PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA

**MINISTRY OF HIGHER EDUCATION AND SCIENTIFIC RESEARCH**

**UNIVERSITY KASDI MERBAH OF OUARGLA**

Faculty Of New Technologies Of Information And Communication

Department Of Computer Science And Information Technology


MASTER thesis

**Domain:** Computer science and Information technology

**Field :** Computer science

**Speciality: Fondamental Computer Science**

**by :** Manal MESSAID and Mohammed Mahdi BENEDDINE

**Theme**

# Orchestration of Interdependent Tasks in Edge Computing Systems

Evaluation Date 12/06/2025

Jury:

| | | | |
|---|---|---|---|
| President | Azzaoui Hanane | MCB | University of Ouargla |
| Examiner | Hamrouni Basma | MCB | University of Ouargla |
| Supervisor | MechaIikh Charaf Eddine | MCB | University of Ouargla |

*Academic year: 2024/2025*

# Acknowledgements

In the name of Allah, the Most Gracious, the Most Merciful. Peace and blessings be upon our Prophet Muhammad, his family, and his companions.

At the conclusion of this academic journey, we are honored to express our deepest gratitude to everyone who supported us and contributed to the completion of this work, whether through academic guidance or moral encouragement.

Manal MESSAID:

I would like to begin by sincerely thanking our esteemed supervisor, Professor Mechalikh Charafeddine, for his valuable guidance, continuous support, and insightful feedback throughout the course of this project. I dedicate this work with pride and deep affection to the memory of my beloved mother (may Allah have mercy on her soul), whose love, strength, and values continue to guide me even in her absence. Her spirit has been a constant source of strength and inspiration. I also extend my heartfelt thanks to my dear family for their unwavering support and encouragement throughout this academic journey. Moreover, I express my deep gratitude to my friend and colleague Mohammed Mahdi BENEDDINE, who was a true partner in this project. His cooperation, dedication, and team spirit greatly contributed to overcoming challenges and achieving our shared goals.

Mohammed Mahdi BENEDDINE:

I sincerely thank our supervisor, Professor Mechalikh Charafeddine, for his generous guidance and expertise, which were essential to the success of this work. I am also deeply grateful to my family for their constant support, motivation, and for providing the ideal environment to pursue my studies. I would also like to express my appreciation to my friend and teammate Manal Messaid, who demonstrated great dedication and commitment throughout this journey. Her collaborative spirit and perseverance played a vital role in bringing this work to fruition.

Together, we extend our heartfelt thanks to all the professors and staff of the Computer Science Department at Kasdi Merbah University, Ouargla, for their efforts and dedication in shaping our academic foundation. We are equally grateful to our friends and fellow students

who accompanied us and supported us along this path.

Above all, we thank Allah Almighty for granting us the strength and perseverance to complete this project. We pray that it marks the beginning of a meaningful and successful professional journey, God willing.

# Abstract

The rapid expansion of Internet of Things (IoT) devices has redefined computing demands, necessitating low-latency, real-time, and bandwidth-efficient solutions. Although cloud computing offers scalability and processing power, its centralized nature introduces latency and bottlenecks—issues especially critical in applications like autonomous systems, smart healthcare, and industrial IoT.

Edge computing addresses these limitations by relocating computation closer to data sources, reducing transmission delays and supporting real-time processing. However, the shift from monolithic to microservice-based applications introduces new challenges for orchestrating workloads across distributed, heterogeneous edge environments.

Existing orchestration methods are often static or tailored for centralized systems, lacking adaptability to mobile contexts, dynamic resource availability, and network fluctuations. This work proposes a heuristic-driven dynamic partitioning algorithm designed to orchestrate microservice-based workloads intelligently across edge nodes. It considers real-time factors such as node proximity, energy constraints, and task dependencies to optimize latency, resource utilization, and responsiveness.

The approach is evaluated using PureEdgeSim, a simulation toolkit supporting mobility, energy-awareness, and microservice orchestration. Results show a clear reduction in end-to-end latency, improved load balancing, and better adaptability under dynamic conditions.

These outcomes demonstrate the promise of heuristic-based orchestration in enabling efficient, scalable, and context-aware edge computing.

***Key words:*** *Edge Computing, Heuristic Algorithms, Workload Orchestration, Dynamic Partitioning, Microservices, IoT.*

# Résumé

L'expansion rapide des dispositifs de l'Internet des objets (IoT) a redéfini les exigences en matière de calcul, imposant des solutions à faible latence, en temps réel et économes en bande passante. Bien que l'informatique en nuage offre évolutivité et puissance de traitement, sa nature centralisée engendre une latence et des goulets d'étranglement — des contraintes particulièrement critiques dans des domaines comme les systèmes autonomes, la santé intelligente et l'IoT industriel.

L'edge computing répond à ces limitations en rapprochant le traitement des données de leur source, réduisant ainsi les délais de transmission et améliorant la réactivité. Toutefois, la transition vers des applications modulaires basées sur les microservices complique l'orchestration des charges de travail dans des environnements distribués et hétérogènes.

Les méthodes d'orchestration actuelles, souvent statiques ou conçues pour des systèmes centralisés, manquent de flexibilité face à la mobilité, à la variabilité des ressources et aux fluctuations du réseau. Ce travail propose un algorithme de partitionnement dynamique, fondé sur des techniques heuristiques, pour orchestrer intelligemment les microservices à travers les nœuds de l'edge. Il prend en compte des facteurs en temps réel tels que la proximité des nœuds, les contraintes énergétiques et les dépendances entre tâches afin d'optimiser la latence, l'utilisation des ressources et la réactivité.

L'approche est évaluée avec PureEdgeSim, un outil de simulation prenant en charge la mobilité, la consommation d'énergie et l'orchestration de microservices. Les résultats montrent une réduction significative de la latence, une meilleure répartition de la charge, et une adaptation accrue aux conditions dynamiques.

***Mots clés:*** *Edge Computing, Algorithmes Heuristiques, Orchestration de Charges, Partitionnement Dynamique, Microservices, IoT.*

# Contents

# List of Figures

# List of Tables

# List of Abbreviations

| | |
|---|---|
| **IOT**: | Internet Of Things |
| **QOS**: | Quality Of Service |
| **MCC**: | Mobile Cloud Computing |
| **F-RAN**: | Fog Radio Access Network |
| **MEC**: | Multi-access Edge Computing |
| **YAFS**: | Yet Another Fog Simulator |
| **LEAF**: | Lihgtweight Edge/Fog Simulator |
| **DES**: | Discrete Event Simulation |
| **AR**: | Augmented Reality |
| **VR**: | Virtual Reality |
| **BC**: | Betweenness Centralit |
| **CMAB**: | Contextual Multy Armed Bandit |
| **GA**: | Genetic Algorithm |
| **MIPS**: | Million Instructions Per Second |

# Chapter 1

# General Introduction

The widespread and rapid proliferation of Internet of Things (IoT) devices has brought about a true revolution in modern digital ecosystems, enabling the development of advanced applications in various domains such as smart healthcare (e-health), autonomous vehicles, and smart cities. These applications are characterized by stringent requirements, including ultra-low latency, high reliability, and substantial real-time data processing capabilities [6]. Although traditional cloud computing remains a powerful solution capable of handling massive amounts of data due to its centralized architecture, this very architecture introduces significant challenges. It often results in noticeable latency because of the physical distance between data-generating devices and processing centers. Moreover, relying entirely on cloud connectivity reduces the system's ability to respond effectively to urgent events and increases the risk of network bottlenecks.

To address these limitations, edge computing has emerged as a promising paradigm that overcomes the structural shortcomings of cloud computing. This model brings computing and storage resources closer to where data is generated, significantly reducing latency, improving network efficiency, and enhancing data security by enabling local processing [26]. As IoT applications become increasingly complex, the need for intelligent solutions to dynamically manage workloads at the edge has grown substantially. This opens the door to smart orchestration algorithms and modular, partitionable architectures as key areas of research in this evolving field.

Edge computing fundamentally transforms the traditional computing model by decentralizing data processing and bringing computational capabilities closer to the data sources—typically sensors, IoT devices, or user terminals. Unlike the cloud-centric approach, where raw data must travel long distances to centralized data centers for processing, edge computing minimizes the need for such transmissions by enabling localized analysis and decision-making.

This proximity significantly reduces end-to-end latency and alleviates bandwidth consumption, which is particularly beneficial in scenarios with limited or intermittent connectivity.

By processing data at or near the edge of the network, this paradigm supports real-time responsiveness, allowing systems to act almost instantaneously on the information they receive. This is critical in environments where delays can lead to performance degradation or even safety risks, such as in autonomous driving, industrial automation, or remote healthcare monitoring [6, 26]. Furthermore, edge computing distributes the computational load, easing pressure on central servers and enabling more scalable and resilient architectures.

In addition to performance benefits, edge computing also contributes to enhanced data privacy and security. By keeping sensitive information closer to its source and limiting its exposure to external networks, the risk of data breaches or unauthorized access is reduced. As applications continue to evolve toward higher levels of interactivity and context-awareness, the ability of edge computing to facilitate intelligent, localized, and adaptive computing becomes increasingly indispensable.

While a significant body of research has addressed workload orchestration within both cloud and edge computing environments, the majority of these efforts operate under the assumption that applications are monolithic in nature—i.e., designed as single, indivisible units that must be deployed and executed as a whole [25].

Such assumptions simplify orchestration strategies by reducing the complexity of component distribution and interdependencies. However, this traditional model no longer reflects the architecture of modern software systems.

Contemporary applications are increasingly modular, built as loosely coupled microservices that can be independently deployed, scaled, and updated. These microservices often span across heterogeneous nodes—from centralized cloud data centers to resource-constrained edge devices—and are interconnected through dynamic communication channels [25].

This architectural shift, while improving scalability and flexibility, introduces a new dimension of complexity in workload orchestration. Specifically, orchestrators must now manage not only where to place services, but also how to maintain service continuity and inter-service communication under changing conditions.

Factors such as user mobility, device heterogeneity, fluctuating network latency, and limited edge resources further complicate orchestration.

Moreover, orchestrating modular applications requires awareness of service dependencies, latency sensitivity, and load balancing, often under highly constrained or volatile environments.

Thus, the challenge lies in designing dynamic, context-aware, and efficient orchestration mechanisms that can adapt in real time to changing conditions while meeting Quality of Service (QoS) requirements.

The primary objective of this research is to design and implement a dynamic partitioning mechanism that leverages heuristic algorithms to efficiently orchestrate distributed, microservice-based applications within edge computing environments.

Unlike static or rule-based partitioning methods, the proposed approach dynamically adapts to changes in network conditions, resource availability, and application-specific constraints.

By utilizing heuristic algorithms—such as greedy methods, genetic algorithms, or ant colony optimization—the system can explore a large solution space and identify near-optimal deployment strategies within a reasonable computational time.

The key goals of the proposed dynamic partitioner include:

- Minimizing end-to-end latency.
- Improving load balancing across heterogeneous edge nodes.
- Adapting to user mobility and dynamic network topologies.
- Reducing energy consumption and communication overhead.
- Ensuring service continuity and QoS even in volatile environments.

The dynamic partitioner will be evaluated through simulations in PureEdgeSim, validating its effectiveness under various realistic edge computing scenarios.

This paper is structured as follows:

- **Chapter 2:** Introduces the edge computing paradigm, related concepts like fog and mist computing, and presents the main simulation tools used for edge computing research, with a focus on PureEdgeSim.
- **Chapter 3:** Reviews existing literature on workload orchestration and service management techniques in edge environments, with a focus on heuristic and AI-based solutions, and highlights existing gaps.
- **Chapter 4:** Details the design of the dynamic partitioning system, the heuristic algorithm employed, the decision-making criteria, and the overall system architecture.

- **Chapter 5:** Describes the experimental setup, including parameters, evaluation metrics, and test scenarios. Presents and discusses the results obtained using PureEdgeSim compared to baseline orchestration techniques.
- **Chapter 6:** Summarizes the findings, contributions, and implications of the research. Outlines potential extensions, such as integrating machine learning for predictive partitioning or validating the model in real-world edge environments.

In conclusion, edge computing offers a transformative approach to handling the growing demands of modern applications by decentralizing data processing and bringing computational resources closer to the data source. This model is particularly well-suited for real-time, latency-sensitive applications, such as those in IoT, autonomous vehicles, and smart cities. The dynamic partitioning mechanism proposed in this research aims to address the challenges associated with modular, microservice-based applications in edge environments. By leveraging heuristic algorithms for efficient orchestration, the system ensures improved performance, load balancing, and service continuity, even under changing conditions. The results of this research, evaluated through simulations in PureEdgeSim, will contribute to advancing the field of edge computing and offer valuable insights into the design of scalable, resilient, and adaptive systems.

# Chapter 2

# Background

Edge computing represents a fundamental paradigm shift in the way computational resources are deployed and utilized. Unlike traditional cloud computing, which relies on centralized data centers located far from end users, edge computing decentralizes processing by placing computational tasks closer to the data sources—typically at or near the network edge. This localized approach reduces end-to-end latency, minimizes bandwidth usage, and enables real-time responsiveness, making it particularly suitable for latency-sensitive and bandwidth-constrained applications such as autonomous vehicles, augmented reality, and industrial IoT.[26]

However, edge computing does not operate in isolation. It is part of a broader ecosystm of computing paradigms that aim to complement and enhance its capabilities.

## 2.1 Edge Computing and Its Related Computing Paradigms

Several related models contribute to this layered architecture, each addressing specific limitations or extending functionality:

### 2.1.1 Cloud Computing

As the foundational paradigm, cloud computing provides virtually unlimited, scalable, and centralized processing and storage resources. It is well-suited for compute-intensive tasks and long-term data analytics. However, its dependence on distant data centers leads to higher latency and increased network load, making it less optimal for real-time applications[9].

### 2.1.2 Mobile Cloud Computing (MCC)

This paradigm bridges the gap between mobile devices and cloud infrastructure by enabling offloading of computational tasks from resource-limited mobile devices to the cloud. While MCC helps reduce energy consumption and extend device capabilities, its reliance on stable network connectivity and cloud availability makes it vulnerable in highly dynamic or constrained environments.

### 2.1.3 Fog Computing

Introduced as an intermediary layer between the cloud and edge, fog computing extends cloud services to the local network level—closer to end devices. It enhances support for location-awareness, context sensitivity, and faster processing. Fog nodes, such as gateways and routers, can perform real-time analytics and decision-making[22], easing the burden on cloud infrastructure and improving service quality.

### 2.1.4 Fog Radio Access Network (F-RAN)

A specialized evolution of fog computing, F-RAN integrates computing resources into the radio access network (RAN) of cellular systems. It addresses the challenge of fronthaul congestion by offloading traffic and computations to nearby fog nodes, thus improving network efficiency and reducing transmission latency in mobile environments.

### 2.1.5 Multi-access Edge Computing (MEC)

MEC is a telecom-driven extension of edge computing that incorporates computational resources directly into the mobile network infrastructure, particularly at base stations and aggregation points in 4G/5G networks. MEC enables ultra-low latency services and high-bandwidth applications[33] by providing processing and storage capabilities at the network edge, often with direct access to radio network information for enhanced decision-making.

### 2.1.6 Mist Computing

Operating at the "extreme edge," mist computing pushes computation further down to the level of IoT devices themselves, such as sensors and actuators. These devices are equipped with limited processing power and are capable of performing basic analysis and decision-

making locally[26]. Mist computing reduces the dependency on upstream nodes and enhances autonomy, particularly useful in highly distributed or remote environments.

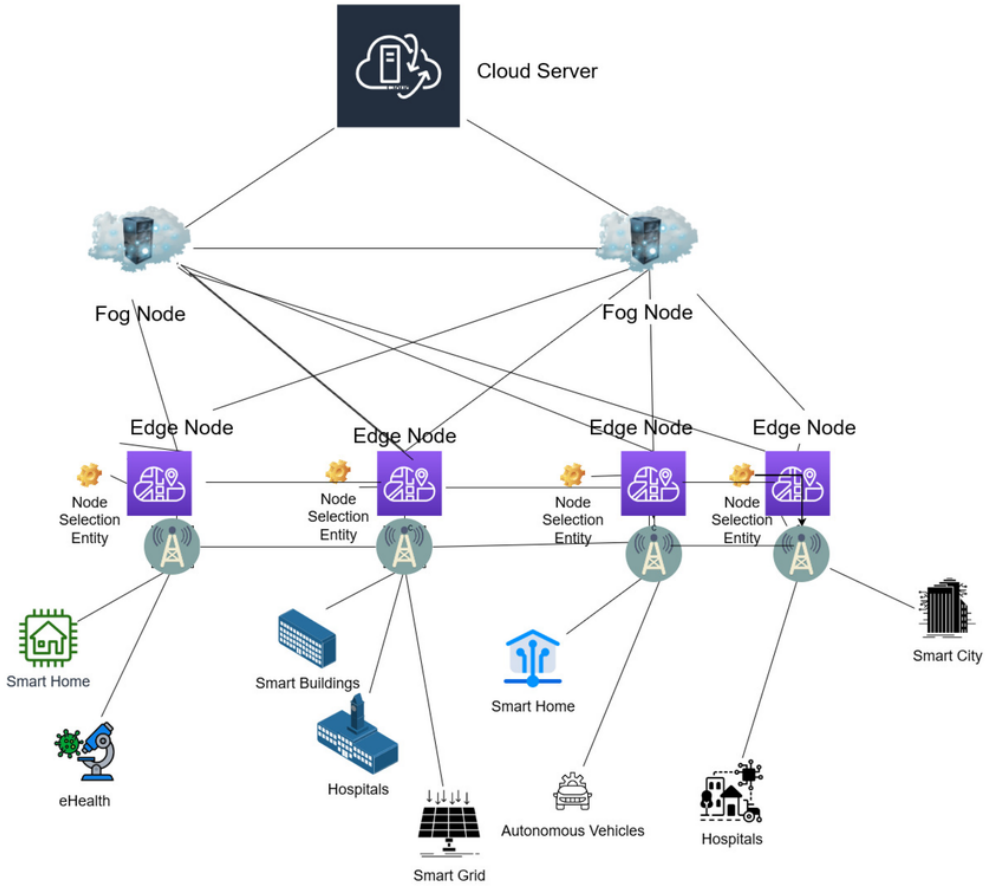



Figure 2.1: Layered architecture of computing paradigms: Cloud, Fog, Edge, and Mist .Reprinted from [13]

Together, these paradigms form a hierarchical and cooperative computing framework that enables seamless distribution of workloads across the cloud-to-edge continuum. Understanding their roles and interactions is essential for designing efficient, resilient, and scalable architectures in next-generation distributed systems.

## 2.2 Simulation and Modeling of Edge Computing Systems

To evaluate the performance, behavior, and efficiency of edge computing systems under various scenarios, several simulation tools have been developed. These simulators differ

in terms of abstraction levels, scalability, modeling granularity, supported features (e.g., mobility, energy, orchestration), and ease of integration with real-world parameters. Choosing the appropriate simulator depends on the specific requirements of the study, such as whether the focus is on microservice orchestration, latency analysis, dynamic topologies, or resource management. Below is an overview of widely adopted simulation frameworks for edge and fog computing research:

### 2.2.1 iFogSim

Developed on top of CloudSim, iFogSim is one of the most established and widely used simulation platforms for fog and edge computing. It enables modeling of latency-sensitive applications by simulating data flows between devices, edge nodes, and the cloud. The simulator supports hierarchical network structures and basic microservice modeling. However, it does not support user mobility and only provides limited support for energy consumption modeling[9].

### 2.2.2 EdgeCloudSim

EdgeCloudSim builds on CloudSim by incorporating mobility models. It provides support for energy consumption modeling, making it suitable for dynamic edge scenarios. However, it does not support microservice-based application architecture, focusing instead on monolithic application models[31].

### 2.2.3 YAFS (Yet Another Fog Simulator)

YAFS is a Python-based simulator offering high flexibility and extensibility. It supports graph-based network modeling and dynamic topologies. It enables microservice-based modeling, but lacks built-in modules for user mobility or energy consumption. Its extensibility allows for custom additions, but such features are not available out-of-the-box[2].

### 2.2.4 PureEdgeSim

PureEdgeSim is a modern simulator that provides full support for microservice architecture, realistic user mobility, and detailed energy consumption modeling. It is particularly well-suited for edge-native architectures, where services are distributed dynamically based on context. Its extensibility makes it ideal for research involving dynamic orchestration and energy-aware strategies[1].

### 2.2.5 LEAF (Lightweight Edge/Fog Simulator)

LEAF is designed for scalable, rapid simulation using an event-driven model. According to its capabilities, it supports microservice modeling, user mobility, and energy consumption, making it effective for large-scale experiments involving orchestration strategies. Although it is lightweight and fast, it also maintains coverage of essential edge computing concerns[3].

### 2.2.6 SimEdge

SimEdge is a real-time simulator for hybrid edge-cloud environments. It offers comprehensive support for microservice applications, mobility-aware orchestration, and detailed energy modeling. It is particularly suitable for evaluating dynamic workload distribution and latency-sensitive services across edge and cloud layers[17].

## 2.3 Discussion

The choice of PureEdgeSim as the simulation platform for this study was driven by its advanced capabilities, which align directly with the goals of the research. Unlike older simulators that lack features for microservices or mobility, PureEdgeSim is built to address the modern challenges in edge computing.

It provides native support for microservice-based application architecture, enabling the decomposition of applications into modular, independently orchestrated services. This matches the research's focus on dynamic workload partitioning using heuristic algorithms.

In addition, PureEdgeSim supports user mobility modeling, which allows tracking users' movements and evaluating service performance under dynamic environments—crucial for maintaining low latency and availability in real-time edge scenarios.

It also includes accurate energy consumption modeling, allowing the study to measure trade-offs among energy usage, transmission cost, and service latency. The simulator's extensible design enables integration of customized modules such as the proposed orchestration algorithm.

Finally, its discrete event simulation (DES) architecture is well-suited for event-driven behavior in edge environments, including service migration, user mobility, and node failure, making simulations more efficient and realistic.

Table 2.1: Comparison of Edge Computing Simulators

| Simulator | Support for Microservice Architecture | Support for User Mobility | Support for Energy Consumption |
|---|---|---|---|
| iFogSim | Yes | No | Yes |
| EdgeCloudSim | No | Yes | Yes |
| YAFS | Yes | No | No |
| PureEdgeSim | Yes | Yes | Yes |
| LEAF | Yes | Yes | Yes |
| SimEdge | Yes | Yes | Yes |

## 2.4 Conclusion

In conclusion, edge computing represents a transformative shift in computational paradigms, offering significant advantages for latency-sensitive applications and resource-constrained environments. The various models—cloud computing, fog computing, MEC, and others—work in harmony to create a decentralized, scalable, and flexible architecture that can meet the diverse needs of modern distributed systems. By using advanced simulators such as PureEdgeSim, researchers can accurately model and evaluate edge computing scenarios, incorporating critical factors like microservice architecture, mobility, and energy consumption. This approach helps design more efficient and resilient systems that are well-suited for the dynamic nature of edge environments [33], thus advancing the field of edge computing and enabling new technological breakthroughs.

# Chapter 3

# Related Work

In this Chapter, we will discuss the related work. We will start by providing...

## 3.1 Micro-Services Orchestration in Edge Computing

Numerous studies have examined workload orchestration within edge computing environments, driven by the need to address the unique challenges posed by distributed, resource-constrained settings[23, 10]. The primary focus of these studies has often been on static or heuristic-based scheduling approaches, designed to allocate computational resources efficiently[35, 29]. These approaches were originally developed with monolithic applications in mind—where applications are deployed as a single unit with fixed resource requirements and interdependencies. While these methods were sufficient for simpler, traditional applications, they face significant limitations when applied to more modern, microservice-based architectures[24].

With the growing adoption of microservice-based applications in edge computing, the need for dynamic and context-aware orchestration strategies has become increasingly apparent[15]. Microservices are inherently more flexible, modular, and independently scalable than monolithic applications. However, this modularity introduces additional complexities in resource management, particularly in highly dynamic environments like edge computing, where factors such as node heterogeneity, user mobility, and network variability continuously change[23, 16]. As a result, traditional scheduling approaches that assume static application structures and fixed dependencies are often inadequate for addressing the demands of these new architectures.

Recent research has led to the proposal of various scheduling and resource allocation algorithms designed to handle the challenges of modern edge computing environments. Many of

these algorithms rely on heuristic techniques, which provide efficient, approximate solutions to complex problems where exact solutions are computationally expensive or infeasible[35, 19]. Heuristic methods, such as genetic algorithms, greedy algorithms, and simulated annealing, have been applied to optimize key performance metrics such as latency, energy consumption, and resource utilization[19, 29]. These techniques offer promising results in terms of improving performance, but they often fall short in managing composite applications—which are composed of multiple interdependent services that require adaptive, real-time decisions[21].

## 3.2 Discussion

A key limitation of many existing heuristic approaches is their lack of flexibility when it comes to managing applications with fluctuating dependencies and variable workloads across heterogeneous edge nodes[24, 16]. For instance, in environments where network conditions are unpredictable or resources are intermittently available, heuristic algorithms that rely on static assumptions can lead to suboptimal performance, resulting in increased latency, inefficient energy consumption, or poor resource utilization[35]. Moreover, these approaches often fail to adapt to contextual factors such as user mobility, which can dynamically alter the optimal placement and orchestration of microservices[15].

Table 3.1: Edge Computing Orchestration Algorithms (Part 1)

| **References** | **Technique** | **Metrics** |
|---|---|---|
| [32] | Fuzzy Logic-based Workload Orchestration using EdgeCloudSim | Latency, Resource Utilization, Task Completion Rate |
| [36] | Online and Approximate Optimization, SFC, Randomized Rounding | Cost Efficiency, Resource Provisioning, Routing Performance, Theoretical Guarantee |
| [8] | Modular Scalable Architecture, Docker Orchestration, Containers, Distributed Deployment | Service Availability, Fault Tolerance, Deployment Flexibility, Performance Stability |

Table 3.2: Edge Computing Orchestration Algorithms (Part 2)

| **Reference** | **Technique** | **Metrics** |
|---|---|---|
| [7] | Genetic Algorithm, Conflict Graph Model, Task Scheduling (Parallel and Sequential Offloading) | Latency, Failure Probability, Near-Optimal Performance |

As a result, there is a pressing need for more dynamic, context-aware orchestration frameworks that can handle the complexities of modern edge applications. Such frameworks must be able to adapt in real-time to changing network conditions, user behaviors, and application states, providing efficient resource allocation and service continuity even in the face of unpredictable environmental factors[21, 15].

The orchestration of microservice-based applications in edge computing environments presents a combinatorial optimization problem of significant complexity[29]. Determining the optimal placement, partitioning, and migration of services across a dynamic, heterogeneous network must account for numerous factors such as resource constraints, service dependencies, user mobility, and fluctuating network conditions. This intricate interplay between multiple, often conflicting objectives transforms the orchestration task into an NP-hard problem[16].

The NP-hard nature of the orchestration stems from the fact that finding an optimal configuration requires evaluating an exponentially large solution space as the number of microservices and edge nodes increases. Each decision regarding service placement, replication, or migration can impact the performance of other services and the overall system behavior[19]. Exact solutions through exhaustive search become computationally infeasible even for moderately sized networks, especially in scenarios where real-time responsiveness is critical. Therefore, heuristic and metaheuristic approaches have emerged as practical alternatives, offering near-optimal solutions within acceptable timeframes[35, 10].

This complexity is analogous to the classic Knapsack Problem, where the goal is to select a subset of items with maximum total value without exceeding the weight capacity of the knapsack. In the context of edge orchestration, each microservice is like an item with specific resource demands (e.g., CPU, memory), and the edge infrastructure represents a set of knapsacks with limited capacities. Just as in the knapsack problem, selecting which services to place on which nodes must balance value (e.g., performance gain, latency reduction) against resource constraints—making the orchestration problem computationally intensive

and well-suited for heuristic optimization techniques.

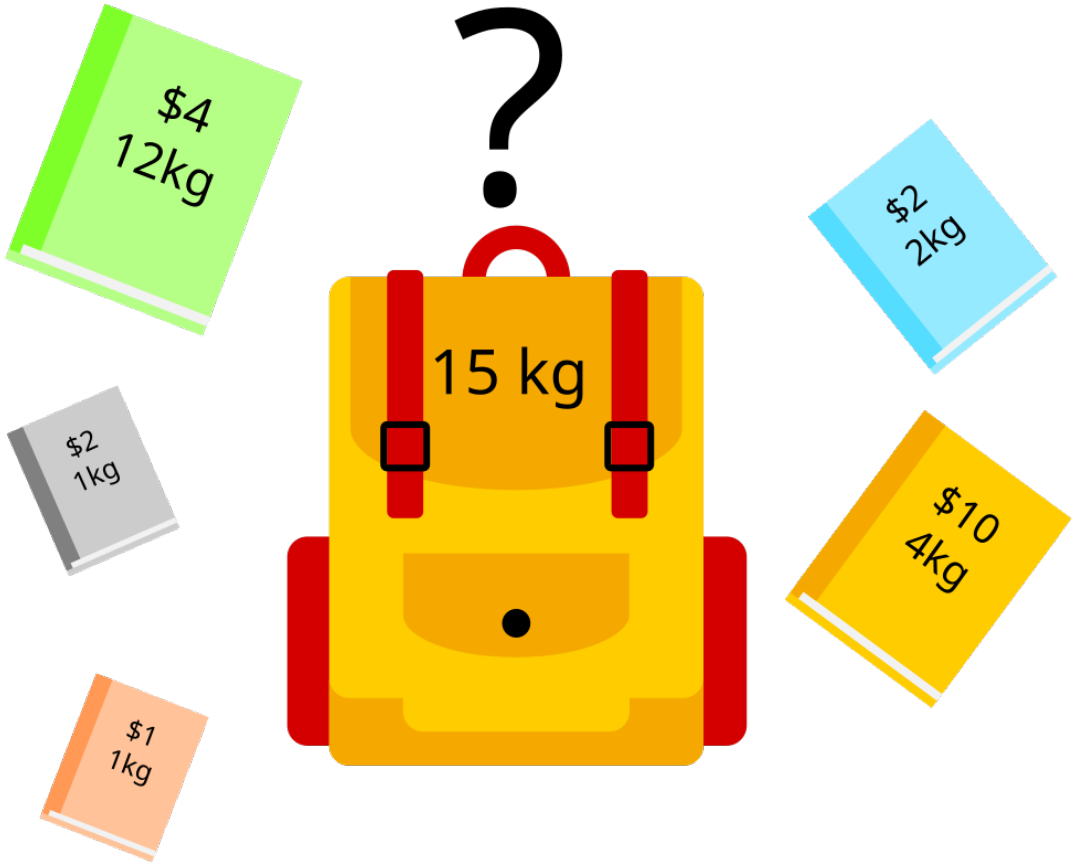



Figure 3.1: Knapsack Problem Illustration.Reprinted from[5]

While existing research efforts have contributed valuable insights into workload orchestration and resource management strategies, they often remain constrained by idealized assumptions that do not fully reflect real-world edge computing dynamics[23, 24]. Many frameworks continue to treat applications as static entities with fixed resource demands and predictable communication patterns. However, modern edge applications exhibit fluid dependencies, varying performance needs, and must contend with intermittent connectivity and user-driven mobility[21, 15].

The lack of adaptive, fine-grained orchestration mechanisms capable of dynamically partitioning and reallocating services in response to contextual changes represents a significant gap in the literature[15, 19]. In particular, there is a critical need for lightweight, heuristic-driven strategies that balance competing objectives such as latency minimization, energy

efficiency, and service continuity in a decentralized and time-sensitive manner[21, 19]. This study addresses this need by introducing a dynamic "RL-Based" workload partitioning and orchestration framework tailored specifically for microservice-based architectures operating at the edge, thus advancing the state of the art toward more resilient and context-aware edge-native systems[21, 15].

## 3.3 Conclusion

In conclusion, the orchestration of microservice-based applications in edge computing environments presents significant challenges due to the dynamic and heterogeneous nature of these systems. While traditional heuristic techniques have shown promise in optimizing performance metrics like latency and energy consumption, they often fall short in managing the complexities of modern, modular applications with fluctuating dependencies and real-time requirements[35, 24]. The research presented here addresses these challenges by proposing a dynamic partitioning and orchestration framework that leverages heuristic algorithms to adapt to changing network conditions, user mobility, and resource availability[21, 15, 19]. By focusing on dynamic, context-aware orchestration, this study contributes to advancing the development of more resilient, scalable, and efficient edge computing systems[15].

# Chapter 4

# System Architecture

The proposed system architecture is designed to address the inherent challenges of orchestrating microservice-based applications in edge computing environments. It is structured into three hierarchical layers, each serving a distinct role in the overall computational and data flow.

At the bottom, the Device Layer consists of a diverse range of IoT components, wearables, sensors, and user-facing technologies such as smartphones, tablets, and AR/VR headsets. These devices are responsible for generating continuous streams of context-rich data and triggering task execution requests. However, they suffer from limited computational power and energy constraints, making them unsuitable for handling resource-intensive operations. As such, their role is primarily to initiate tasks, which are then offloaded to more capable layers for processing.

Above the device layer lies the Edge Layer, which serves as the core computational layer in the system. It includes a distributed set of edge servers, micro data centers, and base stations, strategically deployed close to end-users. These nodes offer moderate computing and storage capabilities that enable fast, low-latency task execution and localized data handling. This layer also hosts a critical module called the Dynamic Partitioner Module, which intelligently decomposes applications into modular subtasks, assigns them to appropriate edge nodes, and manages resource orchestration dynamically. Given its proximity to data sources and its ability to respond in near-real-time, the edge layer is prioritized for executing the majority of application components, especially those requiring responsiveness and minimal delay[27].

At the top of the hierarchy is the Cloud Layer, which provides high-performance computing resources and long-term data storage capabilities. The cloud's involvement is typically reserved for scenarios that require significant processing power, such as large-scale analytics, batch operations, or recovery tasks in case of failures at the edge. By restricting cloud inter-

actions to non-time-critical processes, the system reduces dependency on wide-area networks, thereby avoiding unnecessary latency and network congestion[25].

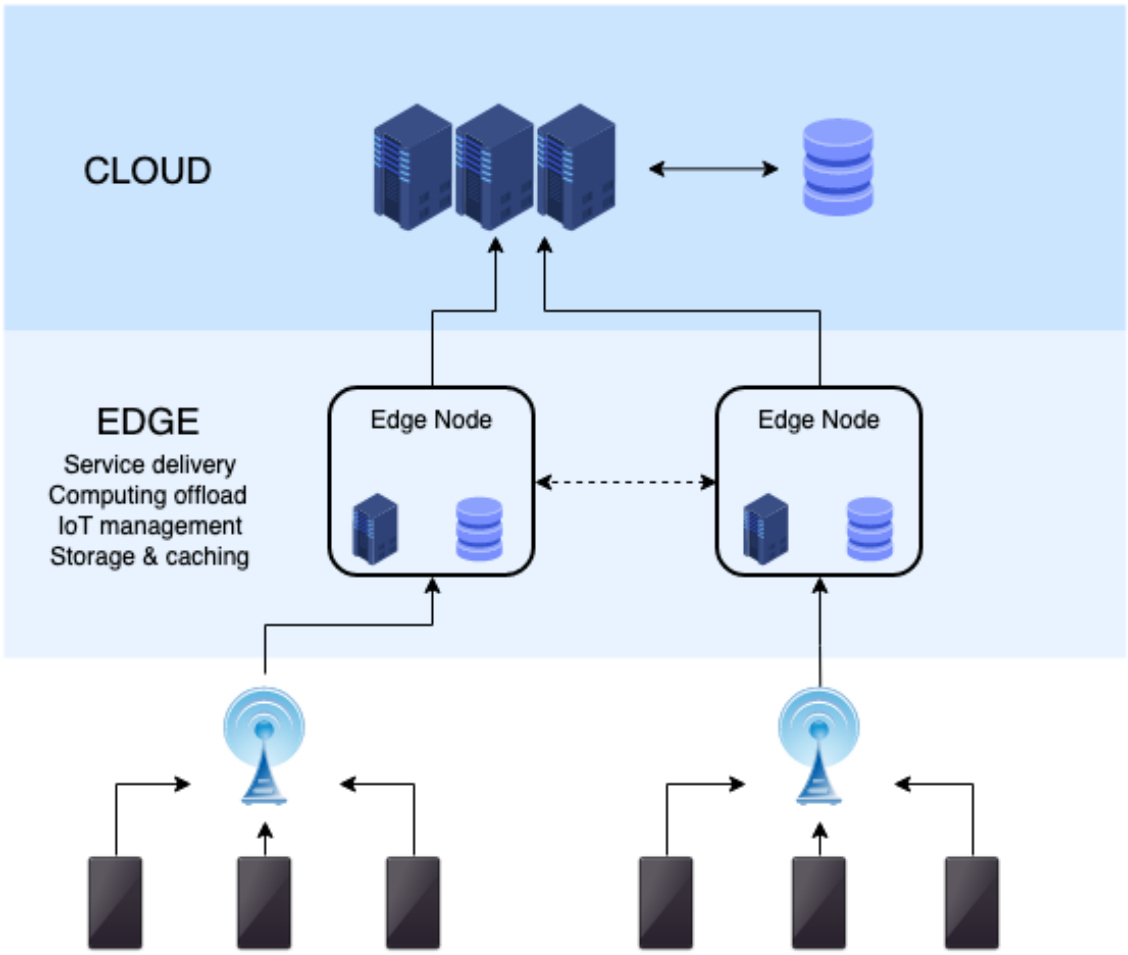



Figure 4.1: Illustration of the three-layer edge computing architecture: Device Layer, Edge Layer, and Cloud Layer.Reprinted from[4]

To efficiently orchestrate modular applications in such a volatile and heterogeneous environment, several papers have employed Genetic Algorithm [7, 11, 37, 20, 30]. In this approach, each candidate solution—representing a specific assignment of subtasks to computing nodes—is encoded as a chromosome. The fitness of each chromosome is evaluated using a weighted cost function, and the population evolves across generations through the genetic operators of selection, crossover, and mutation. A distinguishing feature of GeneticOrch is its adaptive capability : it re-evaluates deployment plans dynamically in response to contextual factors such as network fluctuations, node availability, and user mobility[34].

## 4.1 Types of Edge Applications

To better understand how edge computing supports microservice-based systems, it is essential to explore the types of applications commonly deployed in such environments. These applications vary in their requirements for latency, data handling, adaptability, and reliability. The most prominent types include:

### 4.1.1 Latency-sensitive applications

These applications demand near-instantaneous response times. Even minor delays can significantly degrade performance or user experience. Examples include real-time online gaming, augmented reality (AR), virtual reality (VR), and autonomous vehicle control systems. For these applications, computation must be handled close to the data source to ensure responsiveness[27].

### 4.1.2 Context-aware applications

These systems adapt their behavior dynamically based on environmental data such as user location, movement, or biometric inputs. A common example is mobile health monitoring, where wearables track vital signs and trigger alerts in emergencies. Smart navigation apps that alter routes based on real-time traffic data also fall into this category[25].

### 4.1.3 Data-intensive applications

These applications generate or process large volumes of raw data, often too large to be sent entirely to the cloud. Edge computing enables local filtering, aggregation, or pre-processing before transmission. For instance, in video surveillance systems, edge nodes perform tasks like motion detection or facial recognition locally, reducing bandwidth usage[27].

### 4.1.4 Real-time analytics applications

Such applications analyze live data streams to enable immediate decision-making. Examples include environmental monitoring systems that detect pollution spikes and initiate mitigation actions, and smart grid systems that analyze consumption patterns and adjust energy distribution in real time[25].

### 4.1.5 Mission-critical applications

These applications operate under strict reliability and safety requirements, where failure or delay could be catastrophic. Remote telesurgery platforms and edge-enabled industrial automation systems are prime examples. These systems rely on edge computing to ensure continuous and dependable operation with minimal latency[27].

### 4.1.6 Modular microservice-based applications

These applications are designed using a modular architecture where individual services—each responsible for a specific function—can be independently deployed and orchestrated across edge and cloud nodes. Examples include smart city systems (e.g., distributed traffic monitoring, waste collection optimization), intelligent healthcare platforms (e.g., diagnostics, patient monitoring), and AR/VR pipelines where user tracking, rendering, and content streaming are separated into dedicated services[14, 12].

What follows is the formal mathematical model that underpins the ContextualMAB-Orch framework. This model formulates the task orchestration problem as a contextual multi-armed bandit, where orchestration decisions are made online based on real-time environmental context. The model defines the problem setup, context structure, action-selection policy, reward mechanism, and update strategy.

## 4.2 Mathematical Model of ContextualMAB-Orch Algorithm

### 4.2.1 Static Partitioning

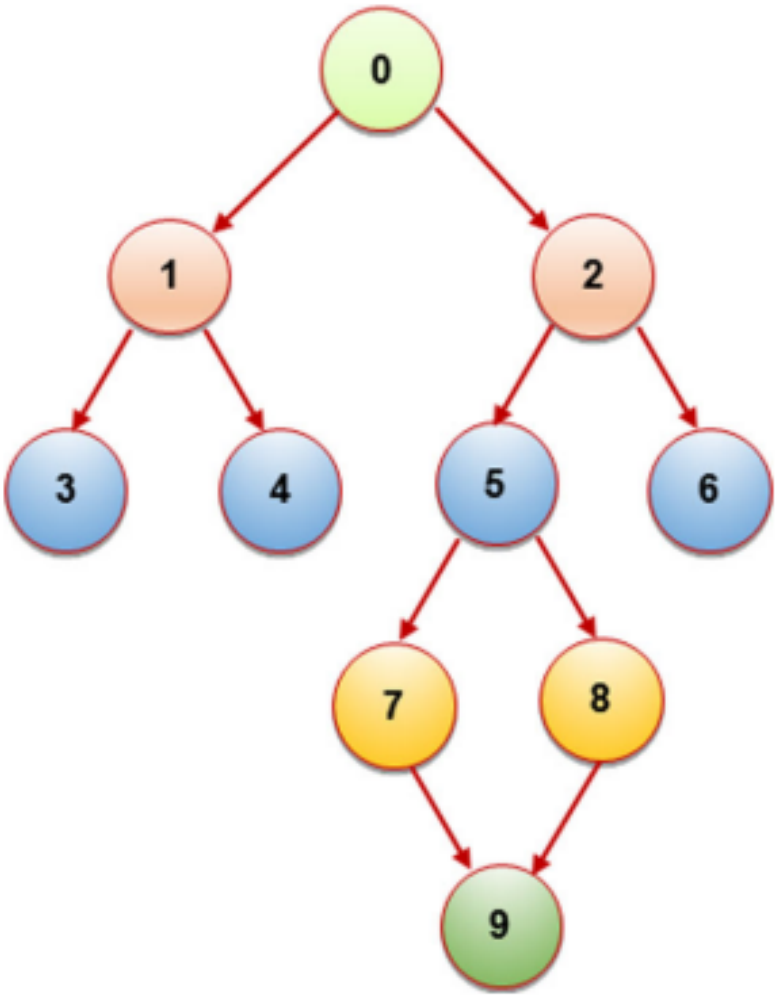



Figure 4.2: A workflow before partitioning
Reprinted from[18]

The concept of Betweenness Centrality (BC) from graph theory was used to partition workflows. This concept measures the importance of nodes (vertices) or edges in a graph based on the number of shortest paths that pass through them. In directed graphs, the goal is to minimize the total weights of the edges along these shortest paths.

To calculate BC, all shortest paths between every pair of nodes are computed, and those

passing through specific nodes or edges are summed. The edge that appears in the largest number of shortest paths is considered to have the highest centrality value and is removed to partition the workflow. The Brandes algorithm (2001) was used for this purpose, offering an efficient time complexity of $\mathcal{O}(nm)$ and space complexity of $\mathcal{O}(n+m)$, where $n$ is the number of vertices and $m$ is the number of edges.

Workflow tasks are modeled as a directed acyclic graph (DAG) $G = (V, E)$, where $V$ represents the tasks and $E$ denotes the dependencies among them. The study examined 40 workflows, each consisting of 10 tasks, with up to two edges removed per workflow for partitioning purposes.

**Example calculations:**

- Between nodes $s = 2$ and $t = 5$:
  The number of shortest paths from previous nodes to node $t$ is 1, and the number of dependencies on $t$ is 3. Thus:

$$\frac{1}{1} \times (1+3) = 4 \quad \text{from node 0 to } t$$

$$\frac{1}{1} \times (1+3) = 4 \quad \text{from node 2 to } t$$

  Total $= 4 + 4 = 8$

- Between $s = 0$ and $t = 3$:

$$\frac{1}{1} \times (1+2) = 3$$

- Between $s = 5$ and $t = 7$:

$$\frac{1}{1} \times (1+0.5) = 1.5 \quad \text{from node 0 to 7}$$

$$\frac{1}{1} \times (1+0.5) = 1.5 \quad \text{from node 2 to 7}$$

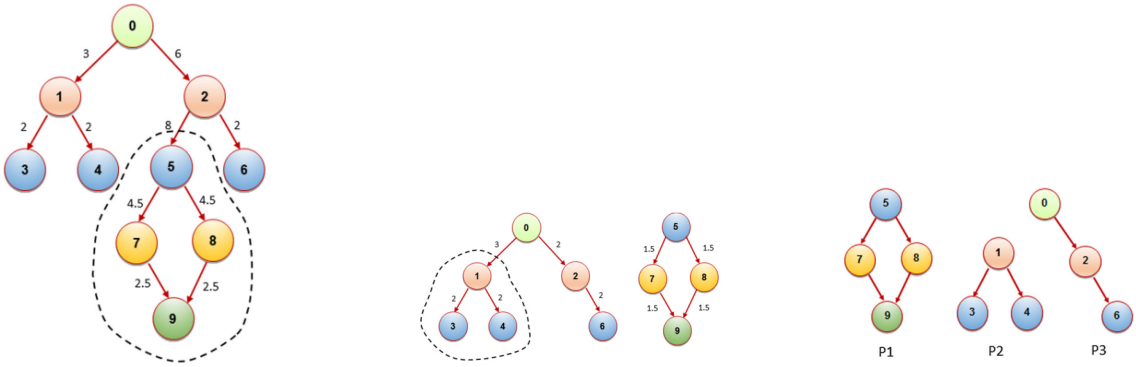
$$\frac{1}{1} \times (1+0.5) = 1.5 \quad \text{from node 5 to 7}$$

  Total $= 4.5$

**Partitioning process:**

The BC value is computed for all edges, the edge with the highest value is removed, and BC is recalculated for the resulting subgraphs. This process is repeated based on the number of edges predefined for removal.

In the example shown in Figure 4.2, the workflow was first split into two sub-workflows (Figure 4.3(b)), and then into three parts after the removal of a second edge (Figure 4.3(c)), based on the edges with the highest betweenness centrality[18].




(a) Workflow with betweenness index

(b) Two sub-workflows obtained after first partitioning of workflow

(c) Three sub-workflows after second partitioning and removal

Figure 4.3: Workflow partitioning process based on betweenness centrality. Reprinted from[18]

### 4.2.2 Contextual MAB

Let us consider a set of partitions $S = \{s_1, s_2, \ldots, s_n\}$ obtained using the above-mentioned method, and a set of computational nodes $N = \{n_1, n_2, \ldots, n_m\}$. The objective is to learn a dynamic policy that efficiently maps each subtask $s_i$ to an optimal node $n_j$ under volatile network and resource conditions. This mapping is learned over time using a contextual multi-armed bandit (MAB) strategy.

### 4.2.3 Contextual Multi-Armed Bandit Formulation

The task orchestration problem is modeled as a contextual MAB, where at each decision epoch, the orchestrator observes a context $\mathcal{C}_i$ associated with subtask $s_i$ and selects an action $a_i$ (i.e., a node assignment) from the available set of arms $A_{\mathcal{C}_i} \subseteq N$.

#### 1. Context Definition

Each context $\mathcal{C}_i$ captures relevant task and system state information, such as:

$$\mathcal{C}_i = \text{Hash}(L_i \parallel \text{CPU}_j \parallel \text{QueueLength}_j \parallel \text{NetworkLatency}_j) \qquad \text{Reprinted from [18]}$$

where $L_i$ is the workload size of subtask $s_i$, and the remaining terms represent node $n_j$'s CPU capacity, current queue length, and latency from the user.

**2. Action Selection: Epsilon-Greedy Policy**

At each decision step, an epsilon-greedy strategy is employed to balance exploration and exploitation:

$$a_i^* = \begin{cases} \text{random}(A_{\mathcal{C}_i}), & \text{with probability } \epsilon \\ \arg\max\limits_{a \in A_{\mathcal{C}_i}} Q(\mathcal{C}_i, a), & \text{with probability } 1 - \epsilon \end{cases} \qquad \text{Reprinted from [18]}$$

**3. Reward Signal**

After assigning subtask $s_i$ to node $a_i^*$, the orchestrator observes the result and assigns a binary reward:

$$r_i = \begin{cases} 1, & \text{if execution is successful and within SLA thresholds} \\ 0, & \text{otherwise} \end{cases} \qquad \text{Reprinted from [18]}$$

SLA thresholds may include maximum acceptable latency, CPU load limits, and node availability.

**4. Q-Value Update Rule**

The Q-value representing the expected reward for taking action $a$ in context $\mathcal{C}$ is updated using the temporal difference rule:

$$Q(\mathcal{C}, a) \leftarrow Q(\mathcal{C}, a) + \eta \cdot (r - Q(\mathcal{C}, a)) \qquad \text{Reprinted from [18]}$$

where $\eta$ is the learning rate ($0 < \eta \leq 1$), and $r$ is the observed reward.

### 4.2.4 Orchestration Procedure

**Algorithm 1:** ContextualMAB-Orch Algorithm

[1] Subtask set $S = \{s_1, \ldots, s_n\}$, computing nodes $N$, exploration rate $\epsilon$, learning rate $\eta$ Subtask-to-node mapping policy

each subtask $s_i \in S$ Observe context $\mathcal{C}_i$ Select node $a_i^*$ using epsilon-greedy policy

Assign $s_i$ to node $a_i^*$ Execute and observe result Compute reward $r_i$ Update Q-value:

$Q(\mathcal{C}_i, a_i^*) \leftarrow Q(\mathcal{C}_i, a_i^*) + \eta \cdot (r_i - Q(\mathcal{C}_i, a_i^*))$

Return learned Q-table as orchestration policy

### 4.2.5 Advantages of ContextualMAB-Orch

Compared to deterministic or static scheduling approaches, the ContextualMAB-Orch framework provides an adaptive, online-learning orchestration mechanism. It can efficiently handle fluctuations in user demand, mobility, and node availability without the need for prior global knowledge or exhaustive search. This makes it particularly suitable for resource-constrained and dynamic edge environments.

## 4.3 Dynamic Partitioning: Genetic Algorithm

This section describes the GeneticOrch algorithm, which leverages a Genetic Algorithm (GA) to optimize the mapping of application partitions to edge nodes. The goal is to minimize total execution delay while considering node mobility and resource availability.

Each individual $I$ in the GA population represents a candidate mapping of $n$ application partitions to $m$ available nodes:

$$I = [i_1, i_2, \ldots, i_n], \quad i_j \in \{1, 2, \ldots, m\}$$

Here, $i_j$ denotes the index of the selected node for partition $j$. The solution vector $I$ encodes one complete task-node assignment.

The fitness of an individual is computed using the inverse of a cost function that combines execution delay, mobility penalty, and edge offloading penalty:

$$\text{Fitness}(I) = \frac{1}{\sum_{j=1}^{n} (R_j + M_j + E_j)}$$

This formulation ensures that individuals with lower total penalties achieve higher fitness scores and are more likely to be selected for reproduction.

### 4.3.1 Execution Time Reward

The execution delay $R_j$ for partition $j$ is calculated by summing the execution times of its subtasks on the assigned nodes:

$$R_j = \sum_{k=1}^{t_j} \frac{L_{jk}}{\text{MIPS}_{d_{jk}}}$$

Where:

- $t_j$ is the number of subtasks within partition $j$.
- $L_{jk}$ is the computational load of subtask $k$ in partition $j$.
- $\text{MIPS}_{d_{jk}}$ is the processing power (in Millions of Instructions Per Second) of the node $d_{jk}$ assigned to execute that subtask.

### 4.3.2 Mobility Penalty

The mobility penalty $M_j$ discourages assigning tasks to mobile nodes when non-mobile alternatives are available:

$$M_j = \begin{cases} 1 & \text{if a non-mobile alternative exists with sufficient resources} \\ 0 & \text{otherwise} \end{cases}$$

This penalty reflects the risk of service disruption due to node movement.

### 4.3.3 Edge Offloading Penalty

The edge offloading penalty $E_j$ penalizes the selection of nodes with higher communication delays when better options exist:

$$E_j = \begin{cases} 1 - \frac{1}{1+D_j} & \text{if an edge device alternative exists with sufficient resources} \\ 0 & \text{otherwise} \end{cases}$$

Where $D_j$ represents the estimated communication delay between the task and the edge node. The penalty increases with delay, promoting selections with lower latency.

### 4.3.4 Maximum Required Resource

This metric determines the peak resource requirement for a given partition:

$$\text{MaxResource}_j = \max_{k=1}^{t_j} \left( \frac{L_{jk}}{T_{jk}} \right)$$

Where $T_{jk}$ denotes the estimated execution time of subtask $k$ under ideal conditions. This helps to filter out nodes incapable of handling the peak demand.

### 4.3.5 Crossover

The crossover operation generates a new individual (child) by combining genes from two parent solutions:

$$\text{Child} = [i_1^{(p1)}, \ldots, i_c^{(p1)}, i_{c+1}^{(p2)}, \ldots, i_n^{(p2)}]$$

Where $c$ is the crossover point, and $p1$ and $p2$ refer to the first and second parent, respectively. The first $c$ genes come from parent 1, and the remaining from parent 2.

### 4.3.6 Mutation

Mutation introduces diversity into the population by randomly altering genes with a small probability $\mu$:

$$i_j = \text{rand}(1, m) \quad \text{if rand}() < \mu$$

This means that with probability $\mu$, the assigned node for partition $j$ is randomly reassigned to a different node, encouraging exploration of the solution space.

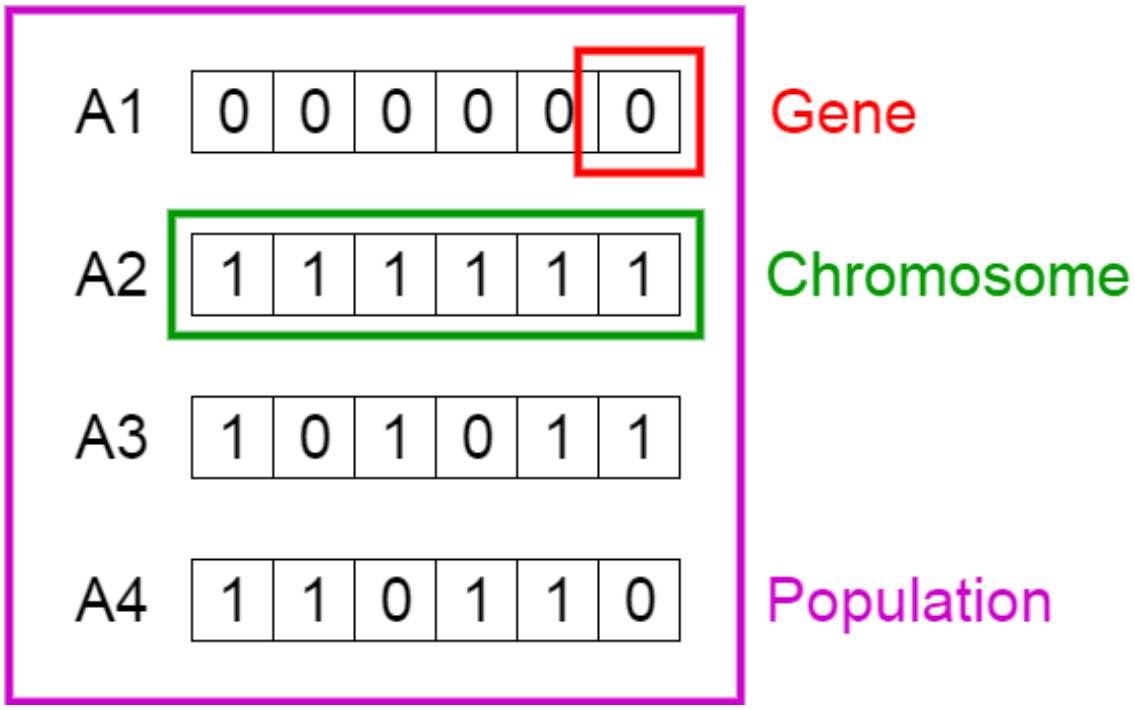



Figure 4.4: Illustration of genetic algorithm components: gene, chromosome, and population.Reprinted from[28]

### 4.3.7 Pseudocode

**Algorithm 2:** Genetic Offloading Orchestration

[1] **Input:** Task partitions $P = \{p_1, p_2, \ldots, p_n\}$, available nodes $N = \{n_1, n_2, \ldots, n_m\}$ **Parameters:** Population size $popSize$, crossover rate $c_r$, mutation rate $\mu$, max generations $maxGen$ Initialize population with random individuals (mappings of partitions to nodes) **for** *generation = 1 to maxGen* **do** Evaluate fitness of each individual in population using:

$$\text{Fitness}(I) = \frac{1}{\sum_{j=1}^{n} (R_j + M_j + E_j)}$$

Select parents based on fitness (e.g., roulette wheel selection) Apply crossover to parents with probability $c_r$ to create offspring Apply mutation to offspring genes with probability $\mu$ Form new population with offspring and possibly elite individuals **return** Best individual $I^*$ representing the optimized partition-node mapping

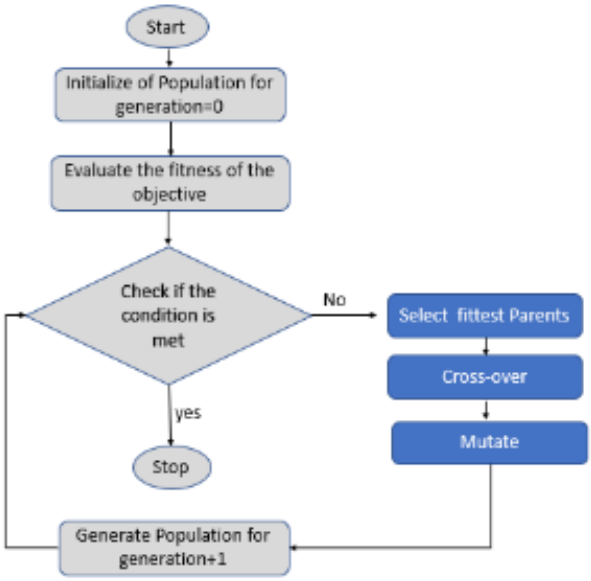



Figure 4.5: GeneticOrch Algorithm Workflow. A diagram illustrating the main steps of the GeneticOrch heuristic, including population initialization, fitness evaluation, selection, crossover, mutation, and generation update.Reprinted from[28]

## 4.4 Conclusion

In this chapter, we presented a comprehensive view of the proposed system architecture tailored for edge computing environments that demand dynamic, context-aware, and efficient orchestration of modular microservice applications. The architecture is structured into three hierarchical layers—Device, Edge, and Cloud—each contributing distinct capabilities to meet latency, resource, and scalability demands.

We further introduced two heuristic-based orchestration strategies: the ContextualMAB-Orch, which leverages online learning to adapt task placement in real time using contextual multi-armed bandits; and the GeneticOrch algorithm, which employs evolutionary optimization to discover optimal task-to-node mappings while accounting for delay, mobility, and communication penalties. Both models were detailed with formal mathematical formulations, algorithmic procedures, and workflow diagrams.

The following chapter will delve into the experimental design, and simulation results used to evaluate the effectiveness and performance of the proposed algorithm in various edge computing scenarios.

# Chapter 5

# Simulation Results

This chapter presents the experimental evaluation of the proposed orchestration algorithms within an edge computing environment. It aims to analyze and compare the performance of two primary algorithms: **ContextualMAB** and **GeneticOrch**, under various task execution scenarios and system configurations.

To ensure comprehensive analysis, different application models are simulated, including **sequential**, **concurrent**, and **complex** task dependencies. Critical factors such as latency, task size, device mobility, and resource constraints are taken into account. The simulation environment replicates realistic conditions using a scalable network of edge devices and heterogeneous data centers.

Performance evaluation is organized around several key metrics, including:

- **Task Success Rate**, reflecting the system's ability to complete tasks reliably.
- **Failure Analysis**, identifying causes such as latency violations or unmet task requirements.
- **Execution Delay**, highlighting each algorithm's responsiveness.
- **Time Complexity**, providing insights into computational efficiency.

Through tables, visual figures, and analytical discussion, this chapter aims to highlight the strengths and limitations of each algorithm. The results demonstrate the effectiveness of **ContextualMAB** as a robust and efficient solution for orchestrating interdependent tasks in dynamic edge computing environments.

## 5.1 Simulation Scenario

The table below presents the characteristics of a set of application tasks, which must be executed either sequentially, concurrently, or based on a more complex dependency structure. Each row represents a specific property for tasks numbered from 0 to 9.

**Latency:** This row indicates the time required to complete each task, measured in seconds. The values vary across tasks, reflecting differences in execution durations. For instance, Task 0 requires only 0.02 seconds, while Task 9 takes 0.08 seconds.

**Task Length:** This row represents the computational load or resource demand associated with each task. For example, Task 2 has a length of 20000 units, indicating a higher processing burden compared to Task 0, which has a length of only 3000 units.

**Require (Sequential):** This row shows a strict linear dependency, where a task cannot begin until its immediate predecessor has completed. For example:

- Task 1 requires Task 0,
- Task 2 requires Task 1,
- ...
- Task 9 requires Task 8.

**Require (Concurrent):** This row illustrates a scenario in which multiple tasks can begin after the completion of a single task. For example:

- Tasks 2 through 8 can start as soon as Task 1 is finished,
- Task 9, however, depends on the completion of Tasks 2 through 8.

**Require (Complex):** This row presents a more intricate dependency structure with branching and overlapping requirements. For instance:

- Tasks 1 and 2 depend on Task 0,
- Tasks 3 and 4 depend on Task 1,
- Tasks 5 and 6 depend on Task 2,
- Tasks 7 and 8 depend on Task 5,
- Task 9 depends on both Tasks 7 and 8.

In summary, this table offers a comprehensive overview of different task execution models—sequential, concurrent, and complex—highlighting variations in latency and resource demands. These classifications are essential for evaluating scheduling strategies and optimizing task coordination in application systems.

Table 5.1: Application characteristics

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| Latency | 0.02 | 0.04 | 0.02 | 0.03 | 0.03 | 0.03 | 0.07 | 0.04 | 0.05 | 0.08 |
| Task length | 3000 | 10000 | 20000 | 5000 | 5000 | 5000 | 15000 | 10000 | 12000 | 5000 |
| Require (Sequential) | / | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| Require (Concurrent) | / | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 2,3,4,5,6,7,8 |
| Require (Complex) | / | 0 | 0 | 1 | 1 | 2 | 2 | 5 | 5 | 7,8 |

The table below presents the characteristics of a set of data centers, labeled from dc1 to dc6, where each row describes a specific resource attribute available at each data center.

**Cores:** This row indicates the number of processing cores available in each data center. As shown, the number of cores varies across data centers, ranging from 10 to 20 cores. This reflects a difference in parallel processing capabilities among the centers.

**MIPS (Million Instructions Per Second):** This row shows the processing power of each data center, measured in MIPS. All data centers are equipped with a processing capacity of 400,000 MIPS, indicating their ability to handle high-performance computing tasks at the same execution speed.

**Summary:** The table demonstrates that while all data centers share identical execution speed (MIPS), they differ in the number of processing cores, which may influence workload distribution and task scheduling in a distributed cloud computing environment.

Table 5.2: Data Center Characteristics

| | dc1 | dc2 | dc3 | dc4 | dc5 | dc6 |
|---|---|---|---|---|---|---|
| Cores | 10 | 15 | 20 | 12 | 10 | 15 |
| Mips | 400000 | 400000 | 400000 | 400000 | 400000 | 400000 |

The table below presents the simulation parameters used to configure the experimental environment for evaluating system performance in an edge computing context. Each parameter is explained as follows:

**Simulation Time:** Indicates the total duration of the simulation, set to 5 minutes. This defines how long the simulated environment runs to observe system behavior.

**Simulation Map:** Represents the spatial area covered by the simulation, measuring 2000 × 2000 meters. This defines the geographical space in which devices are distributed.

**Min Number of Devices:** Specifies the minimum number of devices participating in the simulation, set at 200 devices.

**Max Number of Devices:** Specifies the maximum number of devices that can be added to the simulation, set at 800 devices.

**Number of Devices Added at Each Step:** Indicates how many devices are introduced into the simulation at each incremental step, which is 100 devices.

**Architecture:** Refers to the system architecture used in the simulation, which is Edge architecture, where data is processed closer to the source (i.e., at the network edge).

**Algorithms:** Lists the orchestration algorithms employed during the simulation:

- GeneticOrch: An algorithm based on genetic optimization techniques to efficiently allocate resources.
- ContextualMAB: A decision-making algorithm based on the Contextual Multi-Armed Bandit model, used to make intelligent choices depending on varying contextual factors.

In summary, this table defines the experimental simulation setup used to test task distribution and management strategies in an edge computing environment, leveraging advanced algorithms and a scalable device infrastructure.

Table 5.3: Simulation Parameters

| Parameter | Value |
|---|---|
| Simulation Time | 5 minutes |
| Simulation Map | 2000 * 2000 m |
| Min number of device | 200 |
| Max number of device | 800 |
| Number of devices added at each step | 100 |
| Architecture | Edge |
| Algorithms | ContextualMAB ,GeneticOrch |

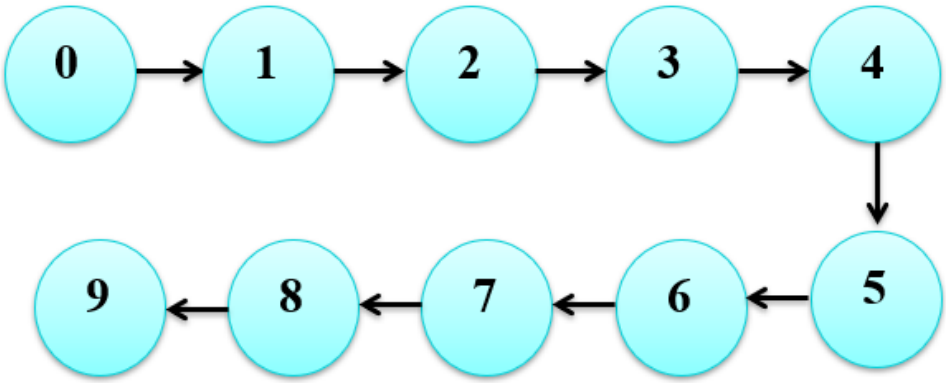



Figure 5.1: Sequential Tasks

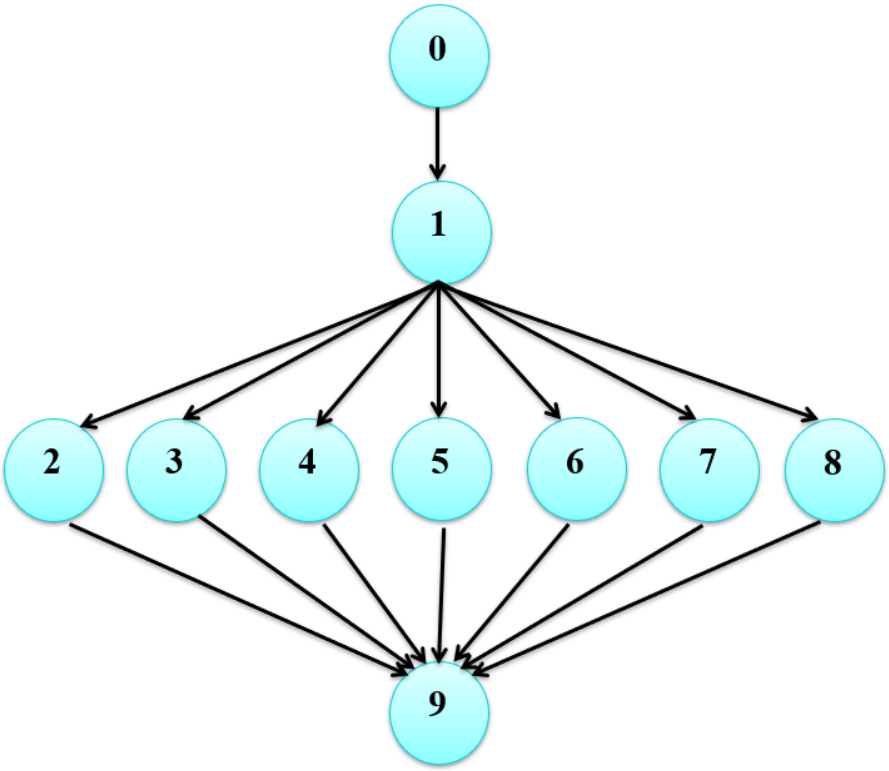



Figure 5.2: Concurrent Tasks

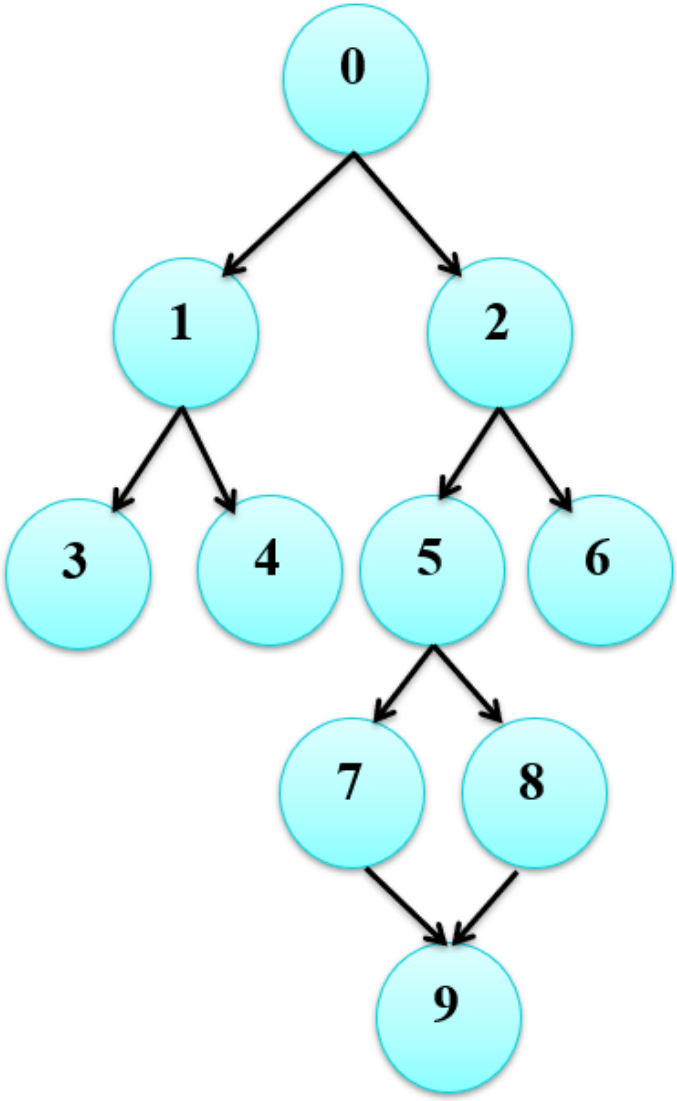
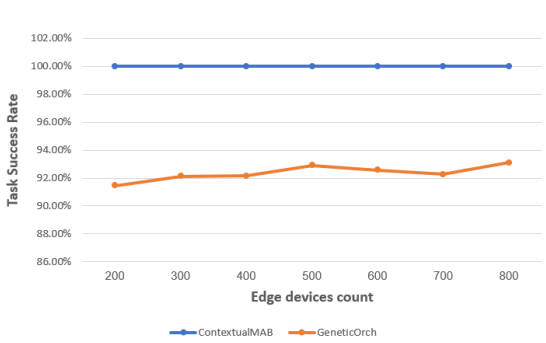


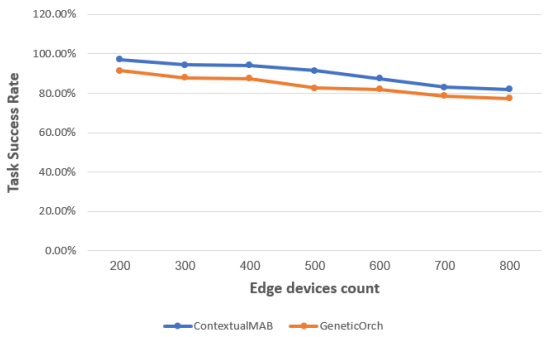

Figure 5.3: Complex Tasks

## 5.2 Tasks success rate

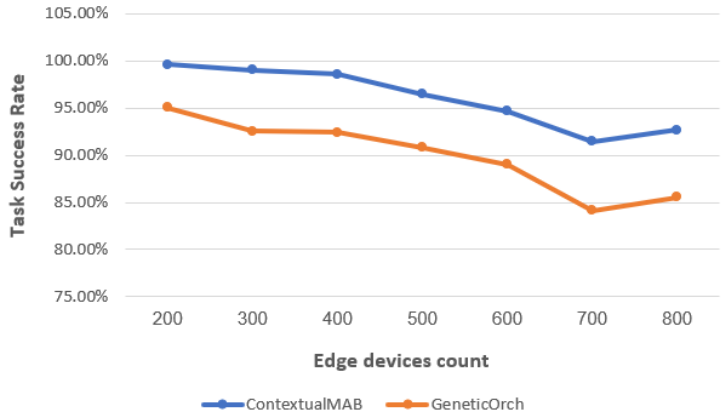
Task Success Rate Analysis To evaluate the performance of the ContextualMAB and GeneticOrch systems in managing and executing tasks across different computing environments, three primary execution scenarios were considered: Sequential execution, Concurrent execution, and a Complex environment. The results are illustrated in three charts, showing how the number of edge devices affects the Task Success Rate for each system.




(a) Sequential




(b) Concurrent




(c) Complex

Figure 5.4: Task Success Rate

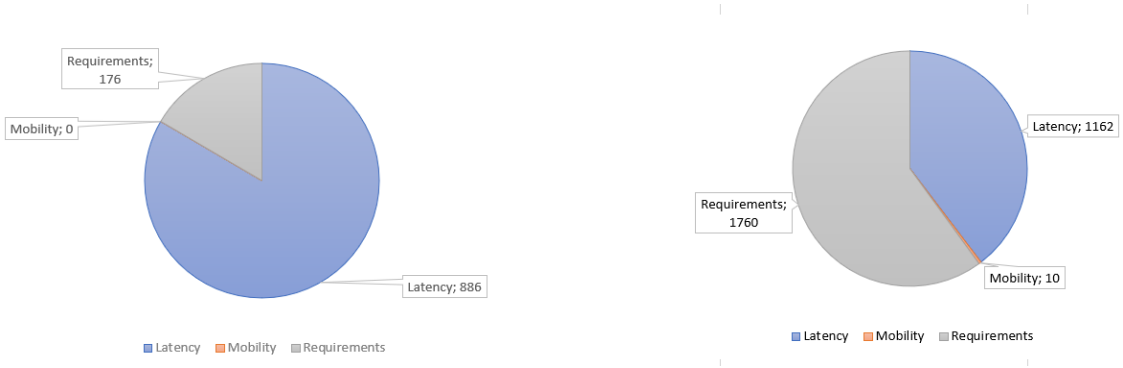
In the Sequential execution scenario ,the ContextualMAB system demonstrated ideal performance, consistently maintaining a 100

In the Concurrent execution scenario , both systems experienced a gradual decline in task success rate as the number of devices increased. ContextualMAB showed superior resilience, dropping to around 91

The Complex environment scenario posed the greatest operational challenge. Here, task success rates dropped more significantly for both systems. ContextualMAB maintained a relatively strong performance, with a minimum success rate of about 83

Overall, the results demonstrate that ContextualMAB provides higher stability and adaptability across varying execution conditions, especially as the system scales or faces increased task complexity. These attributes position it as a more robust choice for intelligent orchestration in edge computing environments.

## 5.3 failed resource




(a) ContextualMAB (b) GeneticOrch

Figure 5.5: Task failed

The following pie charts illustrate the distribution of failed tasks based on three failure causes: **Latency**, **Mobility**, and **Requirements**, using a simulated environment of **500 devices**. The comparison is made between two orchestration algorithms: **ContextualMAB** and **GeneticOrch**.

### ContextualMAB Algorithm

- Total number of failed tasks: **1062**.
- Failures due to **latency**: **886 tasks**.
- Failures due to **requirements**: **176 tasks**.
- Failures due to **mobility**: **0 tasks**.

### GeneticOrch Algorithm

- Total number of failed tasks: **2932**.
- Failures due to **requirements**: **1760 tasks**.
- Failures due to **latency**: **1162 tasks**.
- Failures due to **mobility**: **10 tasks**.

### Analysis and Insights

- The **ContextualMAB** algorithm outperforms **GeneticOrch** by achieving a **64% reduction** in total failed tasks.
- Although **latency** is a common challenge, **ContextualMAB** demonstrates better performance in fulfilling application requirements and adapting to mobility.
- These findings suggest that **ContextualMAB** is a more efficient and robust solution for mobile edge computing environments.

## 5.4 Analyzing delay

This section presents a comparative analysis of task execution delay between the **ContextualMAB** and **GeneticOrch** approaches across three different task execution scenarios: complex, concurrent, and sequential. The analysis is based on the average delay observed as the number of edge devices increases from 200 to 800.

### 5.4.1 Sequential Task Scenario

The first chart, corresponding to the sequential task scenario, shows that **ContextualMAB** maintains a nearly constant delay (around 0.113 seconds) regardless of the number of devices. In contrast, **GeneticOrch** continues to display slight variations and generally higher delays. This scenario clearly demonstrates the superior efficiency and consistency of ContextualMAB in environments with sequential task execution.

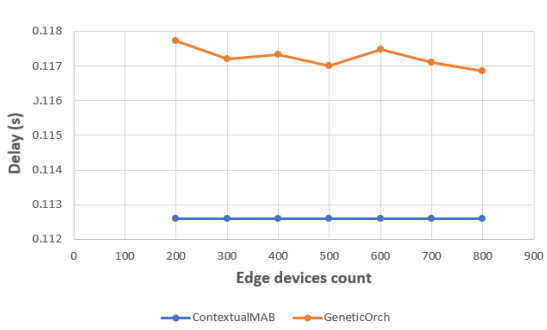
### 5.4.2 Concurrent Task Scenario

In the second chart, representing the concurrent task scenario, both techniques show an increase in average delay. **GeneticOrch** exhibits noticeable fluctuations in delay values, indicating potential instability in performance. On the other hand, **ContextualMAB** maintains a steadily increasing trend with lower delay values overall, reinforcing its reliability in handling concurrent tasks with multiple edge devices.

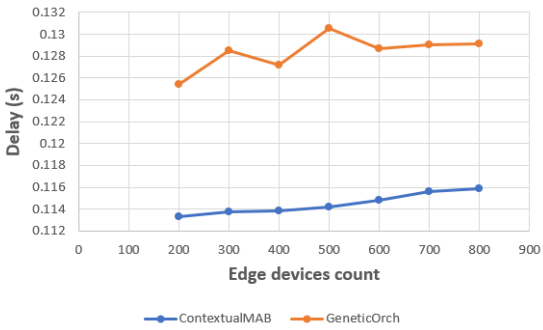
### 5.4.3 Complex Task Scenario

The third chart ,illustrates the delay under the complex task scenario. It is evident that **ContextualMAB** consistently achieves lower delay values compared to **GeneticOrch** across all edge device counts. As the number of devices increases, both techniques experience a rise in delay, but **ContextualMAB** shows a more gradual and stable increase. The largest performance gap is observed at 700 and 800 devices, highlighting the better scalability and adaptability of ContextualMAB in complex environments.

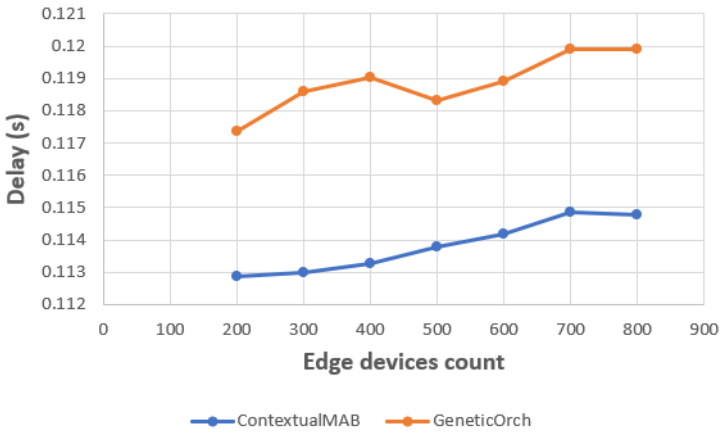
Across all three scenarios, **ContextualMAB** outperforms **GeneticOrch** in terms of delay minimization and performance stability. It proves to be more scalable and reliable, particularly in complex and concurrent task environments. Therefore, **ContextualMAB** is recommended for edge computing systems that demand low-latency and scalable task scheduling solutions.




(a) Sequential




(b) Concurrent




(c) Complex

Figure 5.6: Task failed delay

## 5.5 Time complexity

A comparative analysis was conducted between two algorithms with distinct computational approaches: **GeneticOrch**, based on genetic algorithms, and **ContextualMAB**, which relies on the contextual multi-armed bandit model. The execution time of each algorithm was measured under a unified testing environment to assess their time efficiency.

The results showed that GeneticOrch required 8 time units to complete execution, whereas ContextualMAB needed only 1 time unit. This substantial difference in runtime performance (an 800

This discrepancy can be attributed to the nature of genetic algorithms, which involve iterative processes simulating selection, crossover, and mutation—mechanisms that inherently demand more computational time. In contrast, ContextualMAB relies on simpler, more computationally efficient learning models, which directly translates into reduced execution time.

Therefore, ContextualMAB demonstrates a clear advantage in terms of time efficiency, particularly in scenarios with limited computational resources or real-time decision-making requirements. Conversely, GeneticOrch may still be preferable in contexts that demand extensive solution space exploration, despite its higher execution cost.

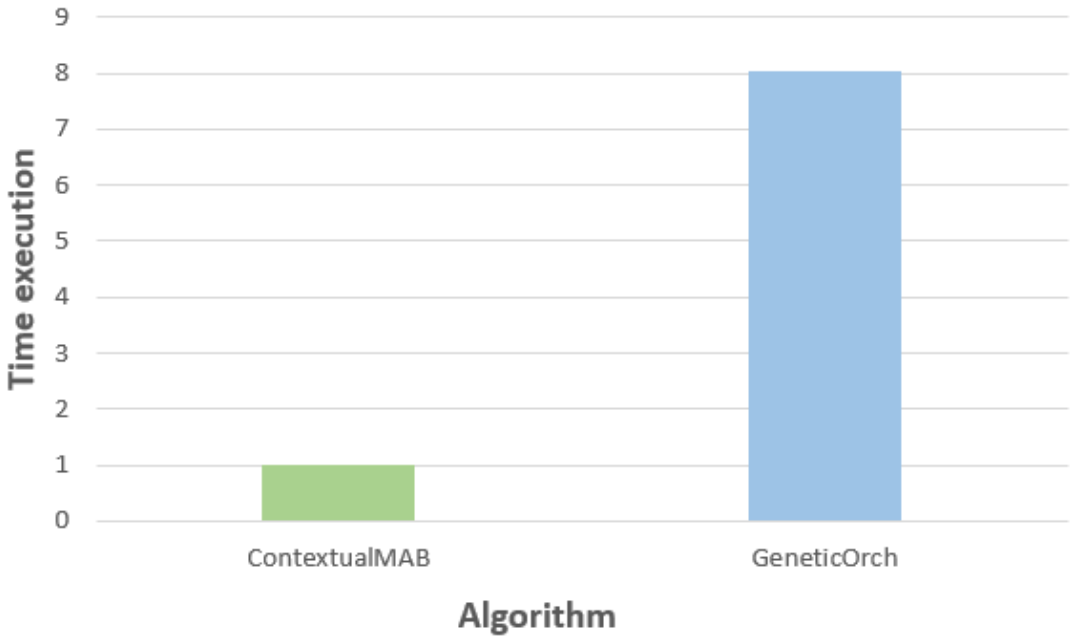



Figure 5.7: Time complexity

## 5.6 Conclusion

In this chapter, various scenarios were presented to simulate task execution within an edge computing environment, using different sets of tasks that vary in execution time, required resources, and inter-task relationships—whether sequential, concurrent, or complex. The characteristics of the participating data centers were also detailed, along with the adopted simulation settings, including the number of devices, geographical distribution, and total execution time.

The results showed that the ContextualMAB algorithm outperformed others in terms of task execution success rate across different scenarios, particularly in complex environments, where it maintained relatively stable performance. In contrast, the GeneticOrch algorithm experienced a noticeable performance drop as task complexity or the number of connected devices increased.

These findings highlight the importance of selecting an appropriate orchestration algorithm based on the nature of tasks and the available infrastructure. They also emphasize the effectiveness of context-aware intelligent algorithms in managing edge computing resources and ensuring quality of service, even under complex operational challenges.

# Chapter 6

# General Conclusion

With the rapid advancement of Internet of Things (IoT) technologies and smart applications, the need has grown for computing solutions capable of real-time responsiveness and efficient processing of the ever-increasing data generated at the network edge. Edge computing has emerged as a strategic alternative to overcome the limitations of traditional cloud computing, particularly in terms of latency reduction and real-time decision-making.

In this research, we addressed the challenge of orchestrating interdependent tasks in edge computing environments, with a focus on microservice-based applications. We proposed a dynamic orchestration mechanism based on the Contextual Multi-Armed Bandit (ContextualMAB) algorithm, which is known for its ability to adaptively learn from environmental context and make task placement decisions based on factors such as network delay, processing capacity, and resource availability.

The proposed model was evaluated using the PureEdgeSim simulation tool, which provides a realistic environment for modeling edge computing systems, including device mobility and energy consumption. The simulation results demonstrated the effectiveness of the proposed algorithm in:

Reducing end-to-end latency,

Improving the task success rate,

And adapting to dynamic changes in the network and resource conditions.

These outcomes confirm that context-aware learning algorithms like ContextualMAB offer a promising approach to building intelligent edge systems capable of making efficient decisions in complex and constantly changing environments.

In the future, this work can be extended by integrating more advanced learning techniques, such as deep contextual learning, or by validating the model in real-world, multi-user edge environments to further assess its scalability and robustness.

# Bibliography


[1] GitHub - CharafeddineMechalikh/PureEdgeSim: PureEdgeSim: A simulation framework for performance evaluation of cloud, fog, and pure edge computing environments.

[2] acsicuib/YAFS, May 2025. original-date: 2018-02-27T09:23:01Z.

[3] dos-group/leaf, May 2025. original-date: 2021-01-22T17:00:06Z.

[4] Edge computing, June 2025. Page Version ID: 1295096943.

[5] Knapsack problem, May 2025. Page Version ID: 1290040011.

[6] Ala Al-Fuqaha, Mohsen Guizani, Mehdi Mohammadi, Mohammed Aledhari, and Moussa Ayyash. Internet of things: A survey on enabling technologies, protocols, and applications. 17(4):2347–2376.

[7] Ahmed A. Al-Habob, Octavia A. Dobre, Ana García Armada, and Sami Muhaidat. Task Scheduling for Mobile Edge Computing Using Genetic Algorithm and Conflict Graphs. *IEEE Transactions on Vehicular Technology*, 69(8):8805–8819, August 2020.

[8] Muhammad Alam, Joao Rufino, Joaquim Ferreira, Syed Hassan Ahmed, Nadir Shah, and Yuanfang Chen. Orchestration of Microservices for IoT Using Docker and Edge Computing. *IEEE Communications Magazine*, 56(9):118–123, September 2018.

[9] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh 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*, pages 13–16, Helsinki Finland, August 2012. ACM.

[10] Lucas Fernando Souza de Castro and Sandro Rigo. Relating Edge Computing and Microservices by means of Architecture Approaches and Features, Orchestration, Choreography, and Offloading: A Systematic Literature Review, January 2023. arXiv:2301.07803 [cs].

[11] Sheuli Chakraborty and Kaushik Mazumdar. Sustainable task offloading decision using genetic algorithm in sensor mobile edge computing. *Journal of King Saud University - Computer and Information Sciences*, 34(4):1552–1568, April 2022.

[12] Lixing Chen and Jie Xu. Budget-constrained Edge Service Provisioning with Demand Estimation via Bandit Learning, March 2019. arXiv:1903.09080 [cs].

[13] Rolden Fereira, Chathurika Ranaweera, Kevin Lee, and Jean-Guy Schneider. Energy efficient node selection in edge-fog-cloud layered IoT architecture. 23(13):6039. Number: 13 Publisher: Multidisciplinary Digital Publishing Institute.

[14] Carlos Guerrero, Isaac Lera, and Carlos Juiz. Genetic Algorithm for Multi-Objective Optimization of Container Allocation in Cloud Architecture. *Journal of Grid Computing*, 16(1):113–135, March 2018. arXiv:2401.12698 [cs].

[15] Md. Delowar Hossain, Tangina Sultana, Sharmen Akhter, Md Imtiaz Hossain, Ngo Thien Thu, Luan N. T. Huynh, Ga-Won Lee, and Eui-Nam Huh. The role of microservice approach in edge computing: Opportunities, challenges, and research directions. *ICT Express*, 9(6):1162–1182, December 2023.

[16] Raj Jain. A Survey of Edge Management and Orchestration in Edge Computing.

[17] Johannes Kässinger, Heiko Trötsch, Frank Dürr, and Janick Edinger. Simedge: Towards accelerated real-time augmented reality simulations using adaptive smart edge computing. In *Proceedings of the Int'l ACM Conference on Modeling Analysis and Simulation of Wireless and Mobile Systems*, MSWiM '23, page 181–190, New York, NY, USA, 2023. Association for Computing Machinery.

[18] Mandeep Kaur, Sanjay Kadam, and Naeem Hannoon. RETRACTED ARTICLE: Multi-level parallel scheduling of dependent-tasks using graph-partitioning and hybrid approaches over edge-cloud. *Soft Computing*, 26(11):5347–5362, June 2022.

[19] Yujie Li, Yaoyao Xu, Fangfang Cao, and Xiang He. A meta-heuristic optimisation algorithm based method for scheduling edge computing resources. *International Journal of Information and Communication Technology*, 25(9):88–103, 2024.

[20] Zhi Li and Qi Zhu. Genetic Algorithm-Based Optimization of Offloading and Resource Allocation in Mobile-Edge Computing. *Information*, 11(2):83, February 2020. Number: 2 Publisher: Multidisciplinary Digital Publishing Institute.

[21] Abdul Rasheed Mahesar, Xiaoping Li, and Dileep Kumar Sajnani. Enhancing task scheduling and QoS optimization in mobile edge computing via microservice-oriented container selection. *Computing*, 107(2):60, January 2025.

[22] Redowan Mahmud, Ramamohanarao Kotagiri, and Rajkumar Buyya. Fog Computing: A Taxonomy, Survey and Future Directions. In Beniamino Di Martino, Kuan-Ching Li, Laurence T. Yang, and Antonio Esposito, editors, *Internet of Everything: Algorithms, Methodologies, Technologies and Perspectives*, pages 103–130. Springer, Singapore, 2018.

[23] Angelo Marchese and Orazio Tomarchio. Orchestrating Microservices-Based Applications in the Cloud-to-Edge Continuum. In Maarten van Steen, Donald Ferguson, and Claus Pahl, editors, *Cloud Computing and Services Science*, pages 170–187, Cham, 2024. Springer Nature Switzerland.

[24] Lionel Nkenyereye, Boon Giin Lee, and Wan-Young Chung. Functionality-aware offloading technique for scheduling containerized edge applications in IoT edge computing. *Journal of Cloud Computing*, 14(1):13, February 2025.

[25] Mahadev Satyanarayanan. The Emergence of Edge Computing. *Computer*, 50(1):30–39, January 2017.

[26] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. Edge Computing: Vision and Challenges. *IEEE Internet of Things Journal*, 3(5):637–646, October 2016.

[27] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. Edge Computing: Vision and Challenges. *IEEE Internet of Things Journal*, 3(5):637–646, October 2016.

[28] shubham.jain. Introduction to Genetic Algorithm & their application in data science, July 2017.

[29] Rute C. Sofia, Doug Dykeman, Peter Urbanetz, Akram Galal, and Dushyant Anirudhdhabhai Dave. Dynamic, Context-Aware Cross-Layer Orchestration of Containerized Applications. *IEEE Access*, January 2023.

[30] Heekang Song, Bonjun Gu, Kyungrak Son, and Wan Choi. Joint Optimization of Edge Computing Server Deployment and User Offloading Associations in Wireless Edge Network via a Genetic Algorithm. *IEEE Transactions on Network Science and Engineering*, 9(4):2535–2548, July 2022.

[31] Cagatay Sonmez. CagataySonmez/EdgeCloudSim, May 2025. original-date: 2017-02-18T10:07:13Z.

[32] Cagatay Sonmez, Atay Ozgovde, and Cem Ersoy. Fuzzy Workload Orchestration for Edge Computing. *IEEE Transactions on Network and Service Management*, 16(2):769–782, June 2019.

[33] Tarik Taleb, Konstantinos Samdanis, Badr Mada, Hannu Flinck, Sunny Dutta, and Dario Sabella. On Multi-Access Edge Computing: A Survey of the Emerging 5G Network Edge Cloud Architecture and Orchestration. *IEEE Communications Surveys & Tutorials*, 19(3):1657–1681, 2017.

[34] Chien-Sheng Yang, Ramtin Pedarsani, and A. Salman Avestimehr. Edge Computing in the Dark: Leveraging Contextual-Combinatorial Bandit and Coded Computing, March 2021. arXiv:1912.09512 [cs].

[35] Sevda Zarei, Sadoon Azizi, and Awder Ahmed. Optimizing edge server placement and load distribution in mobile edge computing using ACO and heuristic algorithms. *The Journal of Supercomputing*, 81(1):257, December 2024.

[36] Zhi Zhou, Qiong Wu, and Xu Chen. Online Orchestration of Cross-Edge Service Function Chaining for Cost-Efficient Edge Computing. *IEEE Journal on Selected Areas in Communications*, 37(8):1866–1880, August 2019.

[37] Anqing Zhu and Youyun Wen. Computing Offloading Strategy Using Improved Genetic Algorithm in Mobile Edge Computing System. *Journal of Grid Computing*, 19(3):38, August 2021.