models for IoT–Fog–cloud architectures in a real-world environment”. In: *Applied Sciences* 12.10 (2022), p. 4837.
- [375] M. Seifenasr, R. AlTawy, A. Youssef, and E. Ghadafi. “Privacy-Preserving Mutual Authentication Protocol With Forward Secrecy for IoT–Edge–Cloud”. In: *IEEE Internet of Things Journal* (2023).
- [376] M. Babitha and M. Siddappa. “A Secure Authentication Model with Optimized Key Generation in IoT–Fog–Cloud Computing”. In: *2023 International Conference on Integrated Intelligence and Communication Systems (ICIICS)*. IEEE. 2023, pp. 1–9.
- [377] F.-J. Lordan Gomis, A. Martin, and D. Lezzi. “Securing the Execution of ML Workflows across the Compute Continua”. In: *Companion of the 2023 ACM/SPEC International Conference on Performance Engineering*. 2023, pp. 269–276.
- [378] G. Morabito, C. Sicari, A. Ruggeri, A. Celesti, and L. Carnevale. “Secure-by-design serverless workflows on the edge–cloud continuum through the osmotic computing paradigm”. In: *Internet of Things* 22 (2023), p. 100737.
- [379] G. Morabito, C. Sicari, L. Carnevale, A. Galletta, G. Di Modica, and M. Villari. “Securing serverless workflows on the cloud edge continuum”. In: *2023 IEEE/ACM 23rd International Symposium on Cluster, Cloud and Internet Computing Workshops (CCGridW)*. IEEE. 2023, pp. 118–124.

- [380] A. Bocci, S. Forti, G.-L. Ferrari, and A. Brogi. “Declarative secure placement of faas orchestrations in the cloud-edge continuum”. In: *Electronics* 12.6 (2023), p. 1332.
- [381] L. Bacchiani, G. De Palma, L. Sciullo, M. Bravetti, M. Di Felice, M. Gabbrielli, G. Zavattaro, and R. Della Penna. “Low-latency anomaly detection on the edge-cloud continuum for Industry 4.0 applications: The SEAWALL case study”. In: *IEEE Internet of Things Magazine* 5.3 (2022), pp. 32–37.
- [382] I. Dhanapala, S. Bharti, A. McGibney, and S. Rea. “Towards a Performance-based Trustworthy Edge-Cloud Continuum”. In: *IEEE Access* (2024).
- [383] M. Brzozowski, P. Langendoerfer, A. Casaca, A. Grilo, M. Diaz, C. Martín, J. Camacho, and G. Landi. “UNITE: Integrated IoT-edge-cloud continuum”. In: *2022 IEEE 8th World Forum on Internet of Things (WF-IoT)*. IEEE. 2022, pp. 1–6.
- [384] T. Pfandzelter and D. Bermbach. “Enoki: Stateful distributed faas from edge to cloud”. In: *Proceedings of the 2nd International Workshop on Middleware for the Edge*. 2023, pp. 19–24.
- [385] H. Ko, B. Kim, Y. Kim, and S. Pack. “Two-Phase Split Computing Framework in Edge-Cloud Continuum”. In: *IEEE Internet of Things Journal* (2024).
- [386] M. Baiardi, G. Ciatto, and D. Pianini. “Infrastructures for the Edge-Cloud Continuum on a Small Scale: A Practical Case Study”. In: *2023 IEEE International Conference on Autonomic Computing and Self-Organizing Systems Companion (ACSOS-C)*. IEEE. 2023, pp. 128–133.
- [387] M. Jansen, A. Al-Dulaimy, A. V. Papadopoulos, A. Trivedi, and A. Iosup. “The SPEC-RG reference architecture for the compute continuum”. In: *2023 IEEE/ACM 23rd International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. IEEE. 2023, pp. 469–484.
- [388] R. Rodrigues Filho, L. F. Bittencourt, B. Porter, and F. M. Costa. “Exploiting the potential of the edge-cloud continuum with self-distributing systems”. In: *2022 IEEE/ACM 15th International Conference on Utility and Cloud Computing (UCC)*. IEEE. 2022, pp. 255–260.
- [389] V. C. Pujol, P. Raith, and S. Dustdar. “Towards a new paradigm for managing computing continuum applications”. In: *2021 IEEE Third International Conference on Cognitive Machine Intelligence (CogMI)*. IEEE. 2021, pp. 180–188.
- [390] A. J. Ferrer, S. Becker, F. Schmidt, L. Thamsen, and O. Kao. “Towards a cognitive compute continuum: an architecture for ad-hoc self-managed swarms”. In: *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. IEEE. 2021, pp. 634–641.
- [391] J. Spillner. “Self-balancing architectures based on liquid functions across computing continuums”. In: *Proceedings of the 14th IEEE/ACM International Conference on Utility and Cloud Computing Companion*. 2021, pp. 1–6.
- [392] J. Nam and M. Choi. “IoT edge cloud platform with Revocatable blockchain smart contract”. In: *Journal of Logistics, Informatics and Service Science* 9.2 (2022), pp. 131–144.

- [393] R. Goleva, R. Sokullu, V. Kadrev, A. Savov, S. Mihaylov, and N. Garcia. “Real-Time and Near-Real-Time Services in Distributed Environment for IoT–Edge–Cloud Computing Implementation in Agriculture and Well-Being”. In: *International Conference on Computer Science and Education in Computer Science*. Springer. 2022, pp. 126–141.
- [394] L. Carnevale, A. Filograna, F. Arigliano, R. Marino, A. Ruggeri, and M. Fazio. “Supporting the Natural Disaster Management Distributing Federated Intelligence over the Cloud–Edge Continuum: the TEMA Architecture”. In: *Proceedings of the IEEE/ACM 10th International Conference on Big Data Computing, Applications and Technologies*. 2023, pp. 1–6.
- [395] T. A. Ahanger, U. Tariq, A. Aldaej, A. Almehizia, and M. Bhatia. “IoT-inspired smart disaster evacuation framework”. In: *IEEE Internet of Things Journal* 11.7 (2023), pp. 12885–12892.
- [396] A. Alqahtani, S. Alsabai, and M. Bhatia. “Applied artificial intelligence framework for smart evacuation in industrial disasters”. In: *Applied Intelligence* 54.11 (2024), pp. 7030–7045.
- [397] M. Singh, S. Bharti, H. Kaur, V. Arora, M. Saini, M. Kaur, and J. Singh. “A Facial and Vocal Expression Based Comprehensive Framework for Real-Time Student Stress Monitoring in an IoT–Fog–Cloud Environment”. In: *IEEE Access* 10 (2022), pp. 63177–63188. DOI: 10.1109/ACCESS.2022.3183077.
- [398] L. Gigli, I. Zyrianoff, F. Zonzini, D. Bogomolov, N. Testoni, M. D. Felice, L. De Marchi, G. Augugliaro, C. Mennuti, and A. Marzani. “Next Generation Edge–Cloud Continuum Architecture for Structural Health Monitoring”. In: *IEEE Transactions on Industrial Informatics* 20.4 (2024), pp. 5874–5887. DOI: 10.1109/TII.2023.3337391.
- [399] K. N. Haque, J. Islam, I. Ahmad, and E. Harjula. “Decentralized Pub/Sub Architecture for Real-Time Remote Patient Monitoring: A Feasibility Study”. In: *Digital Health and Wireless Solutions*. Cham: Springer Nature Switzerland, 2024, pp. 48–65. ISBN: 978-3-031-59080-1.
- [400] S. A. AlQahtani. “An Evaluation of e–Health Service Performance through the Integration of 5G IoT, Fog, and Cloud Computing”. In: *Sensors* 23.11 (2023). ISSN: 1424-8220. DOI: 10.3390/s23115006. URL: <https://www.mdpi.com/1424-8220/23/11/5006>.
- [401] M. Bhatia and S. Kumari. “A novel IoT–fog–cloud-based healthcare system for monitoring and preventing encephalitis”. In: *Cognitive Computation* 14.5 (2022), pp. 1609–1626.
- [402] A. Aral, A. Esposito, A. Nagiyev, S. Benkner, B. Di Martino, and M. A. Bochicchio. “Experiences in Architectural Design and Deployment of eHealth and Environmental Applications for Cloud–Edge Continuum”. In: *Advanced Information Networking and Applications*. Ed. by L. Barolli. Cham: Springer International Publishing, 2023, pp. 136–145. ISBN: 978-3-031-28694-0.
- [403] A. Kallel, M. Rekik, and M. Khemakhem. “IoT–fog–cloud based architecture for smart systems: Prototypes of autism and COVID-19 monitoring systems”. In: *Software: Practice and Experience* 51.1 (2021), pp. 91–116.

- [404] F. A. Silva, T. A. Nguyen, I. Fé, C. Brito, D. Min, and J.-W. Lee. “Performance Evaluation of an Internet of Healthcare Things for Medical Monitoring Using M/M/c/K Queuing Models”. In: *IEEE Access* 9 (2021), pp. 55271–55283. DOI: 10.1109/ACCESS.2021.3071508.
- [405] A. Borghesi, G. Di Modica, P. Bellavista, V. Gowtham, A. Willner, D. Nehls, F. Kintzler, S. Cejka, S. R. Tisbeni, A. Costantini, et al. “Iotwins: Design and implementation of a platform for the management of digital twins in industrial scenarios”. In: *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. IEEE. 2021, pp. 625–633.
- [406] D. K. Panda, V. Chaudhary, E. Fosler-Lussier, R. Machiraju, A. Majumdar, B. Plale, R. Ramnath, P. Sadayappan, N. Savardekar, and K. Tomko. “Creating intelligent cyberinfrastructure for democratizing AI”. In: *AI Magazine* 45.1 (2024), pp. 22–28.
- [407] P. Townend, A. P. Martí, I. De La Iglesia, N. Matskanis, T. O. Timoudas, T. Hallmann, A. Lalaguna, K. Swat, F. Renzi, D. Bocheński, et al. “Cognit: Challenges and vision for a serverless and multi-provider cognitive cloud-edge continuum”. In: *2023 IEEE International Conference on Edge Computing and Communications (EDGE)*. IEEE. 2023, pp. 12–22.
- [408] M. Moussa, P. Glass, N. Abdennahder, G. Di Marzo Serugendo, and R. Couturier. “Towards a Decentralised Federated Learning Based Compute Continuum Framework”. In: *European Conference on Service-Oriented and Cloud Computing*. Springer. 2023, pp. 219–230.
- [409] P. Trakadas, X. Masip-Bruin, F. M. Facca, S. T. Spantideas, A. E. Giannopoulos, N. C. Kapsalis, R. Martins, E. Bosani, J. Ramon, R. G. Prats, et al. “A reference architecture for cloud–edge meta-operating systems enabling cross–domain, data-intensive, ML-assisted applications: Architectural overview and key concepts”. In: *Sensors* 22.22 (2022), p. 9003.
- [410] P. Patros, M. Ooi, V. Huang, M. Mayo, C. Anderson, S. Burroughs, M. Baughman, O. Almurshed, O. Rana, R. Chard, K. Chard, and I. Foster. “Rural AI: Serverless-Powered Federated Learning for Remote Applications”. In: *IEEE Internet Computing* 27.2 (2023), pp. 28–34. DOI: 10.1109/MIC.2022.3202764.
- [411] Y. Cui, Z. Zhang, N. Wang, L. Li, C. Chang, and T. Wei. “User-Distribution-Aware Federated Learning for Efficient Communication and Fast Inference”. In: *IEEE Transactions on Computers* 73.4 (2024), pp. 1004–1018. DOI: 10.1109/TC.2023.3327513.
- [412] T. Si Salem, G. Castellano, G. Neglia, F. Pianese, and A. Araldo. “Toward Inference Delivery Networks: Distributing Machine Learning With Optimality Guarantees”. In: *IEEE/ACM Transactions on Networking* 32.1 (2024), pp. 859–873. DOI: 10.1109/TNET.2023.3305922.
- [413] F. Marozzo, A. Orsino, D. Talia, and P. Trunfio. “Scaling machine learning at the edge-cloud: a distributed computing perspective”. In: *2023 19th International Conference on Distributed Computing in Smart Systems and the Internet of Things (DCOSS-IoT)*. IEEE. 2023, pp. 761–767.

- [414] E. Peltonen, A. Sojan, and T. Päivärinta. “Towards Real-time Learning for Edge-Cloud Continuum with Vehicular Computing”. In: *2021 IEEE 7th World Forum on Internet of Things (WF-IoT)*. 2021, pp. 921–926. DOI: 10 . 1109 / WF - IoT51360 . 2021 . 9595628.
- [415] G. Cicceri, G. Tricomi, Z. Benomar, F. Longo, A. Puliafito, and G. Merlino. “DILoCC: An approach for Distributed Incremental Learning across the Computing Continuum”. In: *2021 IEEE International Conference on Smart Computing (SMARTCOMP)*. 2021, pp. 113–120. DOI: 10 . 1109 / SMARTCOMP52413 . 2021 . 00036.
- [416] D. Bacciu et al. “TEACHING – Trustworthy autonomous cyber-physical applications through human-centred intelligence”. In: *2021 IEEE International Conference on Omni-Layer Intelligent Systems (COINS)*. 2021, pp. 1–6. DOI: 10 . 1109 / COINS51742 . 2021 . 9524099.
- [417] W. Zhuang, Y. Wen, and S. Zhang. “Joint Optimization in Edge-Cloud Continuum for Federated Unsupervised Person Re-identification”. In: *Proceedings of the 29th ACM International Conference on Multimedia*. MM '21. Virtual Event, China: Association for Computing Machinery, 2021, 433–441. ISBN: 9781450386517. DOI: 10 . 1145 / 3474085 . 3475182. URL: <https://doi.org/10.1145/3474085.3475182>.
- [418] S. Holzer, P. Frangoudis, C. Tsigkanos, and S. Dustdar. “SMT-as-a-Service for Fog-Supported Cyber-Physical Systems”. In: *Proceedings of the 25th International Conference on Distributed Computing and Networking*. ICDCN '24. Chennai, India: Association for Computing Machinery, 2024, 154–163. ISBN: 9798400716737. DOI: 10 . 1145 / 3631461 . 3631562. URL: <https://doi.org/10.1145/3631461.3631562>.
- [419] S. Milani, D. Garlisi, C. Carugno, C. Tedesco, and I. Chatzigiannakis. “Edge2LoRa: Enabling edge computing on long-range wide-area Internet of Things”. In: *Internet of Things* 27 (2024), p. 101266.
- [420] Y. Ahn, M. Kim, J. Lee, Y. Shen, and J. Jeong. “IoT Edge-Cloud: An Internet-of-Things Edge-Empowered Cloud System for Device Management in Smart Spaces”. In: *IEEE Network* 38.3 (2024), pp. 109–117. DOI: 10 . 1109 / MNET . 137 . 2200565.
- [421] P. Mahajan and P. D. Kaur. “Three-tier IoT-edge-cloud (3T-IEC) architectural paradigm for real-time event recommendation in event-based social networks”. In: *Journal of Ambient Intelligence and Humanized Computing* 12.1 (2021), pp. 1363–1386.
- [422] G. Toffetti, L. Militano, R. Tharaka, and M. Straub. “ROS-based Robotic Applications Orchestration in the Compute Continuum: Challenges and Approaches”. In: *Proceedings of the IEEE/ACM 16th International Conference on Utility and Cloud Computing*. UCC '23. Taormina (Messina), Italy: Association for Computing Machinery, 2024. ISBN: 9798400702341. DOI: 10 . 1145 / 3603166 . 3632555. URL: <https://doi.org/10.1145/3603166.3632555>.

- [423] J. Legierski, K. Rachwal, P. Sowinski, W. Niewolski, P. Ratuszek, Z. Kopertowski, M. Paprzycki, and M. Ganzha. “Towards Edge–Cloud Architectures for Personal Protective Equipment Detection”. In: *Proceedings of the 4th International Conference on Information Management & Machine Intelligence. ICIMMI ’22*. Jaipur, India: Association for Computing Machinery, 2023. ISBN: 9781450399937. DOI: 10 . 1145 / 3590837 . 3590921. URL: <https://doi.org/10.1145/3590837.3590921>.
- [424] N. Mehran and R. Prodan. “CNN-assisted Road Sign Inspection on the Computing Continuum”. In: *2022 IEEE/ACM 15th International Conference on Utility and Cloud Computing (UCC)*. 2022, pp. 201–206. DOI: 10 . 1109/UCC56403.2022.00038.
- [425] K. Oztoprak, Y. K. Tuncel, and I. Butun. “Technological Transformation of Telco Operators towards Seamless IoT Edge–Cloud Continuum”. In: *Sensors* 23.2 (2023). ISSN: 1424–8220. DOI: 10 . 3390 / s23021004. URL: <https://www.mdpi.com/1424-8220/23/2/1004>.
- [426] Y. Lee and B. Yoo. “XR collaboration beyond virtual reality: work in the real world”. In: *Journal of Computational Design and Engineering* 8.2 (2021), pp. 756–772.
- [427] F. Meneghelli, M. Calore, D. Zucchetto, M. Polese, and A. Zanella. “IoT: Internet of threats? A survey of practical security vulnerabilities in real IoT devices”. In: *IEEE Internet of Things Journal* 6.5 (2019), pp. 8182–8201.
- [428] E. Zeydan, A. K. Yadav, P. Ranaweera, and M. Liyanage. “Securing IoT with Resilient Cloud–Edge Continuum”. In: *2024 International Conference on Intelligent Computing, Communication, Networking and Services (IC CNS)*. IEEE. 2024, pp. 38–45.
- [429] H. Dinh-Tuan. “CRACI: A Cloud–Native Reference Architecture for the Industrial Compute Continuum”. In: *arXiv preprint arXiv:2509.07498* (2025).
- [430] B. Scholten. *The road to integration: A guide to applying the ISA-95 standard in manufacturing*. Isa, 2007.
- [431] *Enterprise-control system integration - Part 1: Models and terminology*. IEC Standard. International Electrotechnical Commission, 2013.
- [432] EMQX. *Exploring ISA-95 Standards in Manufacturing*. Accessed: 2024-06-04. 2023. URL: <https://www.emqx.com/en/blog/exploring-isa95-standards-in-manufacturing>.
- [433] *ISA-88.00.01-2010 Batch Control Part 1: Models and Terminology*. International Society of Automation, 2010.
- [434] *Industrial-process measurement, control and automation - Life-cycle-management for systems and components*. IEC Standard. International Electrotechnical Commission, 2020.
- [435] M. Hankel and B. Rexroth. “The reference architectural model industrie 4.0 (rami 4.0)”. In: *Zvei* 2.2 (2015), pp. 4–9.
- [436] *Batch control - Part 1: Models and terminology*. IEC Standard. International Electrotechnical Commission, 1997.

- [437] P. Adolphs, H. Bedenbender, D. Dirzus, M. Ehlich, U. Epple, M. Hankel, R. Heidel, M. Hoffmeister, H. Huhle, B. Kärcher, et al. "Status report-reference architecture model industrie 4.0 (rami4. 0)". In: *VDI-Verein Deutscher Ingenieure eV and ZVEI-German Electrical and Electronic Manufacturers Association, Tech. Rep* (2015).
- [438] B. Otto, M. t. Hompel, and S. Wrobel. "International Data Spaces: Reference architecture for the digitization of industries". In: *Digital transformation* (2019), pp. 109–128.
- [439] FIWARE Foundation. *FIWARE Smart Industry Reference Architecture*. [https://www.fiware.org/wp-content/directories/marketing-toolbox/material/FIWAREBrochure\\_SmartIndustry.pdf](https://www.fiware.org/wp-content/directories/marketing-toolbox/material/FIWAREBrochure_SmartIndustry.pdf). FIWARE brochure. 2018.
- [440] Á. Alonso, A. Pozo, J. M. Cantera, F. De la Vega, and J. J. Hierro. "Industrial data space architecture implementation using FIWARE". In: *Sensors* 18.7 (2018), p. 2226.
- [441] J. de Caigny and R. Huck. "NAMUR Open Architecture—Der sichere Weg, neue Werte und Services aus Automatisierungsdaten zu schaffen". In: *Handbuch Industrie 4.0*. Springer, 2020, pp. 1–21.
- [442] C. Klettner, T. Tauchnitz, U. Epple, L. Nothdurft, C. Diedrich, T. Schröder, D. Großmann, S. Banerjee, M. Krauß, C. Iatrou, et al. "Namur open architecture: Die namur-pyramide wird geöffnet für industrie 4.0". In: *atp magazin* 59.01–02 (2017), pp. 20–37.
- [443] O. C. E. T. F. 3. *Functional View of the Continuum Reference Architecture: Minimum set of expected functionalities*. June 2024. DOI: 10.5281/zenodo.11656674. URL: <https://doi.org/10.5281/zenodo.11656674>.
- [444] M. Resman, M. Pipan, M. Šimic, and N. Herakovič. "A new architecture model for smart manufacturing: A performance analysis and comparison with the RAMI 4.0 reference model". In: *Adv. Prod. Eng. Manag* 14.2 (2019), pp. 153–165.
- [445] L. Kassner, C. Gröger, J. Königsberger, E. Hoos, C. Kiefer, C. Weber, S. Silcher, and B. Mitschang. "The Stuttgart IT architecture for manufacturing: an architecture for the data-driven factory". In: *International Conference on Enterprise Information Systems*. Springer, 2016, pp. 53–80.
- [446] J.-R. Jiang. "An improved cyber-physical systems architecture for Industry 4.0 smart factories". In: *Advances in Mechanical Engineering* 10.6 (2018), p. 1687814018784192.
- [447] J. Lee, B. Bagheri, and H.-A. Kao. "A cyber-physical systems architecture for industry 4.0-based manufacturing systems". In: *Manufacturing letters* 3 (2015), pp. 18–23.
- [448] D. Gannon, R. Barga, and N. Sundaresan. "Cloud-native applications". In: *IEEE Cloud Computing* 4.5 (2017), pp. 16–21.
- [449] A. Wiggins. *The Twelve-Factor App*. <https://12factor.net/>. Accessed: 2025-01-24. 2011.
- [450] J. Bartlett. "The Cloud Native Philosophy". In: *Cloud Native Applications with Docker and Kubernetes: Design and Build Cloud Architecture and Applications with Microservices, EMQ, and Multi-Site Configurations*. Springer, 2022, pp. 47–52.

- [451] B. M. Michelson. "Event-driven architecture overview". In: *Patricia Seybold Group* 2.12 (2006), pp. 10–1571.
- [452] H. Cervantes and R. Kazman. *Designing software architectures: a practical approach*. Addison-Wesley Professional, 2024.
- [453] A. Braud, G. Fromentoux, B. Radier, and O. Le Grand. "The road to European digital sovereignty with Gaia-X and IDSA". In: *IEEE network* 35.2 (2021), pp. 4–5.
- [454] J. Mügge, J. Grosse Erdmann, T. Riedelsheimer, M. M. Manoury, S.-O. Smolka, S. Wichmann, and K. Lindow. "Empowering end-of-life vehicle decision making with cross-company data exchange and data sovereignty via Catena-X". In: *Sustainability* 15.9 (2023), p. 7187.
- [455] D. Kushner. "The real story of stuxnet". In: *ieee Spectrum* 50.3 (2013), pp. 48–53.
- [456] S. Gilbert and N. Lynch. "Perspectives on the CAP Theorem". In: *Computer* 45.2 (2012), pp. 30–36.
- [457] H. Dinh-Tuan, F. Beierle, and S. R. Garzon. "MAIA: a microservices-based architecture for industrial data analytics". In: *2019 IEEE International Conference on Industrial Cyber Physical Systems (ICPS)*. IEEE. 2019, pp. 23–30.
- [458] D. Soni and A. Makwana. "A survey on mqtt: a protocol of internet of things (iot)". In: *International conference on telecommunication, power analysis and computing techniques (ICTPACT-2017)*. Vol. 20. 2017, pp. 173–177.
- [459] S. Burckhardt et al. "Principles of eventual consistency". In: *Foundations and Trends® in Programming Languages* 1.1–2 (2014), pp. 1–150.
- [460] P. R. M. Vasconcelos, G. A. A. Freitas, and T. G. Marques. "KVM, OpenVZ and Linux Containers: Performance Comparison of Virtualization for Web Conferencing Systems". In: *Int. J. Multimed. Image Process* 6.1/2 (2016).
- [461] J.-M. Yang, K. F. Li, W.-W. Li, and D.-F. Zhang. "Trading off logging overhead and coordinating overhead to achieve efficient rollback recovery". In: *Concurrency and Computation: Practice and Experience* 21.6 (2009), pp. 819–853.
- [462] V. Devadas and M. Curtis-Maury. "Scalable coordination of hierarchical parallelism". In: *Proceedings of the 49th International Conference on Parallel Processing*. 2020, pp. 1–11.
- [463] H. Dinh-Tuan, M. Mora-Martinez, F. Beierle, and S. R. Garzon. "Development frameworks for microservice-based applications: Evaluation and comparison". In: *Proceedings of the 2020 European Symposium on Software Engineering*. 2020, pp. 12–20.
- [464] R. E. Johnson and B. Foote. "Designing reusable classes". In: *Journal of object-oriented programming* 1.2 (1988), pp. 22–35.
- [465] E. Gamma. *Design patterns: elements of reusable object-oriented software*. Pearson Education India, 1995.

- [466] D. C. Schmidt, A. Gokhale, and B. Natarajan. “Leveraging application frameworks: why frameworks are important and how to apply them effectively”. In: *Queue* 2.5 (2004), pp. 66–75.
- [467] M. Fayad and D. C. Schmidt. “Object-oriented application frameworks”. In: *Communications of the ACM* 40.10 (1997), pp. 32–38.
- [468] R. A. Ballance, A. J. Giancola, G. F. Luger, and T. J. Ross. “A Framework-Based Environment for Object-Oriented Scientific Codes”. In: *Scientific Programming* 2.4 (1993), pp. 111–121.
- [469] N. M. Edwin. “Software frameworks, architectural and design patterns”. In: *Journal of Software Engineering and Applications* 7.8 (2014), pp. 670–678.
- [470] D. C. Schmidt, A. Gokhale, and B. Natarajan. “Frameworks: Why they are important and how to apply them effectively”. In: *ACM Queue magazine* 2.5 (2004).
- [471] Q. Li, Q. Tang, I. Chan, H. Wei, Y. Pu, H. Jiang, J. Li, and J. Zhou. “Smart manufacturing standardization: Architectures, reference models and standards framework”. In: *Computers in industry* 101 (2018), pp. 91–106.
- [472] N. Mäkitalo, A. Taivalsaari, A. Kiviluoto, T. Mikkonen, and R. Capilla. “On opportunistic software reuse”. In: *Computing* 102 (2020), pp. 2385–2408.
- [473] J. Singh, A. Gupta, and P. Kanwal. “The vital role of community in open source software development: A framework for assessment and ranking”. In: *Journal of Software: Evolution and Process* 36.7 (2024), e2643.
- [474] V. Myllärniemi, S. Kujala, M. Raatikainen, and P. Sevoón. “Development as a journey: factors supporting the adoption and use of software frameworks”. In: *Journal of software engineering research and development* 6 (2018), pp. 1–22.
- [475] D. Lion, A. Chiu, H. Sun, X. Zhuang, N. Grcevski, and D. Yuan. “Don’t get caught in the cold, warm-up your {JVM}: Understand and eliminate {JVM} warm-up overhead in data-parallel systems”. In: *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*. 2016, pp. 383–400.
- [476] J. Kim, T. J. Jun, D. Kang, D. Kim, and D. Kim. “GPU Enabled Serverless Computing Framework”. In: *2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*. IEEE. 2018, pp. 533–540.
- [477] S. Chen and M. Zhou. “Evolving container to unikernel for edge computing and applications in process industry”. In: *Processes* 9.2 (2021), p. 351.
- [478] T. Combe, A. Martin, and R. Di Pietro. “To docker or not to docker: A security perspective”. In: *IEEE Cloud Computing* 3.5 (2016), pp. 54–62.
- [479] J. Talbot, P. Pikula, C. Sweetmore, S. Rowe, H. Hindy, C. Tachtatzis, R. Atkinson, and X. Bellekens. “A security perspective on unikernels”. In: *2020 International Conference on Cyber Security and Protection of Digital Services (Cyber Security)*. IEEE. 2020, pp. 1–7.
- [480] T. Goethals, M. Sebrechts, M. Al-Naday, B. Volckaert, and F. De Turck. “A functional and performance benchmark of lightweight virtualization platforms for edge computing”. In: *2022 IEEE International conference on edge computing and communications (EDGE)*. IEEE. 2022, pp. 60–68.

- [481] K.-H. Chen, M. Günzel, J. Boguslaw, M. Buschhoff, and J.-J. Chen. “Unikernel-based real-time virtualization under deferrable servers: Analysis and realization”. In: *34th Euromicro Conference on Real-Time Systems, ECRTS 2022*. Dagstuhl. 2022, pp. 6–1.
- [482] J. Cito, G. Schermann, J. E. Wittern, P. Leitner, S. Zumberi, and H. C. Gall. “An empirical analysis of the docker container ecosystem on github”. In: *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. IEEE. 2017, pp. 323–333.
- [483] T. Siddiqui, S. A. Siddiqui, and N. A. Khan. “Comprehensive analysis of container technology”. In: *2019 4th international conference on information systems and computer networks (ISCON)*. IEEE. 2019, pp. 218–223.
- [484] R. Queiroz, T. Cruz, J. Mendes, P. Sousa, and P. Simões. “Container-based virtualization for real-time industrial systems—a systematic review”. In: *ACM Computing Surveys 56.3* (2023), pp. 1–38.
- [485] H. Dinh-Tuan. “Unikernels vs. Containers: A Runtime-Level Performance Comparison for Resource-Constrained Edge Workloads”. In: *arXiv preprint arXiv:2509.07891* (2025).
- [486] M. H. Khan. “What all you need to know about Unikernels | Characteristics, Implementation, and Benefits”. In: *Medium* (Nov. 2021). Accessed: 2024-05-12. URL: <https://damianarado.medium.com/what-all-you-need-to-know-about-unikernels-characteristics-implementation-and-benefits-915a6f9996af>.
- [487] A. M. Potdar, D. Narayan, S. Kengond, and M. M. Mulla. “Performance evaluation of docker container and virtual machine”. In: *Procedia Computer Science 171* (2020), pp. 1419–1428.
- [488] R. Morabito, J. Kjällman, and M. Komu. “Hypervisors vs. lightweight virtualization: a performance comparison”. In: *2015 IEEE International Conference on cloud engineering*. IEEE. 2015, pp. 386–393.
- [489] I. Mavridis and H. Karatza. “Lightweight virtualization approaches for software-defined systems and cloud computing: An evaluation of unikernels and containers”. In: *2019 Sixth International Conference on Software Defined Systems (SDS)*. IEEE. 2019, pp. 171–178.
- [490] NanoVMs, Inc. *Finding memory management errors in the Nanos Unikernel*. <https://nanovms.com/dev/tutorials/finding-memory-management-errors-in-nanos>. Accessed: 2024-07-12. 2025.
- [491] H. Dinh-Tuan and F. Beierle. “MS2M: A message-based approach for live stateful microservices migration”. In: *2022 5th Conference on Cloud and Internet of Things (CIoT)*. IEEE. 2022, pp. 100–107. DOI: 10.1109/CIoT53061.2022.9766576.
- [492] D. Balouek-Thomert, E. G. Renart, A. R. Zamani, A. Simonet, and M. Parashar. “Towards a computing continuum: Enabling edge-to-cloud integration for data-driven workflows”. In: *The International Journal of High Performance Computing Applications 33.6* (2019), pp. 1159–1174.

- [493] K. Kaur, F. Guillemin, and F. Sailhan. “Container placement and migration strategies for cloud, fog, and edge data centers: A survey”. In: *International Journal of Network Management* 32.6 (2022), e2212.
- [494] S. Wang, J. Xu, N. Zhang, and Y. Liu. “A Survey on Service Migration in Mobile Edge Computing”. In: *IEEE Access* 6 (2018), pp. 23511–23528. DOI: 10.1109/ACCESS.2018.2828102.
- [495] T. Chanikaphon and M. A. Salehi. “UMS: Live Migration of Containerized Services across Autonomous Computing Systems”. In: *GLOBECOM 2023 - 2023 IEEE Global Communications Conference*. 2023, pp. 467–472. DOI: 10.1109/GLOBECOM54140.2023.10437519.
- [496] Y. Lu and Y. Jiang. “A Container Pre-copy Migration Method Based on Dirty Page Prediction and Compression”. In: *2022 IEEE 28th International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE. 2023, pp. 704–711.
- [497] D. Tazzioli, R. Venanzi, and L. Foschini. “Stateful Service Migration Support for Kubernetes-based Orchestration in Industry 4.0”. In: *2024 IEEE Symposium on Computers and Communications (ISCC)*. IEEE. 2024, pp. 1–6.
- [498] P. Bellavista, S. Dahdal, L. Foschini, D. Tazzioli, M. Tortonesi, and R. Venanzi. “Kubernetes Enhanced Stateful Service Migration for ML-Driven Applications in Industry 4.0 Scenarios”. In: *2024 IEEE Annual Congress on Artificial Intelligence of Things (AIoT)*. IEEE. 2024, pp. 25–31.
- [499] L. Ma, S. Yi, N. Carter, and Q. Li. “Efficient live migration of edge services leveraging container layered storage”. In: *IEEE Transactions on Mobile Computing* 18.9 (2018), pp. 2020–2033.
- [500] A. Rahmatulloh, F. Nugraha, R. Gunawan, and I. Darmawan. “Event-Driven Architecture to Improve Performance and Scalability in Microservices-Based Systems”. In: *2022 International Conference Advancement in Data Science, E-learning and Information Systems (ICADEIS)* (2022), pp. 01–06. DOI: 10.1109/ICADEIS56544.2022.10037390.
- [501] G. Sun, D. Liao, V. Anand, D. Zhao, and H.-F. Yu. “A new technique for efficient live migration of multiple virtual machines”. In: *Future Gener. Comput. Syst.* 55 (2016), pp. 74–86. DOI: 10.1016/j.future.2015.09.005.
- [502] T. Wu, N. Guizani, and J.-S. Huang. “Live migration improvements by related dirty memory prediction in cloud computing”. In: *J. Netw. Comput. Appl.* 90 (2017), pp. 83–89. DOI: 10.1016/j.jnca.2017.03.011.
- [503] H. Dinh-Tuan and F. Beierle. “MS2M: A message-based approach for live stateful microservices migration”. In: *2022 5th Conference on Cloud and Internet of Things (CIoT)*. 2022, pp. 100–107. DOI: 10.1109/CIoT53061.2022.9766576.
- [504] S. Nadgowda, S. Suneja, N. Bila, and C. Isci. “Voyager: Complete container state migration”. In: *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. IEEE. 2017, pp. 2137–2142.
- [505] H. Dinh-Tuan and J. Jiang. “Optimizing Stateful Microservice Migration in Kubernetes with MS2M and Forensic Checkpointing”. In: *2025 28th Conference on Innovation in Clouds, Internet and Networks (ICIN)*. IEEE. 2025, pp. 83–90.

- [506] H. Dinh-Tuan and F. F. Six. “Optimizing Cloud-Native Services with SAGA: A Service Affinity Graph-Based Approach”. In: *2024 International Conference on Smart Applications, Communications and Networking (SmartNets)*. IEEE. 2024, pp. 1–6.
- [507] M. Mota-Cruz, J. H. Santos, J. F. Macedo, K. Velasquez, and D. P. Abreu. “Optimizing Microservices Placement in the Cloud-to-Edge Continuum: A Comparative Analysis of App and Service Based Approaches”. In: *2024 IEEE 22nd Mediterranean Electrotechnical Conference (MELECON)*. IEEE. 2024, pp. 1321–1326.
- [508] L. Liu and Q. Fan. “Resource Allocation Optimization Based on Mixed Integer Linear Programming in the Multi-Cloudlet Environment”. In: *IEEE Access* 6 (2018), pp. 24533–24542. DOI: 10.1109/ACCESS.2018.2830639.
- [509] Y. Fan, L. Wang, W. Wu, and D. Du. “Cloud/Edge Computing Resource Allocation and Pricing for Mobile Blockchain: An Iterative Greedy and Search Approach”. In: *IEEE Transactions on Computational Social Systems* 8.2 (2021), pp. 451–463. DOI: 10.1109/TCSS.2021.3049152.
- [510] A. Bento, J. Soares, A. Ferreira, J. Duraes, J. Ferreira, R. Carreira, F. Araujo, and R. Barbosa. “Bi-objective optimization of availability and cost for cloud services”. In: *2022 IEEE 21st International Symposium on Network Computing and Applications (NCA)*. Vol. 21. IEEE. 2022, pp. 45–53. DOI: 10.1109/NCA57778.2022.10013618.
- [511] W. Wei, H. Li, and W. Yang. “Cost-effective stochastic resource placement in edge clouds with horizontal and vertical sharing”. In: *Future Generation Computer Systems* 138 (2023), pp. 213–225.
- [512] R. K. Devi and G. Murugaboopathi. “An efficient clustering and load balancing of distributed cloud data centers using graph theory”. In: *International Journal of Communication Systems* 32.5 (2019), e3896. DOI: <https://doi.org/10.1002/dac.3896>.
- [513] K. Afachao, A. M. Abu-Mahfouz, and G. P. Hanke. “Efficient Microservice Deployment in the Edge-Cloud Networks With Policy-Gradient Reinforcement Learning”. In: *IEEE Access* (2024).
- [514] Y. Guo, F. Liu, N. Xiao, Z. Li, Z. Cai, G. Tang, and N. Liu. “PARA: Performability-aware resource allocation on the edges for cloud-native services”. In: *International Journal of Intelligent Systems* 37.11 (2022), pp. 8523–8547. DOI: <https://doi.org/10.1002/int.22954>.
- [515] B. Xu, S. Wu, J. Xiao, H. Jin, Y. Zhang, G. Shi, T. Lin, J. Rao, L. Yi, and J. Jiang. “Sledge: Towards Efficient Live Migration of Docker Containers”. In: *2020 IEEE 13th International Conference on Cloud Computing (CLOUD)*. IEEE. 2020, pp. 321–328. DOI: 10.1109/CLOUD49709.2020.00052.
- [516] M. Kwon, D. Gouk, C. Lee, B. Kim, J. Hwang, and M. Jung. “{DC-Store}: Eliminating noisy neighbor containers using deterministic {I/O} performance and resource isolation”. In: *18th USENIX Conference on File and Storage Technologies (FAST 20)*. 2020, pp. 183–191.

- [517] .NET Foundation. *Microsoft eShopOnWeb ASP.NET Core Reference Application*. Accessed July 12, 2023. Available: <https://github.com/dotnet-architecture/eShopOnWeb>. (Visited on 07/12/2023).
- [518] Broadcom Inc. *RabbitMQ – One broker to queue them all*. Accessed July 15, 2023. Available: <https://www.rabbitmq.com/>. (Visited on 07/15/2023).
- [519] Google. *Google Kubernetes Engine (GKE)*. Accessed July 17, 2023. Available: <https://cloud.google.com/kubernetes-engine/?hl=en>. (Visited on 07/17/2023).
- [520] Google. *Google Cloud Platform (GCP)*. Accessed July 17, 2023. Available: <https://cloud.google.com/>. (Visited on 07/17/2023).
- [521] Broadcom Inc. *Spring Microservices*. Accessed July 17, 2023. Available: <https://spring.io/microservices>. (Visited on 07/17/2023).
- [522] A. Parker, D. Spoonhower, J. Mace, B. Sigelman, and R. Isaacs. *Distributed tracing in practice: Instrumenting, analyzing, and debugging microservices*. O'Reilly Media, 2020.
- [523] S. Boyd, N. Parikh, E. Chu, B. Peleato, J. Eckstein, et al. “Distributed optimization and statistical learning via the alternating direction method of multipliers”. In: *Foundations and Trends® in Machine learning* 3.1 (2011), pp. 1–122.
- [524] A. Nedić, A. Olshevsky, and W. Shi. “Decentralized consensus optimization and resource allocation”. In: *Large-Scale and Distributed Optimization* (2018), pp. 247–287.
- [525] Z. Liu, X. Chen, X. Xu, and X. Guan. “A decentralized optimization method for energy saving of HVAC systems”. In: *2013 IEEE International Conference on Automation Science and Engineering (CASE)*. IEEE. 2013, pp. 225–230.
- [526] C. Hinrichs, S. Lehnhoff, and M. Sonnenschein. “COHDA: A combinatorial optimization heuristic for distributed agents”. In: *International Conference on Agents and Artificial Intelligence*. Springer. 2013, pp. 23–39.
- [527] J. Bremer and S. Lehnhoff. “An agent-based approach to decentralized global optimization—adapting cohda to coordinate descent”. In: *International Conference on Agents and Artificial Intelligence*. Vol. 2. SCITEPRESS. 2017, pp. 129–136.
- [528] Z. Nezami, K. Zamanifar, K. Djemame, and E. Pournaras. “Decentralized edge-to-cloud load balancing: Service placement for the Internet of Things”. In: *IEEE Access* 9 (2021), pp. 64983–65000.
- [529] K. K. Kondru and S. Rajakodi. “RaftOptima: An Optimised Raft with enhanced Fault Tolerance, and increased Scalability with low latency”. In: *IEEE Access* (2024).