NERIJUS ŠATKAUSKAS

FOG COMPUTING SERVICE DYNAMIC ORCHESTRATION METHOD

DOCTORAL DISSERTATION

Kaunas
2025

KAUNAS UNIVERSITY OF TECHNOLOGY

NERIJUS ŠATKAUSKAS

FOG COMPUTING SERVICE DYNAMIC ORCHESTRATION METHOD

Doctoral dissertation
Technological Sciences, Informatics Engineering (T 007)

2025, Kaunas

The dissertation has been prepared at the Department of Computer Sciences of the Faculty of Informatics of Kaunas University of Technology in 2019-2024.

The doctoral right has been granted to Kaunas University of Technology together with Vytautas Magnus University and Vilnius Gediminas Technical University.

Research Supervisor:
Prof. Dr. Algimantas VENČKAUSKAS (Kaunas University of Technology, Technological Sciences, Informatics Engineering, T 007).

Edited by: English language editor Dr. Armandas Rumšas (Publishing House *Technologija*), Lithuanian language editor Aurelija Gražina Rukšaitė (Publishing House *Technologija*).

The dissertation defence will be held on 16 April 2025, at 2 p.m. at the Rectorate hall of Kaunas University of Technology in the meeting of the Dissertation Defence Board of the Informatics Engineering field of science.

Address: K. Donelaičio 73-402, LT-44249 Kaunas, Lithuania.
Phone: +370 608 28 527; e-mail doktorantura@ktu.lt.

The dissertation was sent out on 14 March 2025.
The dissertation is available on the internet at http://ktu.edu and at the library of Kaunas University of Technology (Gedimino 50, LT-44239 Kaunas, Lithuania), at the library of Vytautas Magnus University (K. Donelaičio 52, LT-44244 Kaunas, Lithuania) and at the library of Vilnius Gediminas Technical University (Saulėtekio 14, LT-10223 Vilnius, Lithuania).

KAUNO TECHNOLOGIJOS UNIVERSITETAS

NERIJUS ŠATKAUSKAS

ŪKO KOMPIUTERIJOS PASLAUGŲ DINAMINIO VALDYMO METODAS

Daktaro disertacija
Technologijos mokslai, informatikos inžinerija (T 007)

2025, Kaunas

Disertacija rengta 2019-2024 metais Kauno technologijos universiteto Informatikos fakultete, Kompiuterių katedroje.

Doktorantūros teisė Kauno technologijos universitetui suteikta kartu su Vytauto Didžiojo universitetu ir Vilniaus Gedimino technikos universitetu.

Mokslinis vadovas:
prof. dr. Algimantas VENČKAUSKAS (Kauno technologijos universitetas, technologijos mokslai, informatikos inžinerija, T 007).

Redagavo: anglų kalbos redaktorius dr. Armandas Rumšas (leidykla „Technologija“), lietuvių kalbos redaktorė Aurelija Gražina Rukšaitė (leidykla „Technologija“).

Informatikos inžinerijos mokslo krypties disertacijos gynimo taryba:
prof. dr. Rimantas BUTLERIS (Kauno technologijos universitetas, technologijos mokslai, informatikos inžinerija, T007) – **pirmininkas**;
prof. dr. Tomas BLAŽAUSKAS (Kauno technologijos universitetas, technologijos mokslai, informatikos inžinerija, T 007);
prof. dr. Arnas KAČENIAUSKAS (Vilniaus Gedimino technikos universitetas, technologijos mokslai, informatikos inžinerija, T 007);
prof. habil. dr. Artūras KAKLAUSKAS (Vilniaus Gedimino technikos universitetas, technologijos mokslai, informatikos inžinerija, T 007);
doc. dr. Anton RASSOLKIN (Talino technologijos universitetas, Estija, technologijos mokslai, informatikos inžinerija, T 007).

Disertacija bus ginama viešame Informatikos inžinerijos mokslo krypties disertacijos gynimo tarybos posėdyje 2025 m. balandžio 16 d. 14 val. Kauno technologijos universiteto Rektorato salėje.

Adresas: K. Donelaičio g. 73-402, 44249 Kaunas, Lietuva.
Tel. +370 608 28 527; el. paštas doktorantura@ktu.lt.

Disertacija išsiųsta 2025 m. kovo 14 d.
Su disertacija galima susipažinti interneto svetainėje http://ktu.edu, Kauno technologijos universiteto bibliotekoje (Gedimino g. 50, 44239 Kaunas), Vytauto Didžiojo universiteto bibliotekoje (K. Donelaičio g. 52, LT-44244 Kaunas, Lithuania) ir Vilniaus Gedimino Technikos universiteto bibliotekoje (Saulėtekio al. 14, LT-10223 Vilnius, Lithuania).

TABLE OF CONTENTS

LIST OF TABLES

LIST OF PICTURES

GLOSSARY

AHP – Analytical Hierarchy Process

AMQP – Advanced Message Queuing Protocol

BAN – Body Array Networks

BLE – Bluetooth Low Energy

CoAP – Constrained Application Protocol

CPU – Central Processing Unit

DDS – Data Distribution Service

FOA – Fog Orchestrator Agent

HTTP – Hypertext Transfer Protocol

IMOPSO – Integer Multi-Objective Particle Swarm Optimization

IoT – Internet of Things

JADE – Java Agent DEvelopment Framework

MAS – Multi-Agent System

MIPS – Millions of instructions per second

MQTT – MQ Telemetry Transport

NIST – National Institute of Standards and Technology

PAN – Personal Area Networks

QoS – Quality of Service

RAM – Random Access Memory

REST – REpresentational State Transfer

SPF – Single Point of Failure

1. INTRODUCTION

1.1. Relevance of the Work

Internet of Things (IoT) is made of a vast variety of devices capable of sensing and activating. These devices collect, process, and transmit data to other devices, applications, and platforms. It has been claimed [1] that there could have been about 30 billion devices in use by 2020, and this number is expected to grow to 75 billion by 2025. Such unprecedented progress inherently becomes a part of research, technology, and computing. Internet of Things, however, is made of different paradigms, and one of these paradigms which was brought by the Internet of Things is *Fog Computing* [2].

Fog Computing provides its computational resources as a service [3], just like a cloud does it, but, in contrast to a cloud, such a service as computational resources is available as a geographically distributed one. Fog nodes, in addition, are autonomous and decentralized [4]. They are heterogeneous in terms of their processing and storage resources. These resources are often limited due to lower hardware capabilities. And, finally, there should be some co-ordination or orchestration of the fog node interaction among the fog nodes and the cloud with support of mobility in mind.

The purpose of fog orchestration is to get the data transmitted by their end-devices processed by using the most suitable fog nodes. Application requirements may include CPU, memory, energy, bandwidth, etc., and fog node constraints [5]. Since fog nodes are distributed and heterogeneous, and their end devices can be mobile, it is complicated to find a node for the best service placement in relation to performance and optimization.

The dynamics of the Fog Computing system can be double: fog infrastructure dynamics and application dynamics. As Salaht et al. [6] identify, Fog Computing networks are extremally dynamic, where both end-nodes and fog nodes as a part of fog infrastructure appear and disappear, their capabilities and application requirements keep on changing, and network links are prone to failures and instabilities. A dynamic orchestration approach is needed to deploy new applications and eliminate the current ones when they are no longer needed.

After a dynamic service orchestrator has deployed any relevant services within specific fog nodes, there is another hurdle to overcome, specifically, optimization [7]. Fog Computing keeps computing resources close to its users and to the end nodes so that to reduce any delay for IoT services. It can also deal with privacy, data locality, and bandwidth consumption. There are several objectives that can be enhanced by optimization, such as latency, cost, or energy management. It is a part of the *quality of service* (QoS), but it may come with a trade-off.

There is a wide variety of research papers solving the above Fog Computing service and application placement issues. As Costa et al. in their review paper note [8], a centralized orchestration topology is the most popular among their analyzed papers but only three of those analyzed involve a Single Point of Failure issue. The implementation of a decentralized or distributed orchestration topology yields an

increased complexity due to data synchronization needs which depend on reliable communication and higher processing capacities.

There are other researchers, notably, Saad et al. [9], Toczé et al. [10], Guerrero et al. [11] who offer distributed service orchestration topology solutions, yet they do not directly consider any resilience and fog node failures in dynamic networks. Overall, there are just a few orchestration methods for a distributed and dynamic environment [12]. Even fewer of these distributed topology solutions consider the mobility of the end-devices or fog nodes themselves, as it is expressly admitted by Wang et al. [13].

1.2. Object of the Thesis

The object of this thesis is a dynamic orchestration method which is to be used for a Fog Computing service optimization. The thesis subjects are the criteria based on which potential alternatives are prioritized.

1.3. Aim of the Thesis

The aim of this thesis is to develop a Fog Computing service dynamic orchestration method to optimize the use of Fog Computing resources as a service placement in a heterogeneous and dynamic environment considering the application requirements (CPU, memory, energy, bandwidth, etc.) and fog node constraints.

1.4. Tasks of the Thesis

The main tasks of this thesis are the following:

1. To investigate Fog Computing as an IoT paradigm, available Fog Computing orchestration methods and their challenges.
2. To propose a Fog Computing service dynamic orchestration method to optimize the use of Fog Computing resources.
3. To simulate a proposed Fog Computing service dynamic orchestration method by using the relevant software as a proof of the concept.
4. To implement the proposed Fog Computing service dynamic orchestration method by creating a prototype.
5. To take measurements and obtain graphical representations by using both the simulation and the developed prototype as part of experiments to test the orchestration method's performance.

1.5. Scientific Novelty

The following statements on scientific novelty can been presented on the grounds of this thesis:

1. A two-stage dynamic multi-objective optimization method has been proposed which uses *Integer Multi-Objective Particle Swarm Optimization* (IMOPSO)

to find all non-dominated placements of the services, and subsequently the best placement is concluded by using the *Analytical Hierarchy Process* (AHP), which simplifies the criterion comparison process. Meanwhile, other papers previously considered either a two-stage optimization or a multi-objective optimization.

2. A Fog Computing service orchestrator was implemented as a Multi-Agent System (MAS) with increased security, resilience and monitoring capabilities. Multi-Agent Systems are vulnerable to network attacks, and they lack an integrated monitoring tool. Security issues were addressed by integrating a JADE-S add-on, which allows us to sign and encrypt agent communication messages. Monitoring classes were developed to track the resources.
3. The proposed architecture is capable of continuously monitoring any service requests and available resources to make real-time placement decisions as a distributed orchestrator in contrast to central control units which are the most common solutions.

1.6. Practical Value

Fog Computing is a relatively new paradigm, and it is used as a Cloud Computing alternative to provide resources closer to data sources and its users with location awareness and a minimal latency [14]. Fog Computing resources are provided as a service in a dynamic environment taking their application requirements and fog node constraints into account. As the thesis motivation concludes, insufficient attempts have been made so far to develop a distributed dynamic service orchestrator which would consider a SPOF, node mobility and resilience:

- A Fog Computing service dynamic orchestration method based on a two-step optimization process using the *Integer Multi-Objective Particle Swarm Optimization* (IMOPSO) and *Analytical Hierarchy Process* (AHP) has been proposed as a Matlab solution which can be used for a simulation. It provides graphical representations as part of completed optimization calculations.
- A prototype of the Fog Computing service dynamic orchestration has been implemented which can be used for physical assessment. As the prototype interacts with its environment, it is possible to track, encrypt and sign its communication messages, as well as monitor the available fog nodes and their resources.
- Our dynamic orchestration method can be used for a smart building, namely, for a student campus. The method has been designed to optimize the energy consumption of the campus. It may contain several classrooms, laboratories, a gym and a parking lot with a few electric vehicle charging stations. Sensors

and actuators are installed to monitor the environment, e.g., the room temperature, and to adjust it with thermostats for peak or off-peak hours. Vehicle charging can be stopped during peak hours to avoid any power shortages. Users can be tracked as part of the fog node mobility. Meanwhile, security is enhanced to avoid any intrusions.

This PhD research is related to the project *SPARTA* which serves as part of the EU program *H2020-SU-ICT-2018-2020*.

1.7. Thesis Statements

The following statements shall be defended in the framework of this thesis:

1. Distributed dynamic service orchestration improves resilience and optimizes resources in Fog Computing.
2. A two-stage optimization process serves as a solution to a multi-objective optimization problem to optimize all the objectives at the same time when these objectives are contradictory. The *Integer Multi-Objective Particle Swarm Optimization* (IMOPSO) algorithm finds a set of solutions, and the *Analytical Hierarchy Process* (AHP) algorithm chooses the best solution from a Pareto optimal set.
3. A distributed MAS orchestrator can solve Fog Computing issues related to interoperability, performance, service assurance, lifecycle management, scalability, security, and resilience.

1.8. Scientific Approval

All the results which are provided in this thesis are original. They have been presented as publications in 2 scientific journals with the *Web of Science* index and as 2 other publications in informatics, electronics and software engineering peer-reviewed proceedings.

Experimental results have been presented and discussed at the following 2 international conferences:

1. Orchestration Security Challenges in the Fog Computing, *26th International Conference on Information and Software Technologies* (ICIST 2020), Kaunas, Lithuania.
2. Fog Computing Service Placement Orchestrator, *11th International Conference on Electrical and Electronics Engineering* (ICEEE 2024), Marmaris, Turkey.

1.9. Thesis Organization

This thesis is made of 5 chapters:

Chapter 1 is an introduction which presents the scientific novelty, aims, tasks and other aspects of this thesis. It also briefly identifies Fog Computing, its main issues and the thesis motivation.

Chapter 2 gives a thorough review of Fog Computing as a paradigm. It also describes other IoT paradigms, development issues, methods, concepts, frameworks and technologies. Finally, it specifies the current Fog Computing orchestration challenges.

Chapter 3 introduces a Fog Computing service dynamic orchestration method. It explains both the *Integer Multi-Objective Particle Swarm Optimization* (IMOPSO) algorithm and the *Analytical Hierarchy Process* (AHP) as it is a two-step optimization process method. The simulations are completed in Matlab.

Chapter 4 presents a prototype which is implemented as a *Multi-Agent System* (MAS) in JADE. *Raspberry Pi 4* units are used as fog nodes. Dynamic orchestrators based on IMOPSO and AHP are installed on each fog node as part of a distributed topology.

Chapter 5 provides general conclusions on the conducted research.

2. FOG COMPUTING SERVICE DYNAMIC ORCHESTRATION REVIEW

This chapter presents the concept of Fog Computing and its issues. It identifies various Fog Computing paradigms in general, Fog Computing orchestration, and secure orchestration challenges.

2.1. Fog Computing Paradigm

IoT infrastructure is shaping the world in the way that the exchange of data between physical world items and the virtual world has become more and more common. It is suggested that the number of IoT devices connected to the network might currently be around 50 billion according to source [15], as per Srirama et al. The amount of their generated data could reach 79.4 zettabytes by 2025. It can be considered as a global network of an infrastructure with numerous multiple devices, which are made up of different parts including sensing, communication, networking, and information processing [16]. However, long data transmission distances between the end-users and cloud servers lead to a higher network data flow, data loss, latency, and increased energy usage [17]. This is where Fog Computing emerges with an intention to bring computation and storage capabilities to the edge of the network. A distributed Fog Computing infrastructure is effective, with delay-sensitive IoT applications rendering minimal latency and energy consumption to process data while using resource-limited fog/edge devices.

Fog Computing is a relatively new paradigm, and the "[...] definition of Fog Computing and fog nodes is a topic of ongoing discussion," as noted in [18]. The OpenFog Consortium plays, however, a pioneering role in working on these definitions and standards. Even though the *Industrial Internet Consortium*® (IIC™) and the **OpenFog Consortium**® (OpenFog) have joined their forces as "[...] two largest and most influential international consortia in Industrial IoT, fog and edge computing [...]" [19], the primary object the OpenFog Consortium® was formed for was the "[...] principle that an open Fog Computing architecture is necessary in today's increasingly connected world [...] [20]". Their purpose is to apply their high expertise through the open membership which is run independently in the industry, end users and universities. Proprietary and single vendor solutions, as they believe, can limit the diversity, which, in turn, exerts a negative impact on the cost of systems, quality and innovation. They intentionally strive to ensure that the OpenFog architecture would be a completely operable and secure system.

The OpenFog Consortium® was established in 2015 by such cofounders as *ARM*, *Cisco*, *Dell*, *Intel*, *Microsoft*, and Princeton University. The intention of the *OpenFog Reference Architecture* (OpenFog RA) is to assist the relevant companies, software and hardware developers in creating and maintaining any Fog Computing related elements. Fog Computing, as they have defined it, is:

A horizontal, system-level architecture that distributes computing, storage, control and networking functions closer to the users along a cloud-to-thing continuum.

Fog Computing according to their above-referenced document is a supplement to a traditional computing model which is based on a cloud. Any implemented architecture can be located in different network topology layers. Fog Computing can be assisted with a cloud, and all the benefits of a cloud, such as containers, virtualization, orchestration etc., should be available. Pillars are well-discussed in the reference architecture document, which are made of "[…] security, scalability, openness, autonomy, RAS (reliability, availability and serviceability), agility, hierarchy, and programmability."

The above-referenced document implies that Fog Computing is a part of the Internet of Things (IoT), and it has some specific roles in it. Bonomi et al. [21] seek to prove that the Fog Computing characteristics make it an appropriate platform for different IoT services and applications. It also extends the Cloud Computing to the edge of the network. This ability to extend it to the edge of the network may sometimes lead to a misunderstanding as it is noted by the OpenFog Consortium [20] that the Fog Computing is the same as the Edge Computing. However, there are some differences between the two concepts. Fog may use a cloud, but Edge Computing is known for the exclusion of any cloud. Fog relies on a hierarchy, meanwhile, Edge Computing can be defined by a limited number of layers. Fog Computing is also known for improving networking, storage, control and acceleration status.

Fog computing typically features a layered architecture with three different layers: the end-device (terminal) layer, the fog layer, and the cloud layer [22]. An end-device layer consists of end-devices that are distributed around the accessible area. They are designed to collect data in order to transmit the data to the upper layers. A fog layer, meanwhile, is situated at the edge of a network between a fog layer and an end-device layer. It is designed for computing, filtering, joining, transforming [23], and the storage of data. It can be either mobile or static. A cloud layer consists of storage devices and servers denoted by high performance. However, the number of layers or their names in the architecture may slightly differ, as elaborated in [24], where an orchestrator has its own layer between the fog layer and the IoT application layer to make four layers in total. The paper [25] by Aqib et al. suggests using two fog layers instead, where the first one would consist of small to medium calculation capacity nodes, and the second layer would be made of powerful ones. As for the fog radio access networks (F-RANs) to support 5G, there is a network access layer between the cloud layer and the logical fog layer [26]. Five layers are identified in the reference architecture [27]: (1) sensors, edge devices, and gateways, (2) network, (3) cloud services and resources, (4) software-defined resource management, and (5) IoT applications and solutions. Figure 1 provides a visualization involving more details.

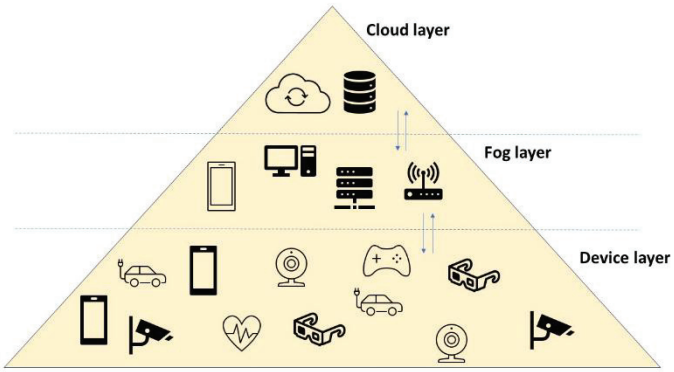


Figure 1. Multi-Layered Fog Computing Structure

Even though the number of Fog Computing layers may sometimes slightly differ, the end-device layer will always consist of IoT devices, such as smartphones, tablets, computers, sensors [28], remote machines, and smart vehicles [29] which communicate in a wired or wireless manner [30] with the fog layer. These end-devices may have limited resources [31], such as storage and computational power, as well as only use a limited bandwidth.

Multi-agent systems (MASs), in turn, as it is claimed [32], have already gained extensive attention from scientists of different areas, such as computer science and civil engineering. They can solve complex tasks by breaking them down into smaller ones. Each task can be assigned to agents that function independently from each other. Agents can make decisions to take actions based on their action history, interactions, and the ultimate goal. Multi-agent systems have recently been a popular research topic, and they are applied in such areas as unmanned aerial vehicles, industrial internet of things, and wireless sensor networks [33]. However, as added in the same publication, irrespective of all the benefits, multi-agent systems are highly vulnerable to network attacks due to an open communication environment and the complexity of the system. There is no integrated system to monitor and manage the activities of all the network nodes. Information exchange is usually very high, but the information flow cannot be verified, and therefore the system is at a security risk.

2.2. Fog Computing Orchestration Dynamics

The availability of Fog Computing resources and services is dynamically changing in the run-time [34]. They are usually mobile, distributed, and they appear and disappear instantaneously. A comprehensive monitoring approach is needed for the right *quality of service* (QoS). Monitoring is the essential function for a proper fog service orchestration [35]. It collects the status information about its fog nodes and communication channels. Once the orchestrator has received an updated view of the infrastructure and services, the orchestrator can make correct decisions, place the services in the node with the relevant resources, and optimize the service placement. There is a trade-off, however, between the information update frequency and the

overhead of nodes. Service availability can be detected by monitoring certain metrics like the service response time. Meanwhile, the causes of a service failure can be found after analyzing the logs.

Whether it is built in a static or dynamic configuration, a monitoring system is expected to collect certain metric values which are relevant to the control of the processes, store the data locally, and transmit the data in a timely manner, whenever needed. Such metrics may include the evaluation of the security level, CPU usage, RAM usage, power usage, range [36], etc. Monitoring can be done continuously when an orchestrator gets a constant data flow of readings, periodically, when the readings are sent as a matter of a cycle, and based on events when a triggering event occurs, such as threshold readings, or a request has been received [5].

For an orchestrator to be dynamically updated with the current infrastructure status, horizontal communication among the nodes is needed as well. The purpose of a communication layer is for an orchestrator to send requests to other nodes (or *local agents* (LA)), as Miele et al. suggest [37]. Incoming connections are observed at a predefined IP address and port. The requested node executes the task and returns its results. A response message is used by an orchestrator to update its architecture configuration table and its service status table. Such communication messages will contain a requested executable command with the relevant arguments.

The most fundamental communication paradigm counterparts a request-response model [38]. REST, HTTP and CoAP are the most common protocols which are based on the request/response method. AMQP, MQTT and CoAP are known for having a very low QoS support for message delivery. However, DDS, oppositely, has a comprehensive set of QoS rules. Another interaction option is the publish-subscribe model. This model was primarily developed because of the need to ensure a distributed, asynchronous, loosely coupled communication between a sender and a recipient [39]. This model consists of 3 parties: the publisher, the subscriber, and the broker. Instead of sending a request message, the subscriber here subscribes to certain events in the system. The broker is responsible for filtering any incoming messages and forwarding them either to the publisher or the subscriber. The publisher is the information provider. There is a considerable number of protocols based on the publish/subscribe interaction model, but the most common models are MQTT, AMQP, and DDS.

2.3. Distributed Service Orchestration

As it has already been identified, it is possible to use a (a) centralized, (b) decentralized, and (c) distributed orchestration topologies. A centralized orchestration is more vulnerable to an SPOF (*Single Point of Failure*). A decentralized orchestration needs a special algorithm to get an FOA (*Fog Orchestrator Agent*) selected. Meanwhile, a distributed orchestration is less common in scholarly papers due to an additional complexity for its consensus management. As Mohd Pakhrudin et al. [40] concluded, a distributed fog network computing system is essential for lower power consumption and the end-user latency. It also reduces the burden to the cloud and enables a localization service for real-time data processing.

When comparing two different approaches [41], such as a distributed and centralized one for the sake of the optimal resource distribution among the nodes to minimize the overall task processing delay and the energy consumption, it was concluded that a distributed architecture performs better for offloading when a delay is prioritized. In the case of a centralized approach, decisions are made by a central unit which is expected to know the status of each of its nodes, however, a fog node makes an individual decision for any service offloading in a distributed topology.

One of the requirements to Fog Computing architectures is the infrastructure resilience. Firouzi et al. [42] use performance metrics where the infrastructure resilience gets the highest score. The reason for this is potential malfunctioning, due to which, the orchestrator must reroute its flow to the neighboring fog nodes in order to resume the data processing. As the authors of the paper claim further, even though the centralized orchestrator can optimize a service provisioning problem jointly with the cloud and the fog layer, intermediate fog orchestrators can contribute to adequate resilience. This resilience, however, can be additionally enhanced by allowing devices on the same layer to communicate directly with each other. Service provisioning can be optimized either cooperatively or individually that way. If that is a non-cooperative case, a cache-like strategy is applicable. If that is a cooperative case, fog nodes communicate with each other through a secure channel for the nearby resource discovery.

Resilience is important due to the need to address a fault tolerance [43]. When a system failure occurs, the architecture should still ensure a seamless connectivity and a minimal downtime. Such a fault-tolerance in a distributed Fog Computing system is achieved by using a master-worker framework as an option. A master fog node is located among general fog nodes. Its purpose is to fulfill such essential tasks as request scheduling, communication with the cloud, and resource management. Certain selection criteria must be used to dynamically consider available resources in order to select the best worker node to make it a master one.

Managing the distribution of a computational load among the fog nodes for a service placement as well as the orchestration of these services is the first complex task which arises, as outlined by Kirsanova et al. [44]. It suggests that an orchestrator has to be responsible for service migration, resource allocation, and service placement to get a global status view. It helps to guarantee a smooth functioning of a complex and diverse system.

Distributed fog infrastructure monitoring to detect the status of hardware resources and network QoS increases fault-resilience in a self-organizing architecture [45]. It is claimed [18] by Yousefpour et al. that Fog Computing orchestration platforms do not get sufficient attempts to implement a monitoring phase. Any analyses require historical and real-time monitoring data for an optimal application service deployment. Monitoring output is used when the services are deployed initially and when they are migrated if the requirements are not met by the current deployment. Monitoring can be challenging in a distributed system because it must deal with limited resources, unstable connectivity, platform heterogeneity, and node failures.

2.4. Service Placement Problem

As suggested by Apat et al. [46], an application placement problem is identified as mapping a certain number of applications to a certain amount of available recourses. Different tasks have to be optimally distributed over fog and cloud resources to achieve specific objectives. Usually, IoT applications have a range of objectives with different constraints. These constraints may include CPU, RAM, the selection of a specific node for a specific task or meeting a placement deadline [47]. Fog service placement compares to a resource allocation [48] which is one of the most relevant challenges. And, while solving these challenges, performance metrics are used to evaluate the effectiveness, such as energy consumption, service delay, resource usage, and service acceptance ratio, service time, cost, QoS, delay, network usage, response time, bandwidth, response deadline, scalability and availability [49]. The optimal usage of fog resources has a significant impact on the *Quality of Service* (QoS) [50], but a distributed and autonomous mechanism is required for this implementation.

Service placement as an orchestration method can be single-objective and multi-objective. Multi-objective optimization is more frequently used in papers, as per Abofathi et al. [49]. The main purpose of an optimization is to seek for a minimum or a maximum of the objective function while scheduling tasks [51]. These objectives themselves can have a critical impact, and they can lead to a delay and changes in energy consumption [51], minimizing costs and meeting deadlines [52], low latency and enhanced Quality of Service [17], makespan [53], hop count, execution time, or network usage [54]. A multi-objective optimization or a multi-objective problem can provide a trade-off solution to satisfy conflicting problems in contrast to a single-objective one.

A service placement problem can be solved by using centralized, decentralized, distributed, or hierarchical approaches [54]. The centralized method has some disadvantages such as (a) a higher network overhead, (b) a longer computation time, (c) a single point of failure (SPOF), (d) heterogeneity management, (e) latency due to the communication between brokers and fog devices. However, a centralized algorithm has a higher chance of finding the optimal solution. Distributed methods (i.e., decentralized ones) have multiple coordinating nodes. They are more flexible, scalable, and adaptable to dynamics. Yet, they are more complex, and not always the optimal solution can be found [6].

A batch [55], static or dynamic [56] method can be used to solve a service placement. A certain number of services are placed based on an estimated workload in a static method. Yet, it leads to a risk of poor service placement due to an excessively low or high resource usage. A dynamic method, in its turn, adjusts the number of services based on demand. However, the resources have to be constantly monitored [5]. Since such algorithms consider the current node state, resources are used more effectively than in the case of static placement. Yet, in batch placement, requests are batched before they are sent for execution.

A service placement solving technique can also be offline and online [50]. It is considered in an offline technique that all the requirements and constraints are known in advance. To be more precise, it can be stated that a placement decision is made in

the compile-time [6]. Meanwhile, online placement decisions are made in the run-time. It is, however, more beneficial to consider a placement as an online technique. It is more dynamic and capable to react to the current changes in the infrastructure.

Dynamic networks are the ones where both the mobile fog nodes and the end users change their characteristics within the time, including network topology changes [57]. A Fog Computing infrastructure has to be mobility-aware, but it is challenging to achieve due to its dynamics. Dynamicity can be related to the fog infrastructure dynamicity and to the application dynamicity [6]. Fog computing architecture is highly dynamic [58], and its nodes may join or leave the network at any time. However, the resources in that particular node may not be sufficient to host a requested service, which leads to a lower QoS, a longer response time, or even service failure. Applications, therefore, should be deployed/removed dynamically while considering the capabilities of the changing fog infrastructure [6].

As the Fog Computing architecture denotes, there are IoT/end-devices at the edge of the network which are highly mobile. Mobility of the end-devices may lead to a low processing time. End-devices may move too far from their relevant fog gateways, resulting in connection interruptions [59]. Poor mobility management leads to data package losses, higher latency, and a lower QoS level [60]. Therefore, any proposed solution should be able to deal with changes in the fog node locations and offloading of the services to maintain the QoS requirements.

2.5. Secure Orchestration Challenges

Fog nodes are computational nodes which talk with different sensors and actuators and provide required services for local data filtering, processing and control; they also act as intermediaries for communication with remote cloud-based services. Frequently, these Fog services are deployed dynamically into the corresponding Fog devices, as required. For example, if an Edge device requires ‘serviceX’, which is currently being provided by Fog node ‘FogA’, and moves to the close proximity of Fog node ‘FogB’, the whole Fog infrastructure must adjust itself in such a way that ‘FogB’ node starts to provide ‘serviceX’ to the Edge node. This kind of Fog infrastructure behavior is assured by using Fog services orchestration techniques discussed in the previous chapter. Usually, Fog service orchestration is implemented by customizing and adapting the already existing Cloud services orchestration solutions [61]. One of the very important issues in such solutions is the QoS assurance and security of the whole system. Figure 2 below summarizes different challenges which must be addressed when designing a security orchestration solution for Fog nodes [62].

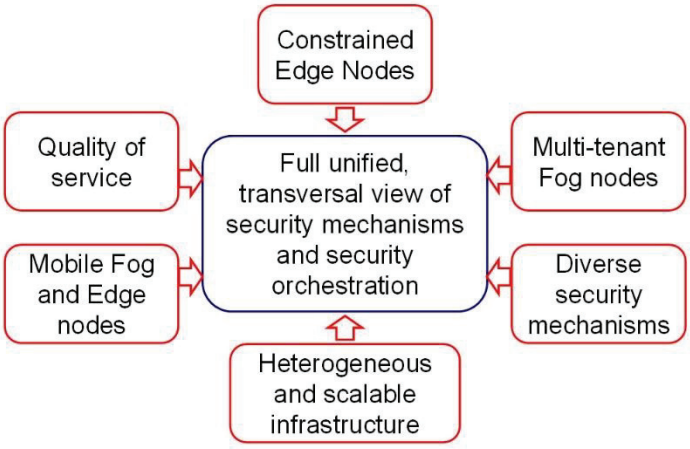


Figure 2. Secure orchestration challenges

Mobile Edge nodes may change the location and approach for a different Fog node which features different architectural and computational or communicational capabilities. Some Edge nodes may be constrained devices with strict constraints on the available security protocols and/or communication methods. The Fog nodes may also be mobile, and approach different Edge nodes with sibling Fog nodes providing services for them. In all cases, the service provision should be optimally divided by all Fog nodes in the nearest proximity of the Edge nodes. On the other hand, the requirements of QoS, i.e., latency, communication bandwidth, data security, etc., must be preserved.

This continuously changing situation requires effective orchestration of the services in Fog nodes, as well as security and QoS assurance. Essentially, after each significant change in the local situation in close proximity of each Fog node, the optimization problem must be solved in order to provide the best possible service for the Edge devices. The constraint categories for such an optimization problem are the following:

1. Security level requirements. Different Edge devices may collect data of different importance, and adequate data protection must be provided while transferring, storing, and processing data.
2. Edge node physical constraints. If an Edge node is constrained, it may be unable to use some of the security and communication protocols due to the limitations in the available processing and/or transmitting power.
3. Communication bandwidth requirements. There is an essential difference in the bandwidth requirements between Edge nodes which provide continuous data streaming and one-time sensors/actuators (e.g., real-time video streaming vs. a temperature sensor which sends several bytes of data every minute).
4. Communication protocol requirements. Some Edge devices may need one-way connectionless broadcasting, whereas others may require strict two-way

communication with acknowledgements. Some solutions may work by using the client/server architecture, yet others may communicate peer-to-peer by using mesh networks.

5. Environment requirements [63]. Some Edge nodes may have short range high bandwidth (i.e., WiFi, ZigBee, BLE, etc.) communication hardware requiring closer proximity to the Fog node, while others may have long range low bandwidth communication hardware (i.e., LoRa).

2.6. Fog Orchestration Solutions

Fog orchestrator components, as concluded by de Brito et al. [64], can generally be divided into three main groups: a fog orchestrator, a fog node which can function as a *fog orchestrator agent* (FOA), and end devices. A fog orchestrator needs to consult its catalogs and certain monitoring data to make an orchestration plan. A fog orchestrator can start its orchestration manually or after reaching a benchmark, such as the availability of other nodes. FOA, in turn, can handle only local resources which are within that particular node.

The main research challenges in the fog orchestration, as identified by Velasquez et al. [65] are the following: resource management, performance, and security management. Resource and service allocation optimization techniques are used, among others, with the objective to address these challenges. The problem is that an allocation procedure is a non-trivial problem, because essentially it is a multi-objective optimization problem.

In order to address the issues in the perspective of Fog Computing, Srinivasa Desikan et al. [4] suggest using four of their proposed algorithms for their identified construction phase and maintenance phase. The construction phase aims to find some probable candidate locations to place the gateways while using a *candidate location identification* (CLI) algorithm. A *Hungarian method-based topology construction* (HTC) algorithm is used to select the optimal gateway locations. Meanwhile, the maintenance phase increases the processing resources in the gateways by intelligent sleep scheduling with the help of the *vacation-based resource allocation* (VRA) algorithm. Their processing and storage resources in the gateways are further improved based on the tracked data arrival rates with the help of the *dynamic resource allocation* (DRA) algorithm. Another option which can be beneficial to improving the performance of a network in terms of reliability and response is caching, as noted in [66]. Caching at the fog nodes can reduce the computational complexity and network load. Even though the computing power is the most critical aspect in the fog node to complete specific tasks, as Zhang [67] suggests, an effective allocation of resources can vary due to limitations. These limitations may include the hierarchy of the fog network, network communication resources, and storage resources.

Yang et al. [68] and Wen et al. [69] state that orchestration has to deal with a number of factors, such as resource filtering and assignment, component selection and placement, dynamics with the runtime QoS, systematic data-driven optimization, or machine learning for orchestration. They implemented a novel parallel genetic

algorithm-based framework (GA-Par) on *Spark*. They normalized the utility of security and network QoS into an objective fitness function within GA-Par. It reduces any security risks and performance deterioration. As their experiments later demonstrated, GA-Par outperforms a *standalone genetic algorithm* (SGA). Skarlat et al. [70] proposed to solve the *fog service placement problem* (FSPP) by using orchestration control nodes which place each service either in the fog cells or in the fog orchestration control nodes. The goal of optimization is to maximize the number of service placements in the fog nodes (rather than in the cloud ones), while satisfying the requirements of each application. The authors used a genetic algorithm to find the optimal FSPP.

Naha et al. [71] identified resource allocation and provisioning as a challenging task considering the dynamic changes of user requirements and limited resources. They proposed their resource allocation and provisioning algorithms based on the resource ranking. They evaluated their algorithms in a simulation environment after extending their *CloudSim* toolkit. There are mainly two steps which are used to solve a deadline-based user dynamic behavior problem. First, they ranked resources based on the processing power, bandwidth, and response time. Later, they provided resources by prioritizing the processing application requests.

As the deployment infrastructure has to adapt itself to extremely dynamic requirements, the fog layer may not provide sufficient resources and, meanwhile, the cloud layer can fail due to latency requirements [72], as per Khebbeb et al. Authors of this paper present a rewriting-based approach to design and verify a self-adaptation and orchestration process in order to achieve a low latency and the right quantity of resources. An executable solution is provided based on Maude, the formal specification language. Properties are expressed by using the *linear temporal logic* (LTL). Their proposed cloud–fog orchestrator works as a self-adaptation controller. It is deployed in the fog layer as a fog node master for low latency requirements. The orchestrator triggers the right actions after a decision has been made.

Smart service placement and management of services in big fog platforms can be challenging due to a dynamic nature of the workload applications and user requirements for low energy consumption and a good response time. Container orchestration platforms are to help with this issue [73]. These solutions either use heuristics for their timely decisions or AI methods, such as reinforcement learning and evolutionary approaches for dynamic scenarios. Heuristics cannot quickly adapt to extremely dynamic environments, while the second option can negatively impact the response time. The authors also noted that they need scheduling policies which would be efficient in volatile environments. They offer a gradient-based optimization strategy using back-propagation of gradients with respect to the input (GOBI). They also developed a *coupled simulation and container orchestration* framework (COSCO) that enabled the creation of a hybrid simulation decision approach (GOBI*) which they used to optimize their *quality of service* (QoS) parameters.

As service offloading is relevant enough in the perspective of time and energy, the selection of the best fog node can be a serious challenge [74], as per Rahbari and Nickray. They presented a module placement method by referring to the classification and regression tree algorithm (MPCA). Decision parameters select the best fog node,

including authentication, confidentiality, integrity, availability, capacity, speed, and cost. They later analyzed and applied the probability of network resource utilization in the module offloading so that to optimize the MPCA.

Linear programming is another extremely popular optimization method used for resource allocation and service placement in fog nodes. Arkian et al. [75] linearized a *mixed integer non-linear program* (MINLP) into the *mixed-integer linear program* (MILP) for the optimal task distribution and virtual machine placement by using the minimization of cost. Velasquez et al. [76] proposed a service orchestrator which tries to minimize the latency of services by using *integer linear programming* (ILP) to minimize the hop count between communicating nodes.

A. Brogi et al. [77] presented a method used to help deployments of composite applications in fog infrastructures, which have to satisfy software, hardware, and QoS requirements. The developed prototype (FogTorch) uses the Monte Carlo method to find the best deployment which ensures the lowest fog resource consumption – the aggregated averaged percentage of the consumed RAM and storage in all the fog nodes.

A sequential decision-making *Markov decision problem* (MDP) enhanced by the technique of Lyapunov optimization was used by Urgaonkar et al. [78] to minimize the operational costs of an IoT system while providing rigorous performance guarantees. The proposed method is intended to be used for a general problem of resource allocation and workload scheduling in cloud computing, but it may also be applied to a service placement problem in fog nodes.

As Fog Computing faces a number of challenges to deal with, optimization is vital, and the classification of optimization problems can play an important role [79]. A service placement problem, in general, has been shown to be NP-complete by Huang et al. [80]. Optimization is typically performed as per Mann et al. [81], who made it up of (a) a set of variables to encode decisions, (b) a set of possible values for each variable, (c) a set of constraints which the variables are to satisfy, and (d) an objective function. Optimization solutions involving end devices and fog nodes differ based on their application area.

Our analyses of the methods used by other authors for service placement problem optimization, as well as the findings of other researchers, show that various well-established optimization methods are used for this task, including integer linear programming, genetic algorithms, the Markov decision process, gradient-based optimization, the Monte Carlo method, reinforcement learning, etc. The objective functions used by the authors of these methods vary from an overall cost minimization [75] [78] to network latency [76], hop and service migration count [76], and the response time and latency of the IoT system [77]. The literature review allows us to conclude that the majority of optimization methods tend to seek for the optimal placement of the services based on the most important parameter of the IoT system, which is represented by the objective function used in an optimization method. Other important parameters of IoT systems in such cases are used as restrictions, and they usually include latency, power, bandwidth and QoS [75] [77], CPU, RAM, and storage demands [70]. This kind of optimization problem formulation allows one to avoid the challenges of multi-objective optimization, but it may be unusable in

situations where more than one objective function is required. Some other approaches tend to evaluate several characteristics by combining them into one composite criterion, such as cost [75] [78] or fog resource consumption [77] composed from an average RAM and storage usage in the fog nodes. The composite criteria calculation equations usually are provided by the authors of the proposed algorithms, and they use some predefined coefficients which are difficult to justify and validate. One very important challenge remains in this area in that case – how to find the best placement of the services according to several different heterogeneous criteria, with different origins and different units of a measurement, when they often contradict each other. The usage of composite criteria is not always the best answer to this challenge.

Dynamic networks are the ones where both the mobile fog nodes and the end-users change their characteristics within the time, including network topology changes [57]. A Fog Computing infrastructure has to be mobility aware, but it is challenging due to its dynamics. Dynamicity can be related to the fog infrastructure dynamicity and to the application dynamicity [6]. The Fog computing architecture is highly dynamic [58], and its nodes may join or leave the network at any time. However, the resources in that particular node may be insufficient to host a requested service, which leads to a lower QoS, a longer response time, or even a service failure. Applications, therefore, should be deployed/removed dynamically, while considering the capabilities of the changing fog infrastructure [6].

See Table 1 for a recent solution review summary.

Table 1. Recent Fog Computing service orchestration solution strategies

Ref.	Solution	Dynamic Orchestrator	Distributed Orchestrator	Resilience
[82]	Application Module placement algorithm	✓	-	-
[83]	ECC platform	✓	-	-
[84]	Management modules	✓	-	-
[85]	PSO-based metaheuristic and a greedy heuristic algorithm	✓	-	-
[86]	Decentralized algorithm	✓	-	-
[87]	Folo: Dynamic task allocation framework	✓	-	-
[12]	Decentralized replica placement	✓	✓	-
[9]	Optimization framework	✓	✓	-
[88]	MM and RM framework	✓	✓ (both)	✓
[89]	ELECTRE load balancing algorithm	✓	-	✓

[90]	MicroFog framework	-	✓	-
[91]	S-HIDRA architecture	-	✓	-
[50]	A3C algorithm	✓	✓	-
[92]	Kubernetes framework	✓	-	-
[93]	Framework	✓	-	-
[94]	Two-stage optimization model	✓	-	✓
[56]	DCSP method	✓	✓	-
[95]	ANFIS and GAO	-	-	-
[96]	CFS model	✓	-	-
[97]	CBR-MADE-k model	✓	✓	-
[98]	Orchestration and management solution	✓	✓	-
[99]	MILP model	-	-	✓

There are numerous research papers trying to solve service orchestration or service placement by using a dynamic approach. Some of them use a distributed control method instead of a centralized or federated one. Some of them focus on resilience. However, virtually none of them is dedicated to a dynamic service placement in a distributed way with an intention to keep it more resilient due to a distributed approach.

2.7. Conclusions

1. This chapter reviews Fog Computing as a paradigm, dynamic and distributed orchestration, its challenges, service placement problems, proposed Orchestration solutions, and the identified Fog Orchestration Security challenges. As Fog Computing is the environment where an Orchestrator acts, Orchestration security challenges depend on the Fog Computing itself and on the available Orchestration solutions.
2. Such keywords as ‘Fog Orchestration’, ‘Fog Orchestration Challenges’, and ‘Fog Orchestrator’ were used to search for papers with relevant information. About 150 papers were checked for more details from the returned search results, and 20 papers were picked for further analysis while rejecting the other ones due to the absence of some relevant information. These papers were reviewed while looking for any specified Orchestration challenges. These challenges were classified in order to detect the most relevant ones among the papers. Security/privacy, dynamics/mobility, and scalability/complexity are the most common challenges to be addressed.
3. Recent service orchestration solutions have been summarized when looking for the problems they solve. Dynamic service orchestrators are rather

common, but few of them are distributed in addition to this. Even fewer of them consider resilience.

4. As the above conducted literature review suggests, our orchestration method should be developed based on such criteria as security, dynamics and scalability. These are the most common challenges. The type of the orchestrator itself is defined by a further solution review where a distributed, dynamic and resilient orchestrator would be the most relevant. However, such a combination of criteria has so far been insufficiently considered in other researches.

3. FOG COMPUTING SERVICE DYNAMIC ORCHESTRATION METHOD SIMULATION

This chapter is dedicated to a simulation of the Fog Computing service dynamic orchestration method based on the IMOPSO and AHP methods in Matlab. Matlab was chosen due to available libraries which were suitable for integration and simulation.

3.1. Orchestrator Components and Architecture

In this thesis, we consider the fog orchestration architecture and components presented in Figure 3. We have a service orchestrator in the cloud layer which is used to optimally distribute the services between several orchestrated fog nodes. The orchestrated fog nodes host some services which communicate with end devices, collect and process data, and make some local decisions on the control of actuators located in the end device layer. Special services (orchestrator agents) are physically located in each fog node, and they communicate with the orchestrator to be provided with all the necessary information needed to make any decisions on service placement; see Figure 3 for more details.

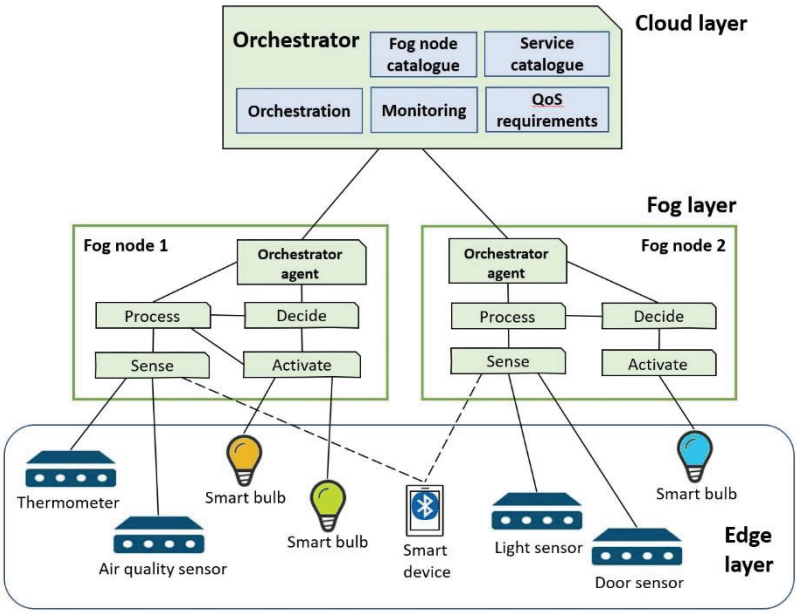


Figure 3. Fog dynamic orchestrator architecture and its components

Orchestrator agents locally monitor the hardware and software environment of the fog nodes. They are aware of the current CPU and RAM usage, power requirement and energy levels, available communication protocols and bandwidth, security capabilities, the state of the hosted services, etc. They summarize all the collected information to provide it to the orchestrator in the cloud layer. The orchestrator is aware of the current situation in all the fog nodes, and, additionally, it has security and

QoS requirements imposed by the application area of the IoT solution, and it makes decisions on starting, stopping, or moving particular services among the orchestrated fog nodes. The decisions made by the orchestrator are communicated down to the orchestrator agents inside the fog nodes; then, the orchestrator agents initialize the required actions on the services. A control cycle performed inside the orchestrator is illustrated in Figure 4.

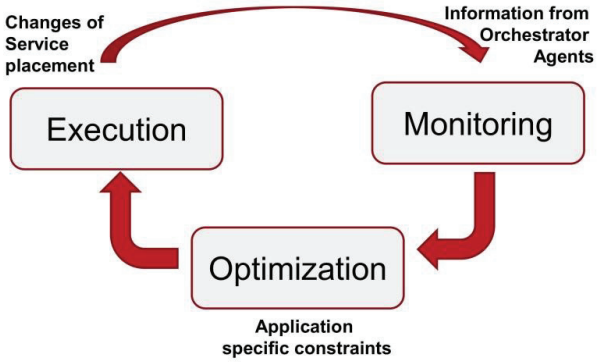


Figure 4. Dynamic orchestration control cycle

The method of the fog service orchestration presented in this thesis is intended to be used inside the orchestrator. The main task of the proposed method is to optimally distribute n services among k fog nodes according to the information collected from the corresponding fog nodes and the requirements imposed by the area of an application of the IoT system. This task of a service distribution is not trivial since several different optimization criteria which contradict each other must usually be considered (i.e., the security level, energy consumption, bandwidth capabilities, latency, etc.). The number of possible different distributions of services among fog nodes increases rapidly with the increase in the number of available fog nodes and services. As any evaluation of all the possible placements of the services is infeasible, therefore, more sophisticated methods are needed. Moreover, the situation and the evaluation criteria can change dynamically due to the dynamic environment of the fog architecture. Some currently available fog nodes as well as end devices may change their location, or new fog nodes may even emerge while, on the other hand, some currently running services may become unused, and some new services may occur.

3.2. Two-Step Multi-Objective Optimization Process

We propose to use a multi-objective optimization process to decide which placement of n available services in k fog nodes is the best according to the given constraints and conditions [36]. An overall diagram of this proposed two-step optimization process for a fog service dynamic orchestration method is presented in Figure 5.

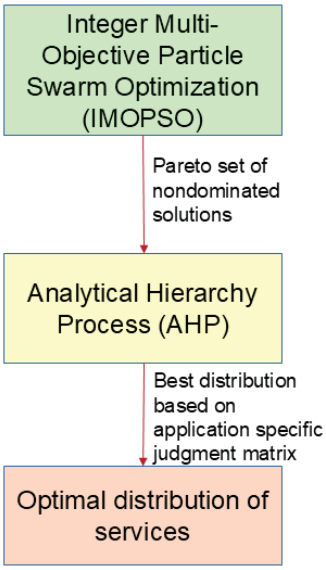


Figure 5. Proposed two-step optimization process diagram

Our optimization process has two main steps – multi-objective optimization and multi-objective decision making – but the problem must be expressed as a formal mathematical model before using any formal optimization methods. The following subsections describe this optimization process in detail. We summarize the key notations used in this thesis in Table 2 in order to give a description of this optimization process.

Table 2. Key notations

Notation	**Description**
n	Total number of services
k	Total number of available Fog nodes for hosting the services
$X_i = (x_1, x_2, \dots, x_n)^T$, $x_j \in \{1,2, \dots, k\}, j = 1,2, \dots, n$	i-th possible distribution of services among the Fog nodes, also the position of the i-th particle in n-dimensional definition area
m	Total number of evaluation criteria, also the number of objective functions
$f_j(x), j = 1,2, \dots, m$	j-th evaluation criteria, objective function
$F_i = \left(f_1(X_i), f_2(X_i), \dots, f_m(X_i)\right)^T$	Score vector of the i-th particle
$V_i, V_i \in R^n$	Velocity of the i-th particle
$pBest_i$	The best score of the i-th particle
$pBPos_i$	The best position of the i-th particle
$gBest$	The best global score of all the particles

$gBPos$	Position of the particle with the best global score
S	Set of particles, swarm
R	External repository of particles, a set of Pareto optimal solutions
X_{opt}	The best service distribution among all the available Fog nodes, particle with the best score

3.3. Service Distribution Problem Objective Functions

The main task of this optimization process is to find an optimal distribution of n services among the possible *k* Fog nodes. Each Fog node may have slightly different characteristics, but we assume, that all the nodes are capable of running all the services. The goal of optimization is to distribute all the services in such a way that a set of important characteristics is optimal. Characteristics of the *i*-th possible service distribution X_i are expressed by the values of the objective functions $f_j(X_i), j=1,2,\ldots,m$ and constraint conditions. The objective of this optimization process is to find the best service distribution X_{opt} which minimizes all the objective functions f_j, i.e., we have a multi-objective optimization problem:

$$X_{opt} = \underset{i}{argmin}\, F(X_i); \qquad (1)$$

where $F(x) = \{f_1(x), f_2(x), \ldots, f_m(x)\}$ is a set of the objective functions, and $x \in \{X_i\}$ is a member of the set with all the possible service distributions.

Constraint conditions are expressed by Equations:

$$\begin{cases} g_j(X_i) \geq 0, \ j = 1,2,\ldots,n_g \\ h_k(X_i) = 0, \quad k = 1,2,\ldots,n_h \end{cases} \qquad (2)$$

Different fog nodes have different performance, network bandwidth, and security characteristics. Different distributions of services among the fog nodes may produce a working system with slightly different characteristics. For example, if one fog node supports a lower level of security (due to limited hardware capabilities), and an important service is placed in this node, then, the overall security of the whole system is reduced to the security of the least secure fog node. We consider multiple objective functions $f_j(.),\ j = 1,2,\ldots,m$ to evaluate all such situations, which include: the overall security of the system, CPU utilization, RAM utilization, power utilization, range, etc. The objective functions which were used in our experiments are described in the following paragraphs.

The security of the whole system while using the *i*-th service distribution $f_{sec}(X_i)$ is defined by the lowest security of all the services. We assign security levels (expressed in security bits, according to NIST [100]) to fog nodes based on their capabilities to support the corresponding security protocols. We assume that services are capable of working on all the fog nodes; then, the value of the security criteria

function $f_{sec}(X_i)$ for the service distribution X_i is the lowest security level of all the fog nodes in which at least one service is hosted. For example, if we have a situation where three fog nodes are able to provide 128 bits of security and one fog node is constrained to support only 86 bits of security, and if at least one service is hosted by it, then, the overall security of the service distribution is equal to 86 bits, i.e., the value of the objective function $f_{sec}(X_i) = 86$. If we use our services in the application area which requires a specific level of security, then such a requirement is expressed as a constraint condition, i.e., if some application area requires at least 128 bits of security, then we have a corresponding constraint $g_{sec}(X_i) \geq 128$.

The criterion of CPU usage $f_{CPU}(.)$ evaluates how evenly, CPU utilization-wise, the services are distributed among the fog nodes. The main idea here is to try to decrease the overall CPU utilization of the system to allow hosting of additional services more easily in case they occur during the runtime of the system. Each fog node has its CPU capabilities expressed in MIPS, which depend on HW capabilities of the corresponding fog node. All services are also evaluated for the required CPU resources. To calculate the value of the CPU usage of the whole system while using the *i*-th service distribution $f_{CPU}(X_i)$, we first calculate a relative CPU usage for each fog node (by dividing the sum of CPU resources required by all the services hosted in each fog node by the capabilities of the corresponding fog nodes), and we find afterwards the maximum CPU utilization among all the fog nodes. The lower the maximum CPU utilization is, the better service distribution we have. We can obtain this situation while using this method of calculation, when some service distributions make up a CPU allocation greater than 100% in some fog nodes and, therefore, corresponding constraints are added to the optimization problem. The usage of this criterion automatically solves some frequent restrictions and incompatibilities, i.e., situations when some services require CPU resources which may not be provided by some fog nodes.

The criterion of RAM usage $f_{RAM}(.)$ which evaluates how evenly RAM utilization is distributed among any fog nodes hosting the services is very similar to CPU usage. The calculation of this criterion is the same as the calculation of CPU usage. A constraint which does not allow exceeding 100% of the RAM utilization in each fog node is also added.

A criterion of the power usage $f_{pw}(X_i)$ of the possible service distribution Xi is evaluated by using the average power requirements of each service (expressed in mW) and the available power of the fog nodes (expressed in mW). The main objective of this evaluation is to maximize the overall runtime of the system. A calculation is performed by dividing the sum of power requirements of all the services hosted in each fog node by the available power of a corresponding fog node to find the maximum among all the fog nodes. A distribution of services is better in such a case when all the fog nodes are evenly loaded power-wise, i.e., when the maximum power utilization is minimized.

The communication of fog devices with sensors and actuators is affected by the physical range between the devices in some cases. Some communication protocols add strict requirements for the range as some of them may be less efficient if the communication range is increased. A criterion of the maximum range $f_{rng}(.)$ may be

used to assess these properties. In this study, a criterion of the range is calculated by averaging the range of each fog node location with respect to all the devices the particular fog node is communicating with in order to find the maximum of these ranges among all the fog nodes hosting at least one service which requires communication with the end devices. The main idea of this criterion is to prefer a shorter communication path as it ensures better performance in most cases. Any corresponding constraints on the range may be also added if a communication protocol induces such restrictions.

Other application-specific criteria such as local storage capabilities, communication latency, bandwidth, etc., may also be evaluated, defining the corresponding objective functions representing the system characteristics which are important in a particular application scenario. All specific implementations of the criteria evaluation functions $f_i(.)$ are implementation specific and are out of the scope of this thesis. A proposed orchestration method is not limited to any specific amount or nature of the objective functions as long as they follow a few common criteria:

• A return value of the objective function must be a positive real number.

• Better values of the criteria must be expressed by smaller numbers (this is because the particle swarm optimization method searches for a minimum of the function).

Generally, one common feature of all these objective functions is that they are conflicting objectives. Saif et al. [51] identifies conflicting objectives such as costs, resource utilization, execution time, or communication time. Finding a solution depending on various conflicting objectives, as per Langhnoja et al. [101], means minimizing the waiting time while at the same time improving the resource utilization or maximizing the profitability. Any optimization of one objective in this case can conflict with another one. For example, we may consider moving all the services to more secure fog nodes to increase security, but such a service distribution will likely cause a reduced power efficiency, an excessive load on some of the nodes, and a lower overall runtime of the system. Moreover, different objectives have different measurement units, e.g., security may be evaluated in bits, while the power requirement of the services is measured in Watts, any available network bandwidth is measured in kbps, etc. Even if all the measurements are converted to real positive numbers, it is still very difficult to objectively compare them. There is no single solution to a multi-objective optimization problem that would optimize all the objectives at the same time. The objective functions are contradictory in this situation; therefore, a set of non-dominated (Pareto optimal) solutions can be found. We propose to use a two-step optimization process in order to deal with this situation, where the first step will use a multi-objective optimization to find a set of solutions (possible distributions of services), the elements of which are a Pareto optimal. We propose to use for this the *Integer Multi-Objective Particle Swarm Optimization* (IMOPSO) method described in the next paragraph. The choice of the particle swarm optimization method is based on the research of other authors (see [102] [103] [104]) which shows that this method is suitable for a similar class of problems, and it demonstrates good results. We will use the *Analytical Hierarchy Process* (AHP) in the second step with

the objective to choose the best solution from a Pareto optimal set for its benefits as per [105] which are further discussed in this chapter.

3.4. Service Distribution Pareto Set Determination Using IMOPSO

The original *Particle Swarm Optimization* (PSO) algorithm is best suited for an optimization of continuous problems, but several modifications still exist; see [106] [107], which enable it to be used for discrete problems. In the case of multiple objectives which contradict each other, the PSO algorithm may be adapted to find a Pareto optimal set of solutions [108] [109]. We used the *Multi-Objective Particle Swarm Optimization* (MOPSO) method proposed by Coello et. al. in [109] to find a Pareto set of the possible service distributions among the fog nodes. In order to use this method, we had to slightly adapt it for it to work in the constrained integer n-dimensional space of possible distributions of services represented as the particles of a swarm (the original method uses a continuous real number space).

We used the vector $X_i = (x_1, x_2, \dots, x_n)^T$, $x_j \in \{1,2,\dots,k\}$, $j = 1,2,\dots,n$ to encode the *i*-th distribution of services, where n is the number of services which have to be distributed among *k* fog nodes. The meaning of the vector element $x_j = l$ is that the *j*-th service must be placed in the *l*-th fog node.

A general diagram of the Integer Multi-Objective Particle Swarm Optimization (IMOPSO) algorithm is shown in Figure 6.

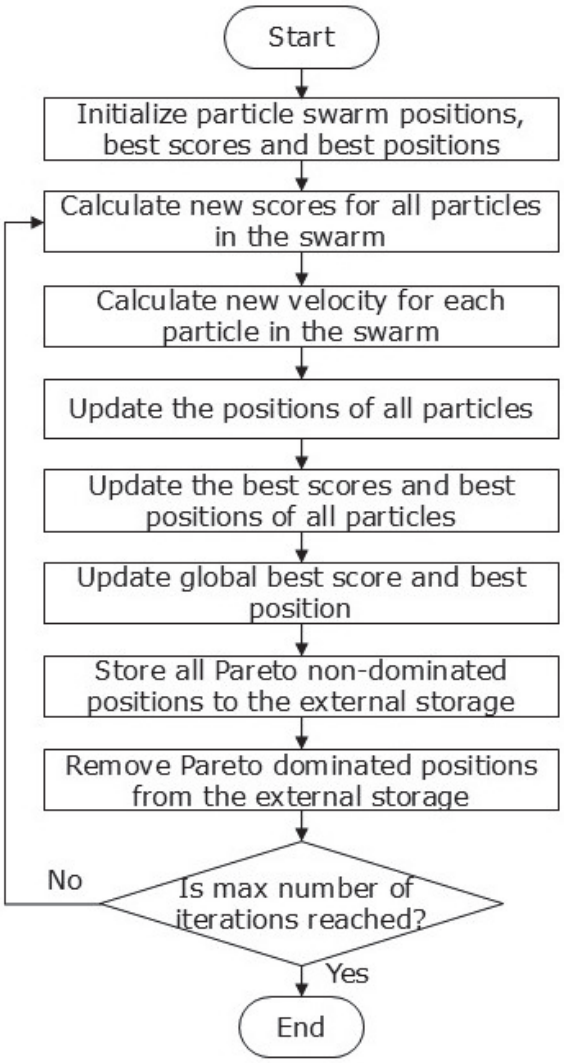


Figure 6. General IMOPSO algorithm diagram

The main steps of the IMOPSO algorithm which was used in a simulation are the following:

1. Initialize the particle swarm S by randomly generating an initial set of positions of the particles (possible service distributions) X_i, $i = 1, 2, \ldots, |S|$, where $|S|$ is the initial size of a particle swarm.
2. Initialize the velocities $V_i = \vec{0}$, the best scores $pBest_i = \overrightarrow{inf}$, and the best positions $pBPos_i = X_i$ of all the particles in the swarm S. Initialize the global best position $gBPos = \vec{0}$ and the global best score $gBPos = \overrightarrow{inf}$.

3. Repeat it until a maximum number of iterations has been reached:
   - Evaluate the new scores F_i of all the particles in the swarm S by using all the objective functions: $F_i = \left(f_1(X_i), f_2(X_i), \dots, f_m(X_i)\right)^T, \quad i = 1, 2, \dots, |S|$.
   - Calculate new velocities of each particle by using the expression $V_i = wV_i + r_1(pBPos_i - X_i) + r_2(gBPos - X_i)$, where w is an inertia weight (initially, a real value around 0.4); r_1 and r_2 are random numbers in the range of [0..1]; V_i is a velocity of the i-th particle; $pBPos_i$ is a position of the i-th particle with the best score; X_i is the current position of the i-th particle; and $gBPos$ is a position of the particle with the best global score.
   - Update positions of all the particles in the swarm: $X_i = round(X_i + V_i)$, $i = 1, 2, \dots, |S|$. The position is approximated to the nearest integer value. If the particle is out of the range, we give it an opposite direction of the speed $V_i = -V_i$, and set the position X_i to the edge of the range of its definition.
   - Update all the best scores $pBest_i$ and the best positions $pBPos_i$ of all the particles in the swarm $i = 1, 2, \dots, |S|$. If the new score F_i dominates the current best score $pBest_i$, then update the best position and the best score of the i-th particle. If the new score neither dominates nor is dominated by the current best score of the i-th particle, then set the best position (and the best score) of the particle to a new position with the probability of 0.5.
   - Update the global best score $gBest$ and the global best position $gBPos$ of the particles using the same algorithm as updating the best scores of the individual particles.
   - Store the positions and scores of the particles that are non-dominated in the external repository (set R). Use all the available particles in the sets S and R during any dominance comparison.
   - Analyze the repository R and remove all the duplicated and dominated scores.
4. The external repository R is a set of Pareto optimal solutions.

3.5. Finding an Optimal Service Distribution Using AHP

We used the *Analytical Hierarchy Process* (AHP) [105] [110] to choose the best solution from a Pareto optimal set by using pairwise comparisons of all non-dominated distributions of services using all the available objective functions. AHP is usually used in situations where a decision must be made when using a small amount of quantitative data, while employing a deep analysis performed by several decision-making parties, by applying a pair-to-pair comparison of possible solutions. AHP may

be adapted to be used by machine-based decision making in scenarios where complex multiple criteria problems are evaluated [111] [112] [113]. A choice of AHP instead of other more formal multi criteria decision-making algorithms is based on the following reasons, as indicated in [105].

AHP allows to automatically check the consistency of the evaluations provided by decision makers. AHP uses normalized values of criteria, and thus it allows to use heterogeneous measurement scales for different criteria. For example, one can use a purely qualitative scale for the security (Hi, Low, Medium) while using inconsistent numeric scales for any power and CPU requirements at the same moment. AHP uses pairwise comparisons of the alternatives only, which enables to ease Multi Objective decision making to get an improved reliability of the results. The importance of the criteria used in the AHP process is also evaluated by using the same methodology, which allows to skip the most controversial step of manual weight assignment to different criteria.

A three-level hierarchical structure of the AHP process is generalized in Figure 7. Level One is an objective of the process which, in our case, is to choose the optimal distribution of all the available services among the Fog nodes. The second level is those criteria which are the same as the objective functions used in the IMOPSO part of the orchestration method. An important step in this level is to use the same AHP process to find a weight of all the criteria by using a pairwise comparison of the criteria. This step should be done manually before placing an automatic service allocation algorithm into production. Moreover, the step of the evaluation of a criteria importance should be different based on the application area in which a service orchestrator is applied. For example, security may be evaluated as more important than power efficiency in a healthcare application compared to the home automation application. We assume in our algorithm that the step of the evaluation of the criteria importance is already performed, and the decision-making system already has its judgment matrix with all the required weights of all the criteria in Level Two. See Figure 7 for our hierarchical AHP framework.

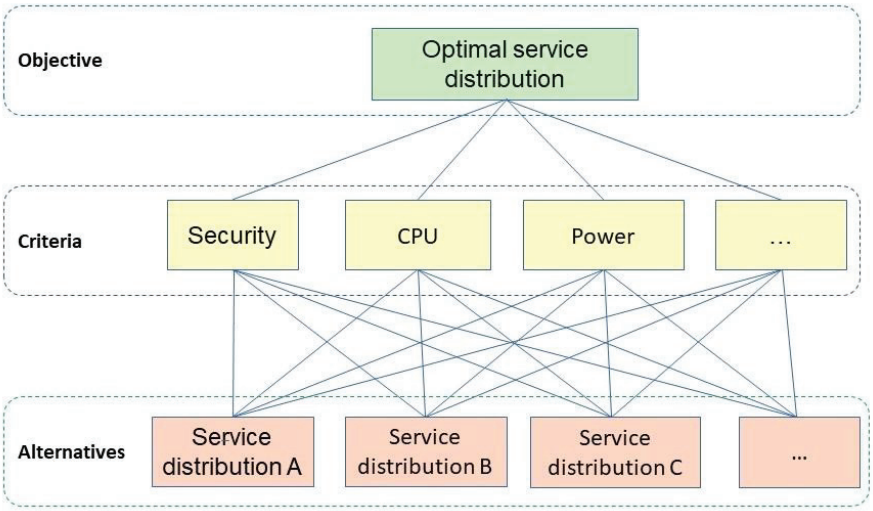


Figure 7. AHP hierarchical framework

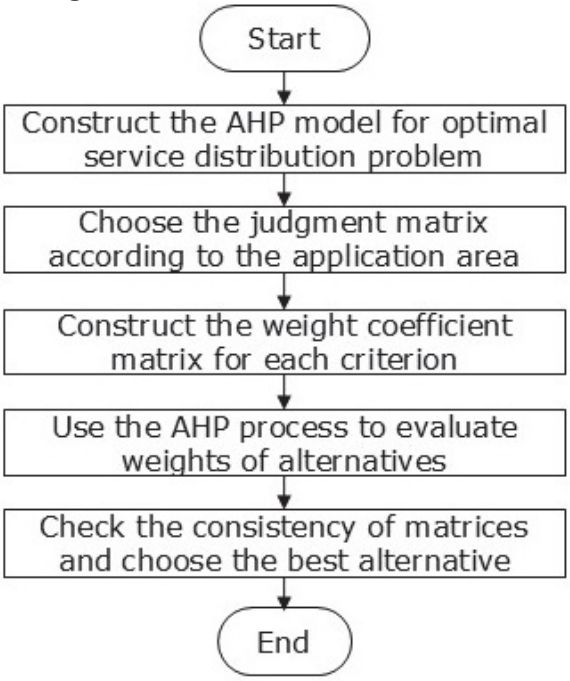


Figure 8. General AHP decision making process

The main steps of the AHP process which was used in our simulation as per Figure 8 are the following:

1. Construct a corresponding AHP framework by using a Pareto optimal solution set to prepare all the data structures for a comparison of the alternatives while using all the available criteria.

2. Load the judgment matrix with the results of the criteria pairwise comparisons prepared to be used in the current application area.
3. Repeat the following step for each criterion (objective function) $f_k(.)$, $k = 1,2,\dots,m$:
   - Construct the weight coefficient matrix $M_k = (m_{i,j})$ using all the alternatives in a Pareto optimal solution set *R*. The size of the matrix M_k is $s \times s$, where $s = |R|$; $m_{i,j} \in (0,9]$; $m_{i,j} = {}^{1}/_{m_{j,i}}$; $m_{i,i} = 1$; $i,j = 1,2,\dots,s$. The matrix M_k elements are calculated by using special comparison functions $m_{i,j} = comp_k(X_i, X_j)$ which use a corresponding objective function $f_k(.)$, calculates two objective function values $f_k(X_i)$ and $f_k(X_j)$, compares them with each other to transform the result to a required real number from the interval (0,9]. The comparison function heavily depends on the meaning of the criteria, and the corresponding real number represents a preference of one alternative over another [104].
4. Provide all the created matrices to the standard AHP decision-making method to get any estimated weights of all the alternatives.
5. Check the consistency of the provided matrices by using consistency indicators provided by the AHP process. Choose the best alternative as the final solution of this orchestration method.

3.6. Simulation Illustrative Scenario

The simulation results of our method are summarized in this section with discussion on each result. The implementation of a real Fog computing environment with the measurement of all the parameters used in the service placement decisions is out of scope of this thesis, and it also makes it difficult to scale the solution and reproduce the results; therefore, we used a simulation. The main objective is to show how the proposed method performs in different situations as well as to test the feasibility of the proposed service placement method.

We implemented our proposed optimal fog service dynamic orchestration method by using *Matlab*. This implementation uses as an input some basic performance information on the Fog nodes and services, a set of the objective functions and an application area specific judgment matrix *J*. The method performs an *Integer Multi Objective Particle Swarm Optimization*, finds a Pareto optimal set of solutions, automatically performs an AHP process using a provided judgment matrix, and finds the best placement of the services in the Fog nodes.

We used an illustrative scenario to evaluate the characteristics of the proposed method. We have 4 Fog nodes and 13 services in this scenario, and they must be optimally placed in those Fog nodes. The capabilities of the Fog nodes and requirements of the services are chosen to show how the proposed method performs in different situations. We used several papers [70] [114] [115] analyzing various requirements of the real hardware and software IoT systems to provide realistic

numbers. A summary of the Fog node parameters and requirements of the services are presented in Table 3 and Table 4.

Table 3. Resources available in fog nodes

	Power, mW	CPU, MIPS	RAM, MB	Security, bits
Fog1	1000	2000	512	256
Fog2	2000	1000	256	112
Fog3	1000	1000	256	128
Fog4	2000	500	512	86

Table 4. Resources required by the services

Service	Processing power, mW	Transfer power, mW	CPU, MIPS	RAM, MB
Sense1	5	20	50	10
Sense2	5	25	60	15
Sense3	5	20	50	20
Process1	100	0	200	60
Process2	150	0	250	75
Process3	130	0	230	70
Process4	120	0	300	50
Process5	120	0	240	80
Process6	140	0	250	55
Process7	200	0	200	70
Actuate1	4	21	50	10
Actuate2	5	20	60	15
Actuate3	4	19	50	10

The security level of any Fog device is determined by the hardware and software capabilities as well as by the availability of the corresponding libraries, and it is expressed in bits according to the NIST guidelines [100].

All services are divided into three main groups. Sense1, Sense2 and Sense3 services are primarily used to communicate with any corresponding sensor devices, collect the measurement data, and provide the data to the other services for processing. On the other hand, services Actuate1, Actuate2 and Actuate3 are mainly used to communicate with the actuator devices. The rest of the services are primarily used to collect data, perform calculations, and make decisions. Resource requirements of the services from different classes are very different.

We used a ‘dynamic’ objective function representing the power requirements of the service in order to better illustrate the capabilities of our method. The power requirements of the service depend on which Fog node is being used to host this service. This is achieved by dividing the power requirements into two parts: the processing power and the transfer power. The processing power is constant, and it is

always required to perform an operation (the values of power requirements were used from publication [70]). On the other hand, the transfer power presented in Table 4 is required if no security is used to transfer the data (i.e., a plain http protocol is being used). The information on the required power levels for a data transfer without any security is based on the experimental results presented by [116] (Venčkauskas et al.). When the service is placed in a Fog node providing more security, then, the corresponding requirement for the transfer power is increased. For example, if a service is placed in a Fog node providing 86 bits of security (e.g., such a node is using 1024-bit RSA for key agreement), then a corresponding transfer power is multiplied by a coefficient of 1.5. The transfer power increase coefficients were based on the results presented in [114] [117]. We decided after an analysis of the data provided to use these multipliers for modeling the increase of power due to an increased security: 1.5 for 86 bits of security, 2.25 for 112 bits, 4 for 128 bits, and 7.5 for 256 bits of security.

3.7. Evaluation Results

We use a simplified scenario where only two objective functions are used to show how the IMOPSO algorithm works and what the Pareto set of solutions looks like. A Pareto set may be displayed in this case by using a two-dimensional chart. The judgment matrix used in this case consists only of 4 elements:

$$J = \begin{pmatrix} 1 & 3 \\ \frac{1}{3} & 1 \end{pmatrix}; \qquad (3)$$

If two objective functions RAM and CPU are being used, then, this judgment matrix means that an even RAM usage distribution among all the Fog nodes is more important than an even CPU usage. A Pareto set produced by the IMOPSO algorithm is presented in Figure 9. Then, a Pareto solution set is used in the second stage employing an AHP process to find the best placement of the services. The best placement is summarized in Table 5.

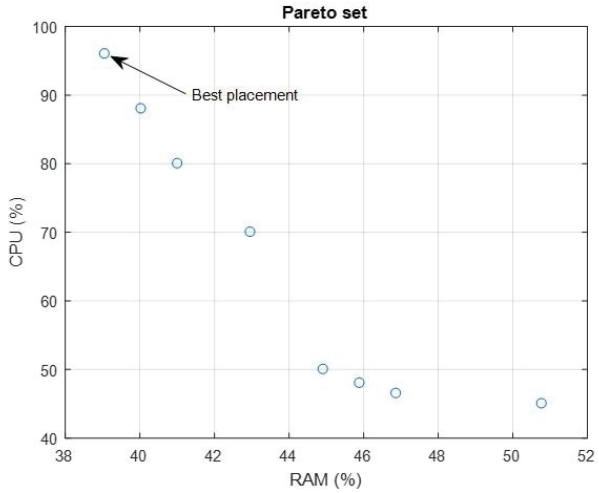


Figure 9. Pareto set of a simplified scenario

Table 5. Best service placement in a simplified scenario

	Fog1	**Fog2**	**Fog3**	**Fog4**
Services	Actuate3 Process4 Process5 Process6	Sense1 Sense2 Actuate2 Process1	Sense3 Process2	Actuate1 Process3 Process7

The best score (the best values of the objective functions) is, in this case, $(39,\ 96)^T$, meaning that this service placement ensures a maximal RAM usage of 39% among all four Fog nodes. The maximal usage of CPU is 96% in this case.

The second scenario shows the influence of the judgment matrix to the optimal placement of services. Four objective functions in this case are used: Power, CPU, Security and RAM. The first judgment matrix is prioritizing Security and Energy over CPU, and RAM:

$$J_1 = \begin{pmatrix} 1 & 3 & \frac{1}{6} & 3 \\ \frac{1}{3} & 1 & \frac{1}{6} & 1 \\ 6 & 6 & 1 & 6 \\ \frac{1}{3} & 1 & \frac{1}{6} & 1 \end{pmatrix}; \qquad (4)$$

The second judgment matrix is prioritizing even power consumption:

$$J_2 = \begin{pmatrix} 1 & 7 & 3 & 6 \\ \frac{1}{7} & 1 & \frac{1}{2} & 1 \\ \frac{1}{3} & 2 & 1 & 2 \\ \frac{1}{6} & 1 & \frac{1}{2} & 1 \end{pmatrix}; \qquad (5)$$

A Pareto set of solutions using the judgment matrix J_1 is shown in Figure 10. Only some projections of the set are shown as the set members are four-dimensional vectors, and they cannot be fully rendered in charts.

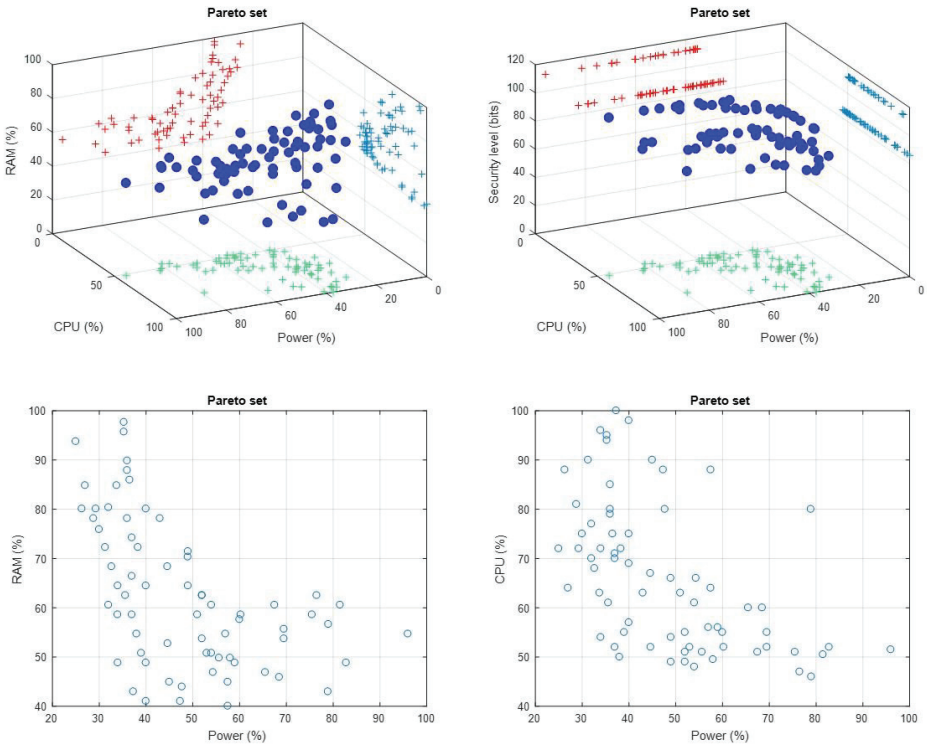


Figure 10. Pareto set of the second scenario; security and energy are prioritized

The best placement of services is presented in Table 6. The best score in this case is $(35,\ 94,\ 112, 98)^T$. The overall security (determined by the security level of the least security capable Fog node hosting at least one service) is 112 bits in this case, and Fog node Fog4 is not hosting any services, as its security is only 86 bits.

Table 6. The best service placement in the second scenario; security is prioritized over other criteria

	Fog1	**Fog2**	**Fog3**	**Fog4**
Services	Actuate3 Process4	Sense1 Sense2	Sense3 Process2	Actuate1 Process3

	Process5 Process6	Actuate2 Process1		Process7

If the judgment matrix J_2 which prioritizes an even power consumption is used in the same situation, then, the best placement is different (see Table 7), and the best score is $(26, 88, 86, 84)^T$.

Table 7. The best service placement in the second scenario; security is prioritized over other criteria

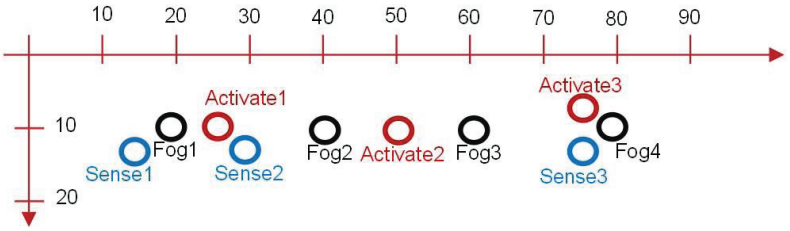
	Fog1	**Fog2**	**Fog3**	**Fog4**
Services	Process5 Process6	Process2, Process3, Process4, Activate1, Activate3	Process1	Sense1, Sense2, Sense3, Process7, Activate2

The maximal power consumption among all the Fog nodes is 26% in this case, and it is significantly better than in the first variant (35%), but the overall security of the solution has been degraded to 86 bits, as several services were placed into the Fog node Fog4.

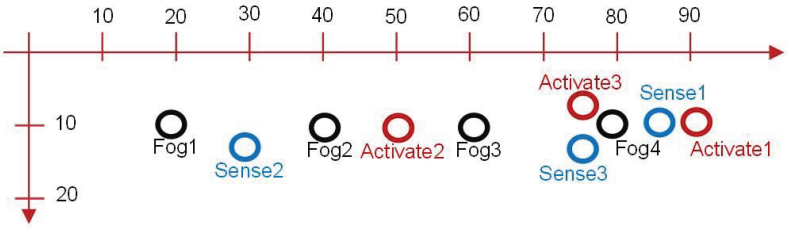
The third illustrative scenario is meant to illustrate how the proposed service placement method works in cases when some devices are changing their positions, and the corresponding services must be reallocated. We will use an objective function considering the range from a physical sensor device to the service monitoring device which is physically placed in one of the Fog nodes to demonstrate this scenario. The range in this case is only important for services which are communicating with sensors or actuators. The range is considered 0 independently of the Fog nodes. They are hosted in with the services which are processing data. A judgment matrix prioritizing the range was used in this scenario, meanwhile, the objective functions in this case are: Range, CPU, Security, RAM.

$$J_1 = \begin{pmatrix} 1 & 3 & \frac{1}{6} & 3 \\ \frac{1}{3} & 1 & \frac{1}{6} & 1 \\ 6 & 6 & 1 & 6 \\ \frac{1}{3} & 1 & \frac{1}{6} & 1 \end{pmatrix}; \qquad (6)$$

We used the data presented in the diagram (see Figure 11) to model the placement of the services. All the coordinates here are presented in meters.



(a).



(b)

Figure 11. Service placement diagram. (a) Initial placement; (b) modified placement

The best service placements in each case are summarized in Table 8 and Table 9, and the corresponding scores are $(13, 67, 128, 73)^T$ and $(9, 54, 86, 55)^T$.

Table 8. Best service placement in the third scenario, the initial placement of sensors and actuators

	Fog1	**Fog2**	**Fog3**	**Fog4**
Services	Process5 Process6	Process2, Process3, Process4, Activate1, Activate3	Process1	Sense1, Sense2, Sense3, Process7, Activate2

Table 9. Best service placement in the third scenario, the placement of sensors and actuators after their location changes

	Fog1	**Fog2**	**Fog3**	**Fog4**
Services	Process1, Process2, Process4, Process7	Sense2, Activate1, Process5	Process3, Process6	Sense1, Sense3, Activate1, Activate3

The evaluation results clearly show that if the range is the most important objective function, then the services are more likely to be placed in the adjacent Fog

nodes. On the other hand, if more sensors are located near a less secure Fog node, then, the overall security of the solutions may decrease (128 bits vs. 86 bits in the second scenario).

3.8. Conclusions

1. The dynamic orchestrator architecture and its components have been introduced. It explains an orchestration control cycle and a two-step optimization process which is used as part of a service orchestration method.
2. The two-step method uses *Integer Multi-Objective Particle Swarm Optimization* (IMOPSO) to find a Pareto optimal set of solutions and the *Analytical Hierarchy Process* (AHP) with an application-specific judgment matrix for a decision on any optimal distribution of services. Such a processing distribution allows to assess different heterogeneous criteria with different units of measurement and a different nature (qualitative vs. quantitative). The method, apart from providing one best solution, also ranks all the Pareto optimal solutions, thereby enabling to compare them with each other (by answering the question "to what extent one solution is better than the other"), and, if needed, to choose the second best, the third best, etc., solution.
3. A simulation was completed in *Matlab*. The available libraries enabled a convenient development. Charts were generated as part of the simulation results for better visual perception. It also allowed to save on expenses while testing proof of the concept since no additional hardware was involved apart from a PC. Three scenarios were used to illustrate different situations. The first scenario was a simplified one with only two objective functions to test the effectiveness of the method. The second scenario was aimed at the optimal placement of services with different matrix priorities. Meanwhile, the third scenario had to reallocate placed services due to range requirements as part of mobility.
4. Our proposed method effectively works with different evaluation criteria, and 4 of them were currently applied in a matrix, notably, Power, CPU, Security, and RAM, but it can be easily expanded by introducing new objective functions representing different criteria after changing the matrix size from 4 to 5 or more. Moreover, the objective functions may be dynamic, which means that not only the value but also the algorithm of an objective function calculation may be different based on service placement in particular Fog nodes with particular software and hardware capabilities.
5. If the same End device, service and Fog device set is used in a different application area (i.e., healthcare vs. home automation), which would require different priorities of criteria (i.e., security is more important in healthcare

compared to home automation), then only the AHP judgment matrix must be changed. The method adapts to a specific situation and provides the appropriate results.

4. MULTI-AGENT FOG COMPUTING SERVICE DYNAMIC ORCHESTRATION METHOD IMPLEMENTATION

This chapter is dedicated to the implementation of a dynamic Fog Computing service orchestration method prototype based on IMOPSO and AHP. As a result, a service placement orchestration is proposed, which functions as a multi-agent system. Fog Computing services are represented by agents which can both work independently and cooperate. Service placement is completed by a dynamic orchestration method using a two-stage optimization process. This service placement orchestrator is distributed, services are discovered dynamically, resources can be monitored, and communication messages among the Fog nodes can be signed and encrypted as a solution to the weakness of multi-agent systems due to the lack of monitoring tools and security.

4.1. Design Motivation

The aim of this orchestrator design process is to demonstrate how orchestrators make decisions to control their services in respect to the QoS and security requirements. Each decision the orchestrators make to start/stop/move their services must be verified in a dynamic way, whether the minimal requirements related to various hardware and software restrictions of the involved hardware devices, as well as requirements due to peculiarities of the application area (e.g., sensitive data should be protected better than environment monitoring data), are met.

The orchestrator design should consider its three main stages:

- Each orchestrator as part of the first stage should take into account such requirements as security, CPU, RAM, and power based on the application area and diverse fog node hardware/software capabilities to decide if it is possible to launch all the required services without violating these requirements.
- The second stage is to find an optimal distribution for deployable services among different fog nodes. It can be vital for saving energy and computation resources in cases when some services need to be stopped, suspended, or moved to other fog nodes.
- The third stage is dynamic service placement for a situation when circumstances change during the runtime, and orchestrators need to change the distribution of their services among the available fog nodes according to these new conditions.

The orchestrator in the first stage should have all the information of services and placements in all the fog nodes collected and synchronized, and it should be aware of the available QoS and security requirements. If all minimal requirements are satisfied, services can be launched. During the second stage, the orchestrator should search for an optimal placement of its services. In the third stage, the orchestrator should detect

certain changes in its resources or environment indicators, process these data, and relocate its services from the current fog node to another one.

Having considered the design requirements and knowing the benefits of multi-agent systems, it provides a good starting point. MAS behaves like a network that can correct itself and analyze itself [118]. These intelligent network nodes can both function individually and cooperate with the objective to pursue their general and individual goals. The integration of MAS constitutes a complex framework designed for system control and optimization. However, security issues must be addressed to maintain its resilience, and resource monitoring needs a solution due to the lack of an integrated tool.

Particle Swarm Optimization (PSO) is a population-based metaheuristic technique that is used to solve optimization problems [119]. It imitates a social behavior of birds where each bird in the flock, based on its individual experience and social experience, approaches their target food. It is the principle of social interaction to solve the problem. This technique is suitable to optimize continuous non-linear functions. As in the wild, with a flock of birds, here, PSO starts with a swarm of potential solutions. Each potential solution is represented by a particle. The population with each iteration is updated by updating the particle's velocity and position. These updates are based on the personal best value and the global best value. Each particle converges to its new position until the global optimum has been found. Multiple objectives, however, require IMOPSO for a set of non-dominated service placements.

The *Analytical Hierarchy Process* (AHP) is used as a second technique to choose the best solution from a Pareto set. AHP, which was developed by Saaty, is a technique that helps to simplify complex and poorly structured problems by making a number of pairwise comparisons [120]. Decision criteria are organized in a hierarchical way, and they are given their weight coefficients based on a potential impact to achieve the desired goal. At the end of the process, the best service distribution alternative is chosen, which corresponds to the highest criteria priority as the final orchestration method output.

In order to meet the design requirements, the whole orchestrator design process is broken down into separate subsections, which include resource monitoring, starting new services, data synchronization, security maintenance, and the orchestrator architecture itself. All these subsections are needed to design a service placement orchestrator as a multi-agent system that overcomes its inherent shortcomings of network attack vulnerabilities and the absence of an integrated monitoring tool.

4.2. Resource Monitoring

Resource monitoring is implemented by using a monitoring agent. It uses a cyclic behavior to wait for messages. The content of these messages is filtered with the method startsWith() in order to take a relevant action. It can get messages from a battery voltage agent (BattVoltAgent), light agent (LightingAgent) or a sensor agent (AbstractSensorAgent). The battery voltage agent keeps track of the Raspberry Pi 4 battery charge level as a fog node. Pi 4 does not have a native *analogue-to-digital* (ADC) conversion option. An external one, such as an MCP3424 18-Bit ADC-4

channel converter, would be needed. However, a high or low pin 3 voltage is checked for simulation purposes by using the Pi4J library.

A sensor agent is used to monitor such external resources as the light intensiveness or temperature, etc. A 5 second interval is used as a sensor update interval which involves communication between end-devices and particular agents and as a sensor agent poll interval which involves communication between agents and the orchestrator. There are a few commands which a sensor agent can receive, such as Get, Set, Move, Changeip. If the sensor agent receives a Get message, it identifies the sender and responds with a value which is obtained by using the getValue() method as an ACL INFORM message. These messages are sent as part of a regular sensor polling task at a predefined interval, e.g., 5 seconds. It can also be triggered once a threshold value has been reached.

Communication with the end-devices to get their values is done by using the COAP protocol. Lighting and temperature agents keep on polling their end-devices for their values as part of an onTick() event which is periodically triggered after a timeout interval. The COAP request method GET is used to get a value from the end-point device. The PUT method is used to set a value instead.

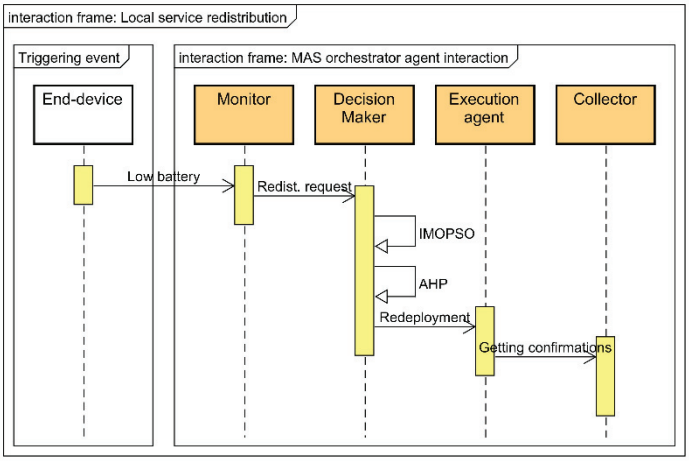


Figure 12. Monitoring initiated service redeployment

As shown in Figure 12 above, an end-device keeps on periodically communicating with its fog node. Once a Monitoring agent has detected that its battery voltage is below the required level, it sends a service redistribution request to the fog node Decision Maker agent. A remote service redeployment is calculated by using IMOPSO and AHP, and its solution is communicated to the Execution agent. Services in the current fog node get stopped. A redeployment solution is synchronized by the current fog node and the fog node where the services have to be moved to. A synchronization agent sends a request to its Execution agent, and the end-device service gets assigned to fog node 1.

4.3. Service Discovery

Service requests are generated when a new end-device appears, or the current end-device is moving from one place to another. End-devices at the current fog node are getting disconnected, and they need to send a request to a closer fog node. These end-devices are identified by their end-point address, such as coap://192.168.0.24/temp for a temperature device or coap://192.168.0.24/led for lights which can be changed or adjusted as required; see Figure 13 for more details.

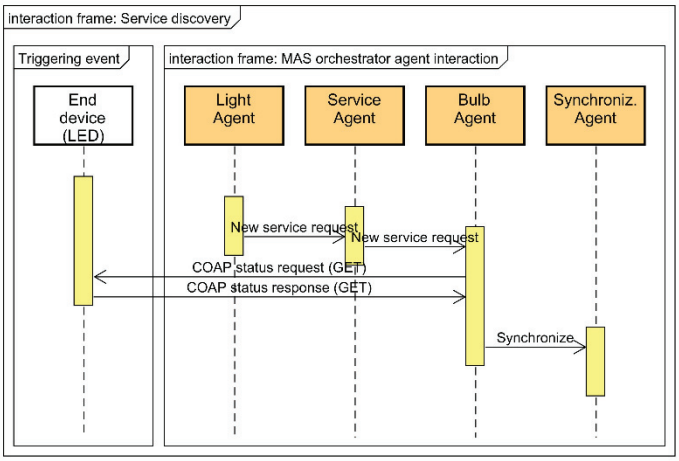


Figure 13. Service discovery

The Light agent works as a light sensing element. If the light level outside falls below a certain threshold, it sends a request to its Service agent to start a new service. The Service agent defines what that service should be like. When that is a light service, it can define what intensiveness of the light is needed. The Service agent afterwards sends a defined request to the Bulb agent. It uses the PUT method to communicate with the end-device to set a required level of light. As the level gets adjusted, the outcome is synchronized.

4.4. Data Synchronization

Currently, synchronization among orchestrator agents and communication among fog nodes is implemented as a three-way communication topology. Upwards communication is bound for vertical synchronization with a parent agent by going one level up within the hierarchy. Downwards communication is meant for sending messages down by one level to a child agent. It is being done by sending ACL INFORM messages within internal fog node agents. Sideways communication is horizontal communication among nearby fog nodes, and it is done by synchronization agents. All the fog nodes need to keep their data about resources and statuses updated in order to make their informed decisions for the best possible service placement when starting a new service. It is also necessary when currently available services are getting relocated due to resource or security restrictions. All the horizontal communication at

the moment is done via Wi-Fi, but it can also be done via *Bluetooth*, *ZigBee* or even *Ethernet*; see Figure 14 for a communication message example.

```
 :sender  ( agent-identifier :name FogOrchestrator4
@fog4  :addresses (sequence http://192.168.0.90:7774 ))
 :receiver  (set ( agent-identifier :name FogOrchestrator4
@fog4  :addresses (sequence http://192.168.0.90:7774 )) )
 :content  "AgentList:4:9:FogOrchestrator4:sniffer0:sniffer0-
on-fog4-cnt:Temp4:sniffer1:Light4:Decision4:Bulb4:sniffer1-
on-fog4-cnt"
```

Figure 14. Agent list synchronization messages

In order to keep the agent list synchronized, it gets updated by using the AMS Service component. This component allows to launch a search based on certain constraints and descriptions. It monitors agent registration, deregistration, and tracks them. A GetAgentList message is sent to all known orchestrators in the fog nodes to retrieve a list. Meanwhile, there is a list of possible orchestrators with their IP addresses stored, but their presence in the fog network is confirmed with a response.

4.5. Security Maintenance

As a default configuration, agent communication messages are neither encrypted nor signed. It may give an opportunity for a hacker using a malicious agent to sniff a message content, or even modify it. It would lead to malicious instructions for a recipient agent. Different requests can be sent to AMS to eliminate some agents if there are no security checks. To maintain security, JADE security add-on JADE-S was used. After it has been installed, services such as SecurityService, PermissionService, SignatureService and EncryptionService have to be enabled. SecurityService is a primary one, and the other ones are optional which can be enabled as required.

Instead of using the method send(), the method sendMessage() is used. It allows to specify whether the message has to be signed and encrypted. Credentials are checked by the method retrievePrincipal(). It is triggered first before a relevant message is sent. It first sends a request message with the content ‘get-principal’, and the answer is formed by a recipient to send it back as an inform message; see Figure 15 for more details on secure communication.

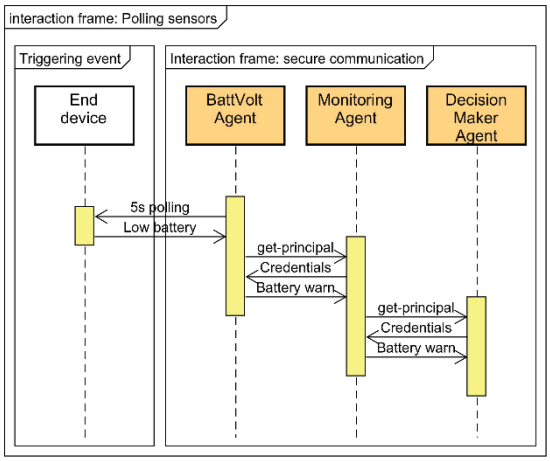


Figure 15. Retrieving credentials in secure communication

In addition to the points outlined above, further modules have to be enabled to adjust the default security settings to the preferred ones. SignAlgorithm allows to choose an algorithm which will be used to have the messages signed, such as MD5withRSA or DSA. The size for public and private keys can be defined by the module AsymKeySize which ranges from 512 as a default value to 2048. There are a few more modules to the above ones to ensure additional customizability.

4.6. Dynamic Orchestrator Architecture

A service placement orchestrator was implemented as a multi-agent system (MAS) Java application using a JADE framework. It allows to develop MAS applications which comply with the FIPA specifications. It offers such features as agent abstraction, asynchronous messaging, or service discovery based on the yellow pages method. The orchestrator is implemented as a distributed monitoring, decision making and execution method which is available in each fog node within a relevant Fog Computing system. Continuous resource monitoring and request processing allows to make informed decisions in a dynamic way. Service distribution among multiple nodes contributes to the enhancement of the computational output, power usage and resilience. A distributed orchestrator is more resilient than a centralized one because of the absence of a single point of failure (SPF). Mobility, changing resource levels and fog node failures preferably lead to dynamic and distributed decision making.

Once the fog nodes have connected to the same network to form a shared infrastructure, they start monitoring their resources, such as CPU, RAM, battery, and security. The monitoring agent waits for resource-related messages using a cyclic behavior. These messages are classified by use cases and tagged with the relevant resource levels. A request agent is waiting for service request messages. They are also classified by use cases. Due to the security add-on JADE-S, messages can be either signed, or encrypted, or both signed and encrypted. The required security measures and the used security measures are identified among the involved agents, and the

messages can be discarded if these requirements are not met; see Figure 16 for an orchestrator architecture [121].

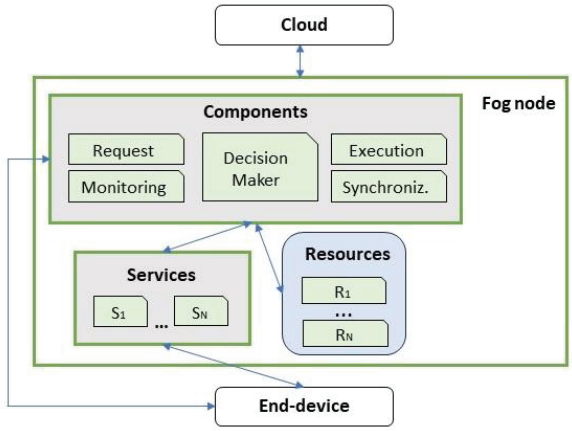


Figure 16. Distributed orchestrator architecture

The Decision Maker agent can get messages either from the Request agent, the Resource Monitoring agent, or from the Synchronization agent. Messages from the Request agent can be related to a new service request or a current service redeployment. The Monitoring agent keeps its Decision Maker informed if the resources are below a required level. There is also the synchronization agent which keeps the resource data synchronized among the involved fog nodes to let the Decision Maker make informed decisions when choosing a local or a remote service distribution. The Decision Maker uses IMOPSO and AHP algorithms to calculate the best deployment based on the objective functions. IMOPSO is used as a primary request processing algorithm for a higher number of options. Once these options are considered by the Decision Maker, a potential solution is further processed with a reference to objective functions. The AHP algorithm uses its criteria matrix to make a final placement decision based on the prioritized criteria. 4 criteria are used at the moment, but this number can be adjusted as required.

As soon as a placement decision has been made, it is communicated to the Execution agent. It runs a cyclic behavior waiting for its messages. The received messages are classified based on the use cases or key words, such as ‘Decision’ for internal purposes. If there is a local redeployment within the same JADE platform, agents can be moved from one container to another. Agents cannot be moved, however, to another platform. When services have to be redeployed remotely, the current agents are ‘killed’, and other ones with the same parameters are created in a remote platform. Any changes in the Fog Computing infrastructure – whether they involve a new service placement or a remote redistribution – are synchronized by a Synchronization agent. Additional details about the architecture are available in the publication [5].

4.7. IMOPSO Prototype Implementation

The launch of a Jade Multi-Agent Platform Orchestrator starts by providing the platform with such required parameters as platform-id, container-name, local-host, port, message transport protocol, active services and agents which will be created by the Launcher class. The ImopsoSolver() class is subsequently used to provide other constants including the criteria count (critCnt), the fog node count (fogNodeCount), the service count (serviceCount), the particle count (partcount), and the epoch count (epochCount). The criteria count represents a quantity of the objective functions. Currently, we are using 3 or 4 in this Particle Swarm Optimization, but it can be changed depending on the number of objectives used for our optimization. These functions are used to calculate the results. Our results, the particle positions, are added to a list and afterwards returned along with some other data. The Fog node count is a count of fog nodes, and currently we are using 4 of them to distribute 12 services. A given number of particles in the constant partcount is generated by repeating a cycle for a number of times given as an epochCount; see Figure 17 for the IMOPSO algorithm which is implemented by using Java classes and methods.

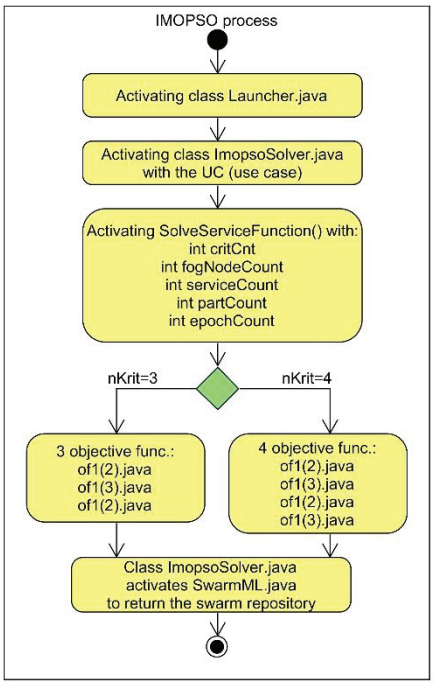


Figure 17. IMOPSO algorithm classes and methods

Random particles are generated to initialize a Pareto set. Vectors get their initial velocities assigned. After the global best positions and global best score have been initialized, the initial best scores and the best positions are assigned to each particle. The values are calculated by using objective functions.

The particle repository is updated based on the state of the particles. A particle can dominate, be dominated, or neither dominate nor be dominated. After the Pareto Dominance has been determined, the new velocity is calculated by subtracting new vectors. If a particle is out of range, it is given the opposite value; see Figure 18 for more details on repository updating.

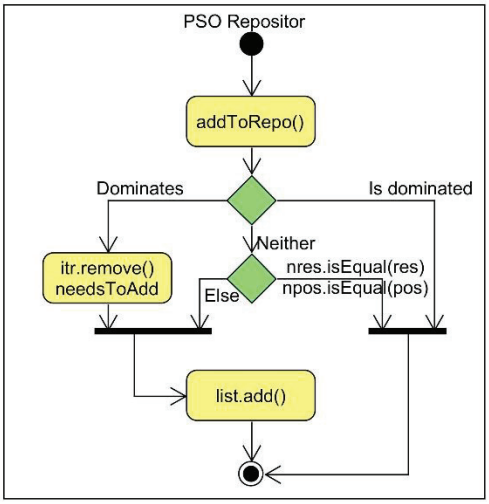


Figure 18. Repository updating

The particle repository update is triggered by the function addToRepo(). If a particle is the first one in the repository, it always gets added. If a particle dominates, the previous one is removed, and a new one is added. If a particle neither dominates nor is dominated, it checks whether the result is in different points.

4.8. AHP Prototype Implementation

After IMOPSO has been completed, another step is to use AHP. It will be used for particles which are added to the particle repository which is initiated as an object called *prepo*. *Prepo* contains the global best positions of particles and coordinates. The method getParticleAt() identifies the particle by its index in the array. The method getPosition() is used to return a particle which is identified by its index, and the method .getResult() is used to return the coordinates of that particle, which, as an example, looks like (2,3,2,2,2,3,2,2,2,2,2,2,) (2.0, 10.0, 2.0, 10.0).

The method comparePairs() compares neighboring particles as *pp1* and *pp2*. These particles are stored. They are compared for each criterion (4 in this case) by assessing their coordinates. A potential outcome of this result is 8, 1/8, or 1. These results are added to a double array allComp[][], the size of which is defined by the number of criteria [nKrit] and alternatives [nAlt].

The priority matrix is initialized as a double array C[][] by the method initializeMatrix() in the class AHP. The number of criteria (nKrit) represents a priority

matrix, and there are 3 or 4 criteria matrixes as required by the quantity of the objective functions. It is further used by the method getEigenFinal().

The method updateBestEigenValue () of the class AHPSolver initiates the method findBestElement() of the same class to find the best value. This process is being done by determining the length of the array eigenVector[i] in order to compare each element to find the highest value. The index i is assigned to a variable bestPos and returned after this cycle has been completed if the value of eigenVector[i] is above the previous maximum value; see Figure 19 for an AHP algorithm which is implemented by using Java classes and methods.

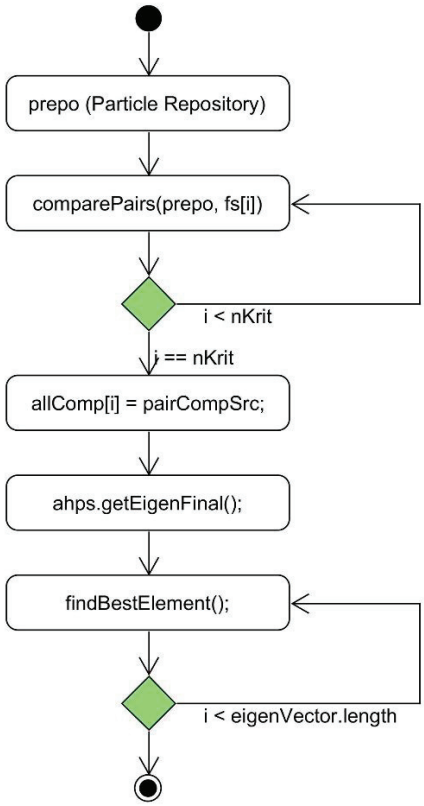


Figure 19. AHP algorithm classes and methods

Eigen results are calculated by the method getEigenFinal() of the class AHPSolver. It is used for both criteria matrix value calculation and for repository particle value calculation. The method initializeMatrix(), class AHP, initializes a matrix. While initializing it, the size is defined vertically and horizontally by the variable *n*, which equals to the length of the criteria matrix or particle repository *prepo*, and the values can be either 1, pairComp[k], or 1 / matrix[j][i].

An initialized matrix is further processed by the method sumColMatrix() to sum up the values within each matrix column. This sum is used to divide each element

with the method devidedColBySum(). Finally, the method normEigenVector() normalizes the matrix. The matrix row values are summed up and divided by the number of values. Normalized Eigen vectors are stored in the array double[] eigenvectorC for criteria and in double[] eigenvector for repository particles.

The Normalized Eigen vector repository particle matrix is transposed by the method transposeMatrix(). Columns and rows are switched up. The method eigenVectorFinal() will use it to calculate its values. The criteria matrix Eigen vector eigenvectorC[] values are multiplied by particle repository normalized Eigen vector values in eigenVec[], and the result keeps adding to itself until all the criteria have been taken into account for each used particle within the transposed matrix double[][] eigenAllT; see Figure 20 for more details on getting the Eigen final values.

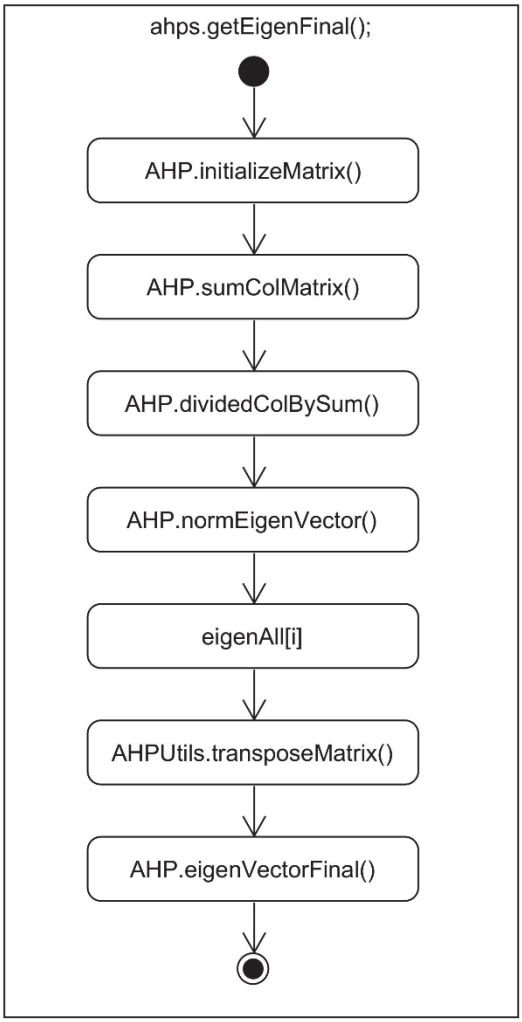


Figure 20. Method getEigenFinal()

4.9. End-Device Layer

A NodeMCU module with a processor ESP8266MOD was used for the end-devices layer. Our NodeMCU development board is designed for IoT applications [122], it is an open-source and low-cost device. Wi-Fi and Bluetooth connections are integrated. It is possible to program NodeMCU with AT commands, Lua Scripts in the NodeMCU firmware, MicroPython and Arduino IDE. However, Arduino was chosen due to higher availability as well as its capability to work with C++ libraries.

Communication between the nodes and the end-devices is ensured by using the CoAP (*Constrained Application Protocol*) protocol with UDP as a transport. CoAP is a transfer protocol which is designed for use with constrained nodes and networks [123]. It allows to send request/response inquiries to the endpoints of end-devices; it also has a built-in feature to discover services and resources. The mjCoAP library is used for a Java application in the prototype, meanwhile, the Thing.CoAP library in C++ is used for NodeMCU. Arduino is based on C++, and it has good integration. The Thing.CoAP library platform can be implemented both as a server and as a client, but only the server part will be used in this prototype since NodeMCU functions as a server in this case.

In order for a NodeMCU end-device to be able to accept requests and to return statuses, one has to create the end-points. These end-point addresses are also included in the FogConstant.java file of a fog node as well; see Figure 21 for end-point address examples.

```
public final static String bulbAddr = "coap://192.168.0.24/LED"; //t
//public final static String bulbAddr = "coap://192.168.0.24/led";
public final static String lightAddr = "coap://192.168.0.24/LED";
public final static String tempAddr = "coap://192.168.0.24/temp";
public final static String humidAddr = "coap://192.168.0.24/humid";
```

Figure 21. CoAP endpoints

The GET and PUT methods are used to send requests from a fog node to the end-device, but CoAP also supports the POST and DELETE methods. The GET method is used to retrieve the information which is available under a certain CoAP URI. The PUT method is used to update or create a resource with the payload; see Figure 22 for pin change messages.

```
}).OnPut([](Thing::CoAP::Request& request) {  //We are here cor
  Serial.println("PUT Request received for endpoint 'LED'");

  //Get the message sent fromthe client and parse it to a strir
  auto payload = request.GetPayload();
  std::string message(payload.begin(), payload.end());

  Serial.print("The client sent the message: ");
  Serial.println(message.c_str());

  if(message == "0") { //If the message is "On" we will turn th
    digitalWrite(LED, 0); //0 reiskia ijungti
  } else if (message == "1") { //If it is "Off" we will turn th
    digitalWrite(LED, 1); //1 reiskia isjungti
  } else { //In case any other message is received we will resp
    return Thing::CoAP::Status::Content("1");
  }

  //In case "On" or "Off" was received, we will return "Ok" wit
  return Thing::CoAP::Status::Content("0");
});
```

Figure 22. NodeMCU state pin change

A state gets changed in the example above based on the received PUT request. A status is then sent back to the fog node which functions as a client. The agent as a part of a multi-agent system accepts an update response. This response is further used for an orchestrator to make decisions on service distribution; see Figure 23 for temperature measurement messages.

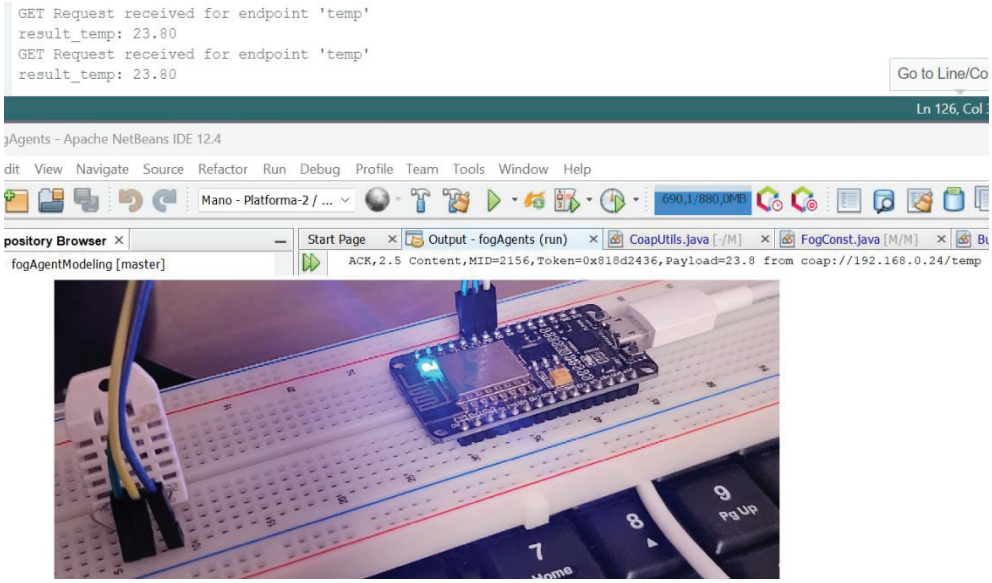


Figure 23. NodeMCU transmits temperature to the prototype

A DHT22 temperature and humidity device is used for readings. Adafruit/DHT-sensor-library is one of potential Arduino libraries for this implementation.

4.10. Software and Hardware Tools

A Raspberry Pi 4 Model B computer with 4 GB of RAM was used as low-resource fog node. It runs Raspbian 10 OS as its operating system and JDK 1.8.0_333 as a Java Development Kit. A personal computer Asus with 11th Gen Intel(R) Core(TM) i7-1165G7 and with 32 GB of RAM was used as a high-resource fog node. It runs Windows 11 Pro OS as its operating system with JDK 1.8.0_333 to keep exported 'jar' files compatible with JDK in Pi 4. A NodeMCU [124] development board was used as an end-device. It uses an ESP8266 [125] microcontroller with integrated capabilities for GPIO, PWM, IIC, 1-Wire and ADC; see Figure 24 for the hardware setup.

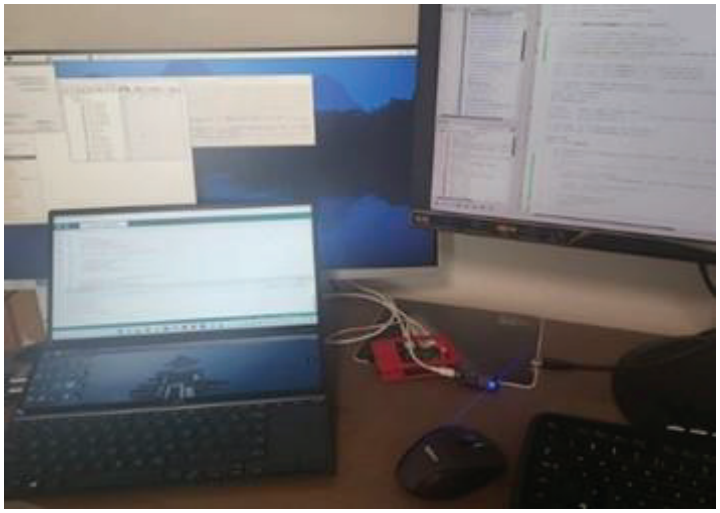


Figure 24. Experimental hardware setup

JADE [45] was used as an agent development framework. It is an open-source platform for multi-agent Java applications. This framework offers a simple and powerful task execution, peer-to-peer agent communication using asynchronous messages, service discovery based on the yellow pages approach, and a possibility to integrate some add-ons and services. As a security solution, JADE-S add-on was used. It enables user authentication, message signing and encryption as an option.

The development environments which were used include Apache NetBeans for Java applications. Arduino IDE was used for NodeMCU applications as for an end-device using the C++ programming language. Compiled Java applications were uploaded to Pi 4 by using the WinSCP client application. Calculations were performed, and charts were concluded mainly in Microsoft Excel. Occasionally, it was done by using Python in Visual Studio Code.

In order to measure the voltage and the current, a USB tester UNI-T UT658B was connected to a power adapter. As a second option, the energy meter PeakTech 9035 was used for power calculations.

4.11. Experimental Metrics

In order to determine the time in the experimental set up so that to identify how fast a solution is given by the orchestrator, a special class which is called the GlobalTimer was developed. This class uses the method System.nanoTime(). It returns in nanoseconds the current value of a running Java Virtual Machine's high-resolution time source [126]. This method is designed to measure only the elapsed time. To avoid any negative impact of logging events, logged events are stored only in RAM and printed as required only after the process has been completed; see Figure 25 for an event logging class.

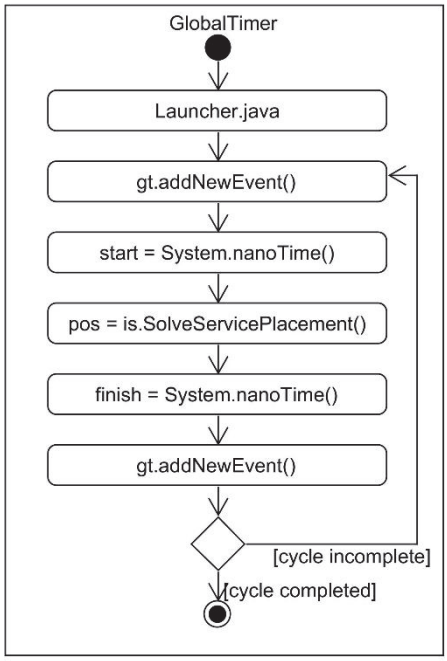


Figure 25. Logging new events

The method GlobalTimer has the class addNewEvent() which handles all the attempts to log the data. Logs are stored in 3 TreeMaps based on the data type they are identified: eventListEid, eventListAgent, and eventListTime. Logged events can be printed by the agent name using the method gt.printAllEventsDetailsByAgent(), by Eid using gt.printAllEventsDetailsByEid(), or by time using gt.printAllEventsDetailsByTime(); see Figure 26 for more details.

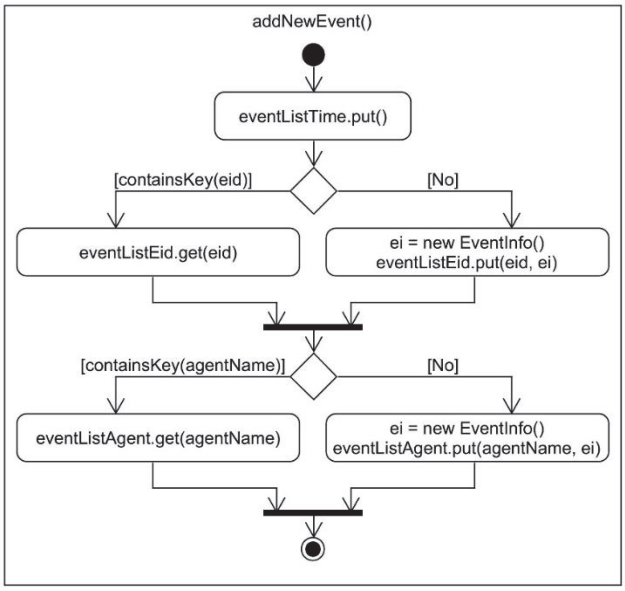


Figure 26. Adding events to relevant TreeMaps

In order for PSO iterations to be successful, it has to lead to a convergence. As Modiri and Kiasaleh [127] claim, convergence can be faster at the expense of a higher number for errors. Also, it can have a lower number of errors, but the performance will be slower. Therefore, such a success rate can be used to evaluate a convergence performance as metrics in an attempt to find the right balance.

While performing stress tests, a specifically designed stress test package was installed on a Raspberry Pi 4. It allows to overload gradually a CPU or an RAM with a certain number of workers. Workers act as a workload, which increases with an increasing number of the number of workers, thus leaving only a certain percentage of the available resources unoccupied.

Specific calculations, such as battery level sensing, placement finding, or service redistribution, did not seem to exert a diverse impact on the power demands. The main factor was the duration of a specific process. Therefore, it was reasonable to use electrical energy measurement (mW/h).

4.12. Optimization Coefficient Impact Evaluation Results

For the IMOPSO algorithm to function optimally, at the highest level possible, it is necessary to consider such coefficients as inertia, cognitive, and social. The inertia weight was first introduced in [44]. The purpose of the inertia weight w is to balance between the local search and the global search. When the inertia weight w is small, PSO works like a local search algorithm, and if there is a solution within the initial search area, a global optimum will be found quickly. Otherwise, it will not be found at all. When the inertia weight w is high, it focuses on the global search; see Figure 27 for more details.

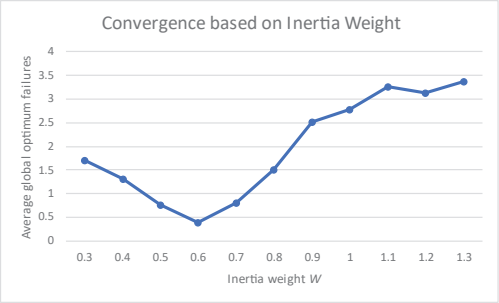


Figure 27. Inertia weight impact on convergence

As a default setting, 4 fog nodes, 12 services and 200 epochs were used. The number of particles ranged from 50 to 500. The range of the inertia weight was from 0.3 to 1.3. The cognitive and social coefficient was 1.499, which had a slight adjustment to the value of 1.496180. As per publication [45], it leads to a convergent behavior. As it is visible from the test results, an inertia weight coefficient of 0.6 demonstrated the best outcome in finding a global optimum. It can be considered as a threshold value which will be used in further experiments.

The inertia weight range between 0.3 and 1.3 was used in our next experiment to test the inertia weight impact in respect of a particle count and the response time; see Figure 28 for more details.

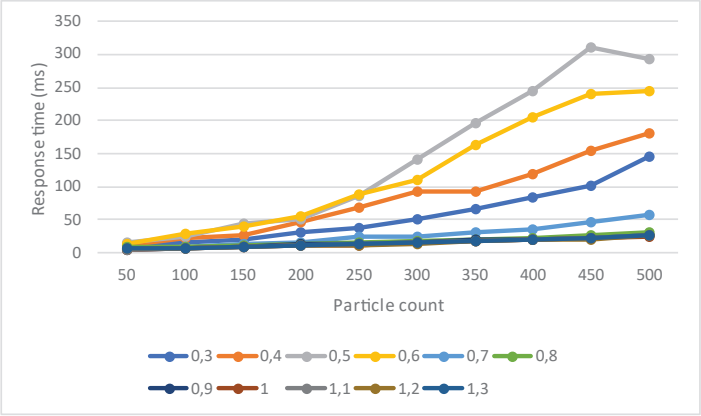


Figure 28. Response dependency on particle count based on inertia weight

As the chart curves suggest, a response time for an inertia weight of 0.7 is low, and the global optimum can be found quickly. When using an inertia weight of 0.3 to 0.6, it significantly increases the duration which is needed to complete its algorithm. Using an inertia weight of 0.8 to 1.3 will make it only slightly faster, but the risk of failing to find a global optimum greatly increases, especially with a low particle count.

The response time is considered as a comparison factor in the experiment above. However, the following experiment considers the number of failed convergence attempts within the range of an increasing particle count. The number of errors means the number of failed convergence attempts for a 20-attempt cycle with each particle

count. The average error rate is getting lower as the particle count is increasing. It suggests that it happens due to a higher particle population size. The higher the size is, the higher are the chances that it will succeed; see Figure 29 for more details.

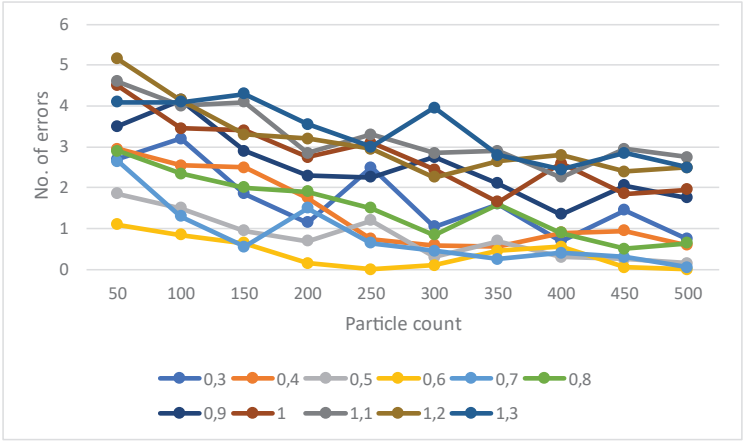


Figure 29. Convergence dependency on particle count based on inertia weight

Different fog nodes may have different technical characteristics since heterogeneity is part of Fog Computing. They also may have constrained resources and different QoS requirements. We use a judgement AHP matrix to evaluate such a situation and to prioritize different criteria. All the experiments are mainly completed by using a 4 criteria matrix. It includes power, CPU, security, and RAM. Power primarily in our experiments is expressively prioritized over the other criteria. There is no particular reason for this, and any criterion can be adjusted as required; see the matrix below for more details.

$$Q = \begin{bmatrix} 1 & 7 & 7 & 7 \\ 1/7 & 1 & 2 & 2 \\ 1/7 & 1/2 & 1 & 2 \\ 1/7 & 1/2 & 1/2 & 1 \end{bmatrix}$$

The purpose of the experiment outlined below is to test how much delay can a certain number of criteria contribute to our optimization process. As a default setting, 4 fog nodes, 12 services, 50 particles and 200 epochs were used. A PC and a Raspberry Pi 4 are used as fog nodes to compare a powerful fog node and a fog node with limited hardware resources; see Figure 30 for more details.

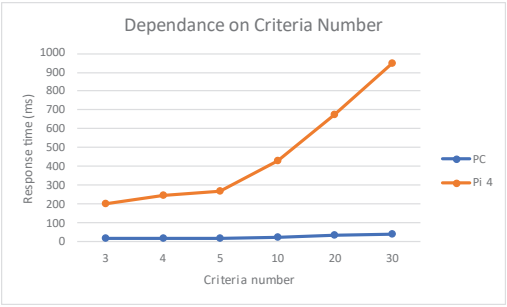


Figure 30. Algorithm optimization response time based on the number of criteria

The range of 3 to 5 criteria is mostly expected. However, in order to test a broader scope, the range of 10 to 30 is also included. 3 to 5 criteria generate only 3 to 10 comparisons within the matrix. It has a little effect on the response time. However, 10 to 30 criteria generate 40 to 435 comparisons, and the response time significantly rises in a hardware-restricted fog node device. Still, a 1 s response time is manageable in real applications even though 30 criteria are barely ever needed.

4.13. New Service Starting Evaluation

In order to evaluate a New Service Starting Latency, the following experiment was conducted. Agent communication is illustrated and verified by using Sniffer which is available on the JADE platform.

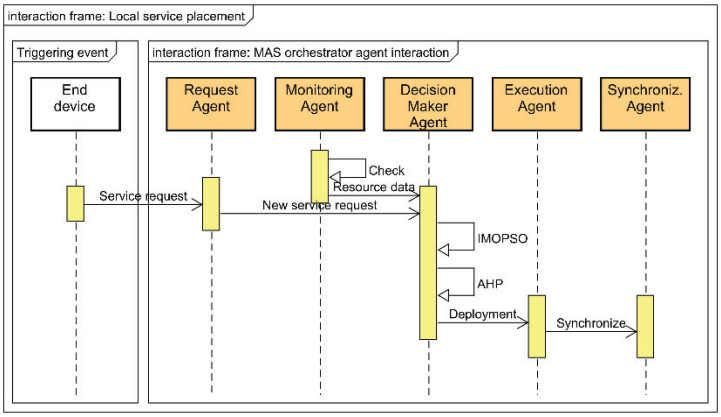


Figure 31. New service starting locally

Figure 31 above demonstrates the whole process which is needed to get new services running in a relevant fog node. The whole process starts with an end device communication with a fog node. The agent of an end device generates an event and sends a message to a Request agent. The Request agent sends a message to a Decision agent. This procedure is identified as an End Device Communication, and it takes a relatively significant period of time only when computational resources are high, as it

is visible from the PC bars in the Figure. When the computational resources are lower, Placement Problem Solving has the biggest impact on the resource consumption.

When a placement has been found, a message is sent by the Decision agent to an Execution agent. Deployment is rather quick, and it does not depend significantly on the resources. When it has been completed, a message is sent to the Collector agent to finalize the process.

A new service deployment can take slightly more time when services are deployed in a remote agent. It means that a communication between the agents of two different fog nodes will be involved. When a placement solution is made by the agent Decision1, a message is sent to the agent Exe1. Exe1 thereafter sends a message to a Sync1 agent, which is meant to synchronize events between the participating fog nodes. It is Sync0 in this case as an external fog node agent which gets informed by Sync1 via a remote message. Sync0 sends a message to Exe0 for a new service local deployment, and, after it has been completed, a final message is sent to Collector0.

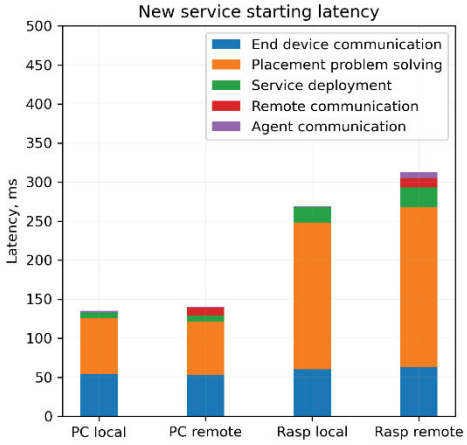


Figure 32. New service starting latency

Resources are mostly relevant to the Placement Problem Solving as it is visible from Figure 32 where the results are given as relative values by each component. End device communication is the second factor of latency to consider. Other components did not play a significant role here; see Figure 33 for more details on component dependency.

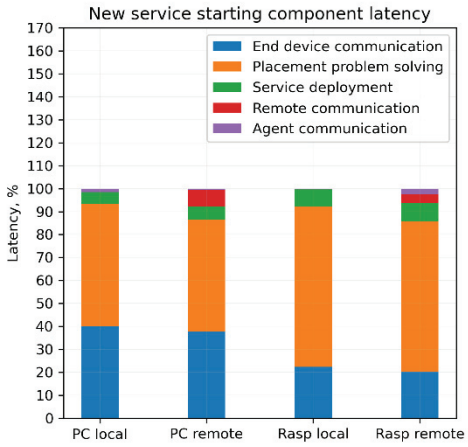


Figure 33. New service starting component latency

4.14. Current Service Redistribution Evaluation

In order to evaluate the Service Redistribution Latency, the following experiment was conducted. Agent communication is illustrated and verified by using Sniffer which is available on the JADE platform.

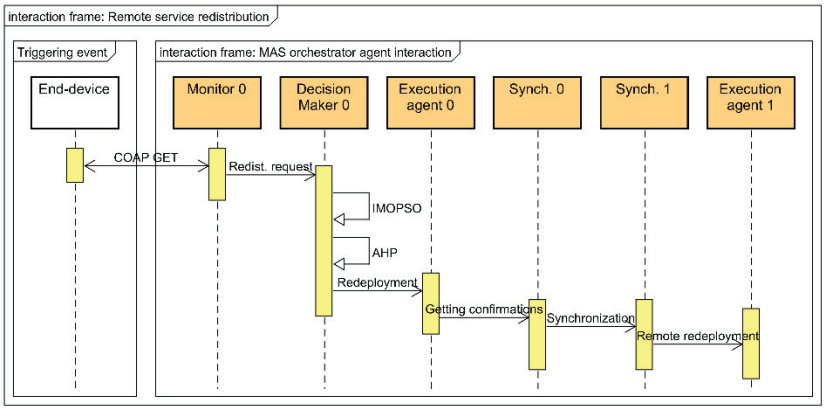


Figure 34. Remote service redistribution

Figure 34 above illustrates an example of service redistribution for the currently available services. A low battery event is generated, which serves as a trigger to relocate services from fog node No. 1 to fog node No. 0 in order to keep them available. Along with the moving three end-device services such as Light1_0, Service1_0 and Bulb1_0, 9 services in addition were used to have a Service Placement Problem solved by the Decision Making Agent.

As the event of a low battery has been generated, Monitor1 agent detects it and sends a corresponding message to its Decision1 agent. This communication is

identified as Battery Level Sensing, and it takes a respective duration of time which is irrelevant to a higher or lower availability of resources. Higher calculation power is, however, significantly important to the Placement Problem Solving as it is visible in the Raspberry chart bar in Figure 35.

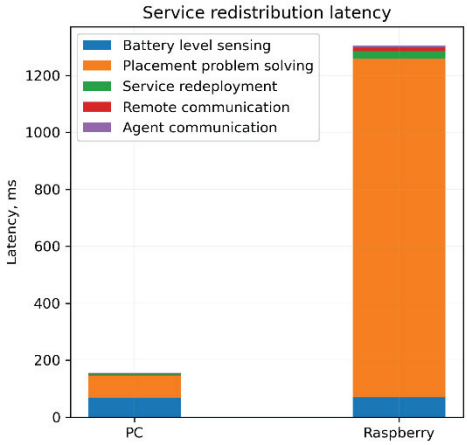


Figure 35. Service redistribution latency

Once a placement solution is available, the services Light1_0, Service1_0 and Bulb1_0 can be disabled locally and launched respectively in fog node No. 0. The agent Decision1 sends a message to the agent Exe1 to disable them. It has to be synchronized, and it is done by communication between Sync1 and Sync0 agents, which is identified as Remote Communication. The above-mentioned services get launched by the agent Exe0 which completes the moving of services, and a confirmation message is sent after that to the agent Collector0.

The following comparative chart demonstrates a relative latency by the components for a service redistribution scenario. Again, Battery Level Sensing can play a relatively significant role in devices with higher calculation power availabilities, just as it was with the end-device communication in a new service distribution scenario. A Placement Problem Solving is directly dependent on resources, however; see Figure 36 for more details.

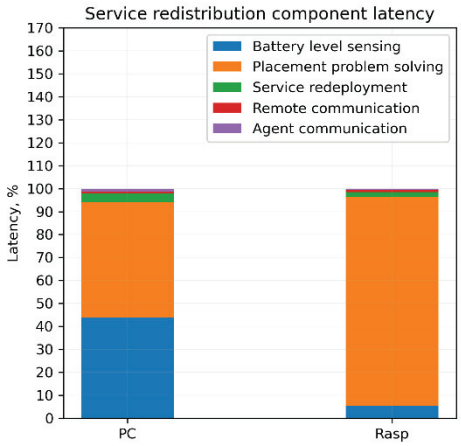


Figure 36. Service redistribution component latency

4.15. Scalability Evaluation

As it is visible from the previous experimental results, decision making takes a considerable part of time, and it requires further tests to define these functional dependences better. In the experiment described below, a matrix was used for an illustrative scenario with the following criteria: power, CPU, security, and RAM. Power in this case is expressively prioritized over the other criteria. See the matrix below for more details.

$$Q = \begin{bmatrix} 1 & 7 & 7 & 7 \\ 1/7 & 1 & 2 & 2 \\ 1/7 & 1/2 & 1 & 2 \\ 1/7 & 1/2 & 1/2 & 1 \end{bmatrix}$$

For each set of parameters, 20 attempts are made to perform the same experiment in order to calculate an average of the output values. These output values may slightly vary with each attempt since the IMOPSO method initially uses random locations for its particles. The following chart demonstrates a dependency between the time and the number of particles. It shows how much time it takes to generate a placement for a specific number of particles. A comparative calculation is made by using a Raspberry Pi 4 which is identified as Pi 4 in the chart and a personal computer which is labelled as PC; see Figure 37 for more details.

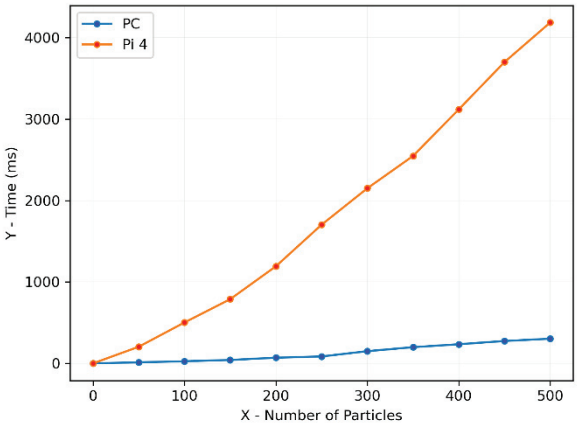


Figure 37. Delay dependency on particles

The following chart illustrates a dependency between the time and the number of epochs. 200 epochs were used for time dependency on the number of particles in the experiment above. 200 initial particles were used for time dependency on the number of epochs in the following experiment; see Figure 38 for more details.

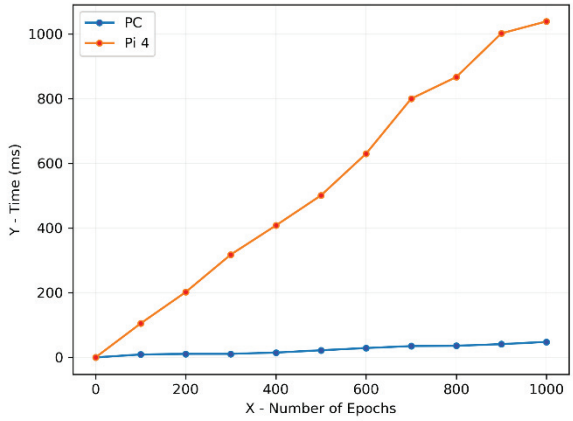


Figure 38. Delay dependency on epochs

The following chart demonstrates the service placement efficiency when the complexity increases [128]. The number of fog nodes, services, particles and epochs is gradually increased. Each point in the chart was calculated by obtaining an average of 20 attempts for each parameter set. 6 parameter sets were used which are listed below in the table. See figure 39 for a scalability chart. Table 10 summarizes the parameters used.

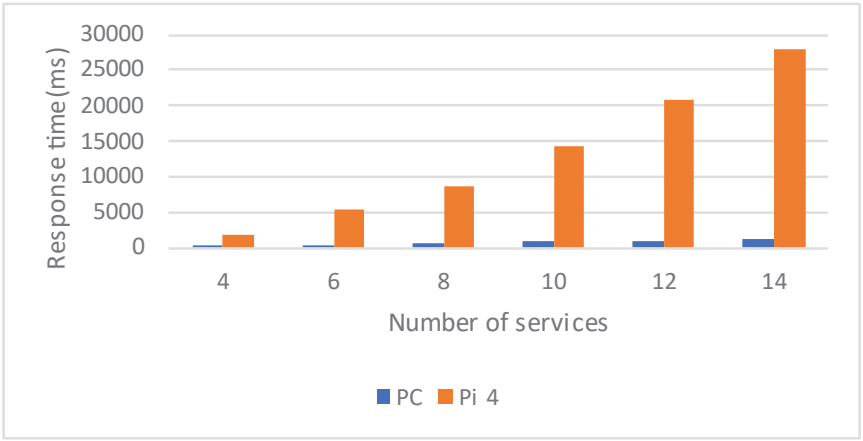


Figure 39. Scalability of the service placement finding algorithm

Table 10. Scalability evaluation parameter sets

Fog nodes	Services	Particles	Epochs	Response time (ms) for PC	Response time (ms) for Pi 4
4	12	200	300	121	1802
6	18	300	300	305	5432
8	24	400	400	498	8644
10	30	500	600	684	14401
12	36	600	700	747	20923
14	42	600	800	1161	27923

The above experiments allow to conclude on the following dependencies:

- The service placement time has a linear dependency on the number of particles and the number of epochs. In the experiments of a new service starting and service redistribution, 200 particles and 400 epochs were used by default. Any increase in these parameters will increase the placement delay in a linear manner.
- Service placement time has a linear dependency on the complexity of the problem which is defined by the number of fog nodes and services. In the experiments of a new service starting and service redistribution, 2 fog nodes and 12 services were used by default. Any increase in these complexity parameters will increase the placement delay linearly.
- A time delay of the service placement suggests that the complexity of the problem to solve can be defined by up to a few tens of fog nodes or services, especially with devices of a relatively low calculating power, like Raspberry Pi.

4.16. Hardware Stress Test Evaluation Results

The following experiment is meant to test an effect of CPU availability on the response time. The goal is to optimize CPU utilization in such a way that additional services can be hosted, or some background processes can take place without overloading the CPU. In order to learn an extent and the threshold to which a CPU can get overloaded by additional services, a CPU stress test was performed. See Figure 40 for more details.

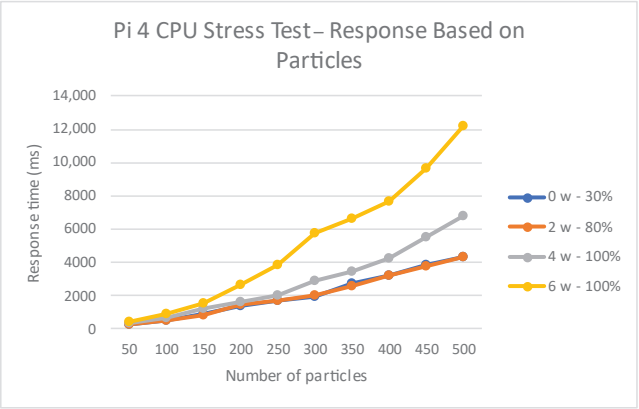


Figure 40. Optimization process response time based on CPU overloading

Raspberry Pi 4 was used as a fog node. The optimization process performance evaluation was done by using 4 fog nodes, 12 services, and 200 epochs as a default setting. The number of particles ranges from 50 to 500. To perform a CPU stress test, a stress tool was installed. A CPU was stressed with 0 to 6 workers, which means concurrent process threads that are launched in CPU to overload it. The algorithm itself overloads Pi 4 CPU at around 30% once it has been started. 0 to 2 workers do not seem to pose any negative effect on a CPU. However, 4 to 6 workers, which overload a CPU up to 100%, are critical. The services would have to be redistributed before such a level has been reached.

The following experiment is intended to test the effect of RAM availability. The optimization process performance evaluation was done by using 4 fog nodes, 12 services, and 200 epochs as a default setting. The number of particles ranges from 50 to 500. See Figure 41 for more details.

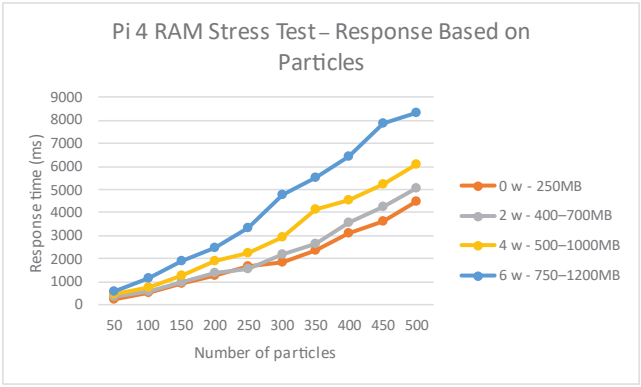


Figure 41. Algorithm optimization response time based on RAM overloading

Raspberry Pi 4 background processes initially overload RAM with about 220 MB. Once the algorithm has been launched, the level goes up to 250 MB. It suggests that the algorithm is not that much dependent on RAM as it is dependent on CPU. However, if there are some significant background processes or additional services, it can still slow down the algorithm, and the fog node services need a redistribution.

The following experiment is completed to test a low battery level. An external analog-to-digital module would be needed for this since Raspberry Pi 4 does not natively offer an ADC capability to detect a certain level of voltage. However, a low or high input was used for the test purposes. The class BattVolt.java was created to test a battery voltage level. The library P4J was used to read an input value by the BattVolt agent. The polling interval by default was 5000 ms.

When using Raspberry Pi 4 as a low-resource fog node, it takes up to 1.5 s to complete the whole process of service redistribution. When using a PC as a powerful fog node, it would require about 10 times lower duration. More details are given [16] by Aral and Ovatman. CPU overloading does not seem to affect battery level sensing or service redistribution. However, a two-stage IMOPSO and AHP algorithm is sensitive enough to overloading. Still, a few seconds to redistribute fog node services due to a low battery level might not be a problem in real-life conditions; see Figure 42 for more details. Table 11 summarizes the parameters used.

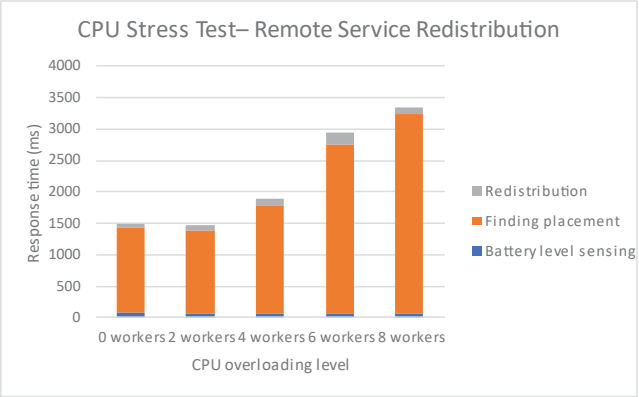


Figure 42. Service redistribution due to low battery

Table 11. Service redistribution response time

Process	0 workers	2 workers	4 workers	6 workers	8 workers
Battery level sensing	75	60	68	52	67
Finding placement	1359	1326	1723	2687	3167
Redistribution	63	78	97	202	113
Total time	1497	1464	1888	2951	3347

4.17. Security Package Impact Evaluation Results

The following five experiments are meant to test the security influence on the fog node response time while using different security levels and two fog nodes with different hardware capabilities. JADE-S, a security add-on package, was used as a security option [46]. This add-on allows to develop multi-agent applications with a certain degree of security, including guaranteed message integrity, confidentiality and authorization checks. It allows for agents and containers in a platform to be owned by authenticated users which are authorized by the platform administrator. Each agent has a public and a private key pair which is used to sign and encrypt messages.

The JADE security guide [] claims that the signing and encrypting of messages can slow down the agent communication performance, and that this is the reason why it is not done by default; yet, it is important to check to what extent it can happen. The optimization algorithm performance evaluation was done by using 2 fog nodes, 12 services, and 200 epochs as a default setting. The services have to be moved from one fog node to another due to a low battery level. The SecurityHeper() package is used with the methods setUseSignature() and setUseEncryption() to set a signature and encryption for a message. The methods getUseSignature() and getUseEncryption() of the same package are used to retrieve a signature and encryption. Default configuration values are used, such as an RSA asymmetric algorithm, and 512-bit key size is used for public and private keys.

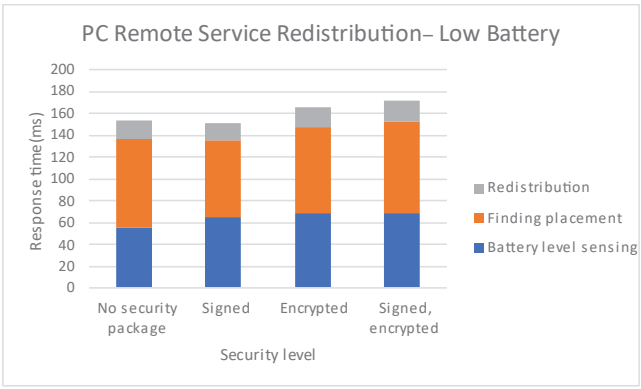


Figure 43. Security package impact on communication in high-resource nodes

As the chart above in Figure 43 suggests, 4 experiments with different security configurations were performed. It includes (a) no security package, (b) signed, (c) encrypted, (d) signed and encrypted. A PC was used as a high-resource fog node, and the services had to be moved from one fog node to another due to a low battery level. Such a remote service redistribution did not demonstrate any significant response time variation because of different security approaches. There is a slight response time increase mainly due to the battery level sensing time increase.

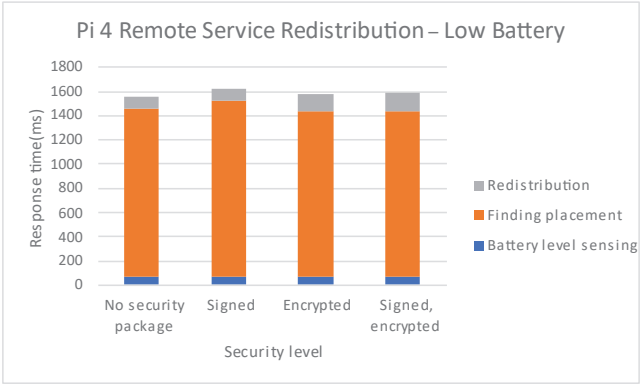


Figure 44. Security package impact on communication in low-resource nodes

When a low-resource fog node is used, such as Raspberry Pi 4, the response time increases almost by 10 times, as it observed in Figure 44 above. It increases mainly by the impact of the placement finding algorithm which is mostly resource-prone, as it was witnessed in report [16]. The usage of a security package does not seem to have any significant impact, apparently, because of no communication between the agents involved. The response time may increase slightly in redistribution and battery level sensing in their turn as these processes involve a sensed state communication to a decision maker and a communication of a decision made by a decision maker to execution agents, and, finally, synchronization agents.

In order to increase the security level, the security parameters can be adjusted instead of using the default ones. It can be achieved by launching addition services. The following experiment is meant to test the impact of the key size on the response time. The same scenario of remote service redistribution due to a low battery level is used. A Raspberry Pi 4 and a PC were used as 2 different fog nodes. 3 different key length configurations were used with each fog node. The service jade.security.AsymKeySize has to be launched, and the key length options are 512, 1024, and 2048.

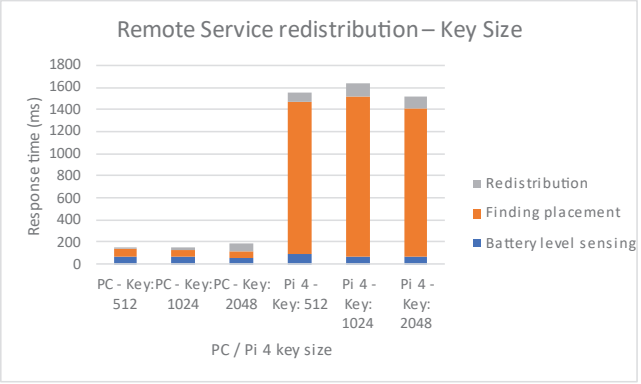


Figure 45. Response time dependency on asymmetric key size

As evidently shown in Figure 45, the size for the public and private keys did not exert any significant impact on the agent communication performance. There might be some slight increase in time due to the procedures which involve more communication of the agents as in redistribution and the battery level sensing, but the biggest impact is made by the placement finding method.

Messages can be intercepted and read if they are exchanged as plain text. Therefore, additional measures to encrypt them are beneficial. The following experiment is meant to test the impact of a message encryption algorithm on the response time. The same scenario of remote service redistribution due to a low battery level is used with a Raspberry Pi 4 and a PC as 2 different fog nodes. 4 different symmetric algorithm configurations were used for messages with each fog node. The service jade.security.SymAlgorithm has to be launched, and the selected options were AES, Blowfish, DES, and TripleDES. More details are available in Figure 46.

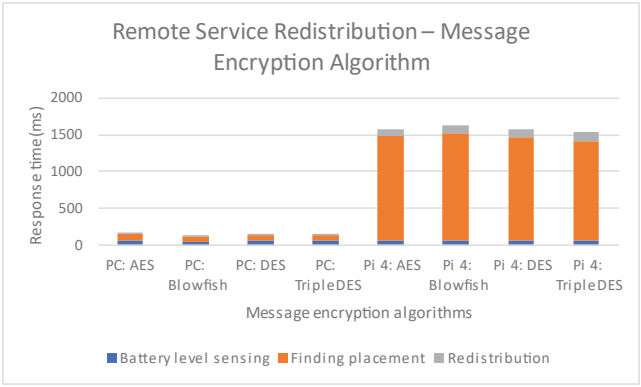


Figure 46. Response time dependency on message encryption

As visualized in the chart above in Figure 46, the message encryption algorithm did not have any significant impact. One of the potential reasons might be the messages themselves which are being communicated since they are very short. Short messages might be resource non-demanding. Long messages would give a better idea, but these involved in the presently conducted experiment are simple agent communication commands, and no increase in their length is needed in practice.

The following experiment is meant to test the impact of a key pair algorithm on the response time. Two different asymmetric algorithm configurations were used to generate key pairs with each fog node. The service jade.security.AsymAlgorithm has to be launched, and the options to choose from are RSA, which is the default one, and DSA. More details are available in Figure 47 below.

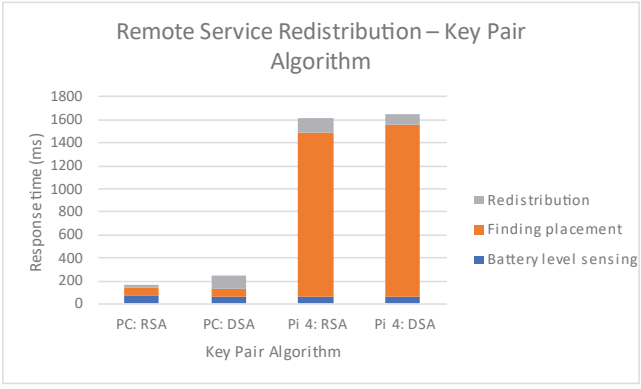


Figure 47. Response time dependency on asymmetric key pair algorithm

RSA and DSA do not seem to have any significant advantage over each other. It allows to conclude that the usage of a security add-on does not contribute to a higher response time, as it is suggested in the Jade security guide. Moreover, the usage of other security configurations rather than the default ones does not obviously contribute to a higher response time either.

4.18. Energy Consumption Evaluation Results

As it was concluded in source [43], it is of importance to maximize the overall runtime of the system. Mobile fog nodes are known for scarce power supply, and therefore resources have to be used in the optimal way. The best service distribution in the perspective of power is the one when nodes are evenly loaded to keep the whole system available as long as possible.

The following experiment is meant to measure the power in Watts, which is needed to get a service placement solution. For a method to be physically available, it has to be running in a fog node. A Raspberry Pi 4 was used as a host for the JADE platform. Measurements were made solely with a running Pi 4, and with launched JADE working on its tasks in the Pi 4. The power needs were compared with a regular 40 W bulb for illustrative purposes. An official power supply adapter with the output of 5.1 V and 3.0 A was used. A USB tester UNI-T UT658B was used to measure the voltage and the amperage which served as a power input for Pi 4; see Figure 48 for more details.

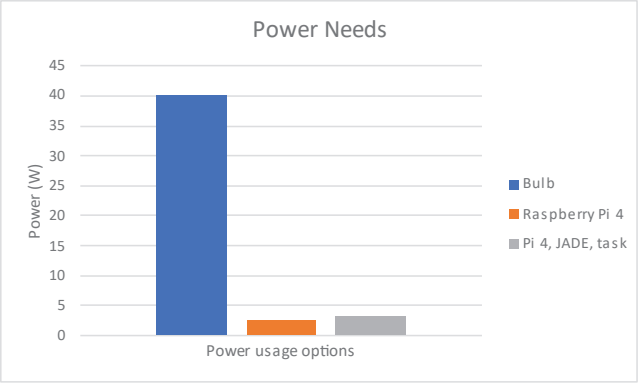


Figure 48. Service placement power consumption

A remote service redistribution scenario was used. 5.1 V and 0.5 A were needed to keep the Raspberry Pi 4 running. It is equal to 2.55 W of power. If JADE was launched and no optimization processes were running, the power consumption settled to the same 2.55 W after a few seconds. However, once an orchestration method starts, the voltage stays more or less the same, but the current increases to 0.62 A, which is equal to 3.162 W. More details are given in Table 12.

Table 12. Energy consumption comparison

Options	Energy (Wh)	Price (cnt/h)	Price (cnt/30 days)
Bulb	40	0.856	616.32
Raspberry Pi 4	2.55	0.055	39.29
Pi 4, JADE, placement	3.162	0.068	48.72

That is a really tiny amount of energy which is needed, and the costs to maintain such a fog node are minimal as well. Even if the orchestration method is running without any interruptions for the whole month, which is barely needed in real-life conditions, a self-cost will be just around 0.1 EUR, considering an estimate that the price for electricity is 0.214 EUR/kWh. The power demands did not vary, however, on the type of tasks which were running such as the battery level sensing, service placement finding or redistribution. It suggests that only the active time of the optimization should be considered as the whole instead of breaking it into different processes.

The following experiment was conducted to analyze a service placement solution energy consumption based on task scalability. The orchestration method was run 20 times with each parameter set by using a Raspberry Pi 4 as a fog node. The Fog computing system complexity was gradually increased by increasing the number of the fog nodes, services, particles, and epochs. An average value was used for each of the 20 attempts.

A rated Raspberry Pi 4 power supply adapter has the values of 5.1 V and 3.0 A. It means that it has enough power for the Raspberry Pi 4 itself, its operations and its peripherals which might be connected to its pins within the power range. However, in order to simulate a mobile node, an external power bank was connected with the USB tester UNI-T UT658B. DC power was calculated by using the formula P = V * I. The voltage and the current of the Pi 4 after it launched its OS were, respectively, equal to 5.1 V and 0.5 A. However, its current increased to 0.62 A when the JADE framework used our optimization method for a placement calculation, and therefore its power needs increased to 3.162 W. The energy required for the optimization was calculated considering the response time and the power which is needed to complete the optimization process as per the formula E = P * t. See Figure 49 and Table 13 for more details.

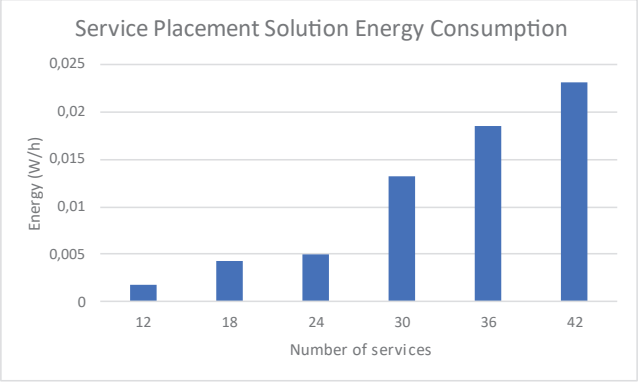


Figure 49. Scalability impact on consumption

Table 13. Scalability parameter sets

Nodes	Services	Particles	Epochs	Response (s)	Energy (W/h)
4	12	200	300	2.032	0.00178
6	18	300	300	4.853	0.00426
8	24	400	400	5.72	0.00502
10	30	500	600	15.082	0.01325
12	36	600	700	21.118	0.01855
14	42	600	800	26.305	0.02310

As visualized in Table 13, finding a service placement for 14 nodes and 42 services requires a very small amount of consumed energy, which is around just 23 mW/h. It is barely considerable, but what requires more attention is the time itself. The response time of over 20 seconds is unacceptable in field conditions. However, the calculation response time can be improved by using a more powerful fog node, such as a PC. With the current complexity configuration and a low-resource fog node, it can still be beneficial to use it with up to 6 nodes and 18 services with a response time of a few seconds. A bigger problem to address in this case would be keeping a mobile fog node running on a battery for a while when its power supply is limited.

4.19. Conclusions

1. A service dynamic orchestration method has been implemented as part of a prototype on a JADE platform. The interaction of the prototype with its environment is described, including resource monitoring, starting new services, data synchronization, security maintenance, and the architecture. Sequence diagrams illustrate the interaction among the MAS agents.
2. Different experiments were conducted to test the method effectiveness in different conditions. These experiments can be classified as parameter tests, stress tests, security tests, power consumption tests, and scalability tests.
3. Our distributed service dynamic orchestrator has a low computational output delay which is caused by the communication and synchronization of JADE agents acting as services. As soon as the placement optimization process is completed and a solution is made available by using IMOPSO and AHP, fog nodes can quickly launch new services or redeploy the available services. It takes just up to 50 ms for a Raspberry Pi 4 fog node which is low on resources to eliminate its three services and start three new services while communicating and synchronizing that solution among its involved agents.
4. The highest computational overhead, however, is generated by a two-step optimization algorithm. A total delay for PCs can reach up to 50%, and it can reach 70% and more on less powerful devices, like the Raspberry Pi 4. It can be even higher if the number of deployable services and fog nodes increases.

5. The inertia weight, according to source [129], demonstrates a clear balanced relationship between exploration and exploitation. It is possible to avoid premature convergence by choosing the right inertia weight *w* [130]. The inertia weight of 0.6 showed the best results, since it leads, on average, to the lowest number of failures in finding the global optimum. Meanwhile, the number of criteria is used for a judgement matrix to highlight the importance of certain qualities, such as security, CPU, RAM, and power. Four criteria were used for all other experiments but, as it is visible from the results of the current study, even 30 criteria with 435 comparisons have a barely visible impact on a high-resource fog node performance. As for a low-resource fog node and a 30-criterion matrix, service placement can be found in less than 1 s, which is still usable in field conditions.
6. It is possible to conclude, based on the CPU stress test, that at least 20% of the CPU must be available for calculations. Otherwise, the response time increases by two or three times. It is not that relevant if the number of particles is low, suggesting that the calculations are relatively simple, but it becomes critical in more complex cases. A CPU stress test has the biggest impact on the service placement phase in contrast to other ones that are relatively simple. When the duration increases from 1.1 s to 3.2 s, it may have a considerable negative effect on tasks that require a short response time.
7. The security add-on JADE-S did not add any clearly observed delay. Service placement decision making takes most of the time, but it does not require any agent intercommunication that is signed, encrypted, or both. Other processes are less complicated and less time-consuming. Some spontaneous time increases or decreases were noted, but they were irregular and did not seem to play a significant role.
8. A USB tester and an energy meter were used to measure the voltage and the current. The power or energy were measured or calculated as required. A self-cost of the JADE platform calculations for 1 month would make only around 0.1 EUR, and the availability of a Raspberry Pi 4 as hardware would add another 0.4 EUR. However, mobility requires the use of batteries, and even such modest energy needs can be an issue when the capacity of the batteries is low enough.
9. Apart from resilience, service redistribution enables user mobility, and constant resource monitoring and fog node synchronization allows fog nodes to adapt to the constantly varying computational resource availability. Such a distributed fog node system can be used for low intensity IoT networks with mobile users and simple constraints, such as *Body Array Networks* (BAN) and *Personal Area Networks* (PAN) or a smart home monitoring application. If

an application involves a larger number of fog nodes where shorter delays are a must, a faster service placement algorithm is needed.

5. GENERAL RESEARCH CONCLUSIONS

1. In order to identify Fog Computing challenges, over 150 papers were checked for more details, and 20 papers were picked as the ones meeting the criteria for further analysis. Challenges were classified to highlight the most common ones. As the obtained results allowed to conclude, security/privacy, dynamics/mobility, and scalability/complexity issues were mostly discussed.
2. Since there is a continuously changing situation in the Fog Computing environment, it requires Fog Service Dynamic Orchestration for problem optimization. The best possible service distribution must be ensured while taking into consideration different constraints. Such constraints may include different security requirements. Different devices in the network may collect different data, which requires a different level of protection while transferring, storing and processing the data. There might be some node physical constraints. A node may be unable to use some security or communication protocols due to processing limitations. The communication bandwidth and communication protocol requirements might also be a concern. There are nodes which use continuous data streaming or a single one-off data transmission in a while, which makes these capabilities and requirements significantly incompatible. There might be one-way communication or two-way communication with acknowledgements. There might be client-to-server or peer-to-peer communication, and all these points require consideration.
3. A new optimization process ensures the optimal service distribution based on its demonstration in Matlab as a part of its simulation. This two-step process is made of Multi-Objective Particle Swarm Optimization (IMOPSO) to generate particle sets as the first step and the Analytical Hierarchy Process (AHP) with a specific judgement matrix as the second step for service distribution priority decisions. It considers different heterogeneous criteria with distinct qualitative and quantitative measurement units.
4. Our two-stage optimization process seamlessly works with different IMOPSO objective functions. One can easily add some new objective functions for a consideration of different criteria. These functions can also be used dynamically. In such a case, the capabilities of software and hardware in a particular node have an impact both on the optimization process values and the calculation algorithm.
5. Our Fog Computing Service Dynamic Orchestrator prototype interacts with its infrastructure as a Multi-Agent System (MAS). The orchestrator is available in all the fog nodes where it makes optimized service placement decisions after considering resources such as the CPU, memory, battery, and security. This orchestration method does not focus on a central control unit. It allows to make a service placement decision for any orchestrator which synchronizes its resource availabilities with other nodes within the fog network. It greatly improves scalability and resilience to fog node failures. It also supports mobility and adapts to resource availability changes.
6. It is possible to conclude, based on a variety of experiments which were conducted, that the optimization method has a linear dependency on the

infrastructure complexity and scalability. An optimization process as a part of the orchestration method requires most of the time, whereas other processes are less resource dependent. It may be required to optimize this optimization process further, or use it with a limited infrastructure.

6. SANTRAUKA

6.1. Darbo aktualumas

Daiktų internetas (IoT) yra sudarytas iš daugybės prietaisų, galinčių reaguoti į aplinką ir aktyvinti. Šie įrenginiai renka, apdoroja ir perduoda duomenis į kitus įrenginius, programas ir platformas. Nord ir kt. teigia [1], kad iki 2020 m. galėjo būti naudojama apie 30 mlrd. įrenginių, bet tikimasi, kad 2025 m. jų skaičius išaugs iki 75 mlrd. Tokia precedento neturinti pažanga skatina mokslinius tyrimus, technologijas ir kompiuterinės įrangos plėtrą. Visgi daiktų internetas (IoT) susideda iš skirtingų paradigmų. Ir viena iš šių paradigmų, kuriai įtakos turėjo daiktų internetas, yra ūko kompiuterija [2], kaip nurodo Hosseinioun ir kt.

Ūko kompiuterija teikia savo skaičiavimo išteklius kaip paslaugą [3], kaip ir debesijos atveju, bet, priešingai nei debesyje, tokia paslauga, kaip skaičiavimo ištekliai, yra geografiškai paskirstyta. Taip pat ūko mazgai yra autonomiški ir decentralizuoti [4]. Heterogeniškumas būdingas dėl duomenų apdorojimo ir saugojimo išteklių pobūdžio. Šie ištekliai dažnai yra riboti dėl mažesnių aparatinės įrangos techninių galimybių. Galiausiai, turi būti ūko mazgo sąveikos koordinacija, arba orkestravimas, tarp ūko mazgų ir debesies, atsižvelgiant į mobilumą.

Galima rasti daug mokslinių straipsnių, sprendžiančių minėtas ūko kompiuterijos paslaugų ir programų paskirstymo problemas. Kaip apžvalgos straipsnyje pabrėžia Costa ir kt. [8], centralizuota orkestravimo topologija yra populiariausia jų analizuotuose straipsniuose, bet tik trijuose iš jų atkreiptas dėmesys į bendrojo trikties taško (angl. *Single Point of Failure*) problemą. Decentralizuotos, arba paskirstytos, orkestravimo topologijos įgyvendinimas padidina sudėtingumą dėl duomenų sinchronizavimo poreikio, kuris priklauso nuo ryšio patikimumo ir didesnių apdorojimo pajėgumų.

Yra ir kitų tyrėjų, tokių kaip Saad ir kt. [9], Toczé ir kt. [10], Guerrero ir kt. [11], kurie pasiūlo paskirstytos paslaugų orkestravimo topologijos sprendimus, bet jie tiesiogiai neatsižvelgia į atsparumo ir ūko mazgų sutrikimus dinaminiuose tinkluose. O paskirstytai ir dinamiškai aplinkai yra vos keli orkestravimo metodai [12]. Dar mažiau šiuose paskirstytos topologijos sprendimuose atsižvelgiama į pačių galutinių įrenginių arba ūko mazgų mobilumą, kaip pastebi Wang ir kt. [13].

6.2. Disertacijos objektas

Šio darbo objektas yra dinaminis orkestravimo metodas, kuris naudojamas ūko kompiuterijos paslaugoms optimizuoti. Subjektai yra kriterijai, pagal kuriuos galimoms alternatyvoms yra taikomas prioritetas.

6.3. Disertacijos tikslas

Šio darbo tikslas yra sukurti ūko kompiuterijos paslaugų dinaminį orkestravimo metodą, leidžiantį optimizuoti ūko kompiuterijos išteklių naudojimą, perkeliant paslaugas heterogeniškoje ir dinamiškoje aplinkoje, atsižvelgiant į programos kriterijus (CPU, atmintį, energiją, pralaidumą ir t. t.) ir ūko mazgo apribojimus.

6.4. Disertacijos uždaviniai

Pagrindinės šios disertacijos užduotys yra tokios:

1. Ištirti ūko kompiuteriją kaip daiktų interneto (IoT) paradigmą, galimus ūko kompiuterijos orkestravimo metodus ir su jais susijusias problemas.
2. Pasiūlyti ūko kompiuterijos paslaugų dinaminį orkestravimo metodą, leidžiantį optimizuoti ūko kompiuterijos išteklių naudojimą.
3. Atlikti siūlomo ūko kompiuterijos paslaugų dinaminio orkestravimo metodo modeliavimą, naudojant atitinkamą programinę įrangą kaip koncepcijos įrodymą.
4. Realizuoti ūko kompiuterijos paslaugų dinaminio orkestravimo metodą, sukuriant prototipą.
5. Atlikti matavimus ir pateikti duomenis grafiškai, tiek atliekant modeliavimą, tiek naudojant prototipą, vykdant eksperimentus, kad būtų galima įvertinti orkestravimo metodo našumą.

6.5. Mokslinis naujumas

Šioje disertacijoje laikomasi šių teiginių:

1. Buvo pasiūlytas dviejų etapų dinaminis daugiatikslio optimizavimo metodas, kuriam naudojamas sveikųjų skaičių daugiatikslio dalelių spiečiaus optimizavimo (angl. *Integer Multi-Objective Particle Swarm Optimization*) (IMOPSO) algoritmas, kad surastų visas nedominuojamas paslaugų alternatyvas, o vėliau geriausia alternatyva surandama naudojant analitinio hierarchijos proceso (angl. *Analytical Hierarchy Process*) (AHP) algoritmą, kuris supaprastina kriterijų palyginimo procesą.
2. Sukurtas ūko kompiuterijos paslaugų orkestratoriaus prototipas, naudojant daugiaagentę sistemą (MAS) su sustiprintu saugumu, atsparumu ir išteklių stebėjimo galimybėmis. Daugiaagentės sistemos yra pažeidžiamos tinklo atakų, taip pat joms trūksta integruotų stebėjimo įrankių. Saugumo problemos sprendžiamos integravus JADE-S priedą, kuris leidžia pasirašyti ir šifruoti agentų komunikacijos pranešimus. Ištekliams stebėti sukurtos stebėjimo klasės.
3. Siūlomas metodas suteikia galimybę nuolat stebėti bet kokias paslaugų užklausas ir turimus išteklius, kad būtų galima priimti realaus laiko paslaugų perkėlimo sprendimus kaip su paskirstytu orkestratoriumi, skirtingai nuo centralizuotų valdymo metodų, kurie yra labiausiai paplitę.

6.6. Praktinė vertė

Ūko kompiuterija yra santykinai nauja paradigma, o ji naudojama kaip debesų kompiuterijos alternatyva, skirta ištekliams teikti arčiau duomenų šaltinių ir jų naudotojų, taip pat įrenginiai gali identifikuoti savo vietą ir pasižymi minimalia delsa [14]. Ūko kompiuterijos ištekliai teikiami kaip paslauga dinamiškoje aplinkoje, atsižvelgiant į paskirties reikalavimus ir ūko mazgo apribojimus. Kaip nurodyta disertacijos motyvacijoje, pastebima per mažai pastangų sukurti paskirstytą paslaugų dinaminį orkestratorių, atsižvelgiant į SPOF, mazgų mobilumą ir atsparumą:

- Pasiūlytas ūko kompiuterijos paslaugų dinaminio orkestravimo metodas, pagrįstas dviejų etapų optimizavimo procesu, naudojant sveikųjų skaičių daugiatikslio dalelių spiečiaus optimizavimo algoritmą (IMOPSO) ir analitinio hierarchijos proceso (AHP) algoritmą, kaip „Matlab“ sprendimas, kuris naudojamas modeliavimui. Atlikti optimizavimo skaičiavimai papildomi grafikais.
- Sukurtas ūko kompiuterijos paslaugų dinaminio orkestravimo prototipas, kuris gali būti naudojamas fiziniam vertinimui. Prototipui sąveikaujant su aplinka, galima stebėti, šifruoti ir pasirašyti komunikacinius pranešimus, taip pat galima stebėti pasiekiamus ūko mazgus ir jų išteklius.
- Galima naudoti šį dinaminio orkestravimo metodą išmaniajam pastatui, pavyzdžiui, studentų miesteliui, kurio tikslas yra optimizuoti energijos sąnaudas. Jį gali sudaryti auditorijos, laboratorijos, sporto salė ir automobilių stovėjimo aikštelė su keliomis elektrinių automobilių įkrovimo stotelėmis. Aplinką stebės ir valdys jutikliai ir aktyvikliai, pavyzdžiui, patalpos temperatūrą stebės jutiklis, o termostatas ją pritaikys piko arba ne piko valandoms. Galima sustabdyti automobilių įkrovimą per piko valandas, kad nesusidarytų elektros energijos trūkumas. Ūko mazgų mobilumo principas leidžia sekti naudotojus. Taip pat sustiprinamas saugumas, kad būtų išvengta įsilaužimų.

Šis doktorantūros tyrimas siejasi su projektu SPARTA, kuris yra ES programos H2020-SU-ICT-2018-2020 dalis.

6.7. Disertacijos teiginiai

Šioje disertacijoje laikomasi šių teiginių:

1. Paskirstytas dinaminis ūko paslaugų orkestravimas pagerina ūko kompiuterijoje atsparumą ir optimizuoja resursus.
2. Dviejų etapų optimizavimo metodas yra daugiatikslio optimizavimo uždavimo sprendimas, leidžiantis optimizuoti visus tikslus tuo pačiu metu, kai šie tikslai yra prieštaringi. Sveikųjų skaičių daugiatikslio dalelių spiečiaus optimizavimo algoritmas (IMOPSO) sudaro sprendimų rinkinį, o analitinio

hierarchijos proceso algoritmas (AHP) parenka geriausią sprendimą iš Pareto optimalių dalelių rinkinio.
3. Paskirstytasis MAS orkestratorius gali išspręsti ūko kompiuterijos problemas, susijusias su tarpusavio sąveika, našumu, paslaugų užtikrinimu, gyvavimo ciklo valdymu, masto keitimu, saugumu ir atsparumu.

6.8. Mokslinis pagrindimas

Visi rezultatai, kurie pateikiami šiame darbe, yra autoriniai. Jie pateikiami kaip publikacijos šiuose mokslo žurnaluose su „Web of Science“ indeksu:

1. „Method for Dynamic Service Orchestration in Fog Computing“, „Electronics“ 2021 m., https://doi.org/10.3390/electronics10151796
2. Šatkauskas, N.; Venčkauskas, A. Multi-Agent Dynamic Fog Service Placement Approach. Future Internet 2024, 16, 248, https://www.mdpi.com/1999-5903/16/7/248.

Eksperimentiniai rezultatai buvo pristatyti ir aptarti šiose konferencijose:

1. „Orchestration Security Challenges in the Fog Computing“, „26th International Conference on Information and Software Technologies“ (ICIST 2020), Kaunas, Lietuva.
2. „Fog Computing Service Placement Orchestrator“, „11th International Conference on Electrical and Electronics Engineering“ (ICEEE 2024), Marmaris, Turkija.

6.9. Disertacijos struktūra

Disertaciją sudaro 5 skyriai:

1 skyrius – įvadas, kuriame aptariamas mokslinis naujumas, tikslai, uždaviniai ir kiti šios disertacijos aspektai. Jame taip pat trumpai aprašoma ūko kompiuterija, jos pagrindinės problemos ir disertacijos motyvacija.

2 skyriuje pateikiama išsami ūko kompiuterijos kaip paradigmos apžvalga. Taip pat aprašomos kitos daiktų interneto (IoT) paradigmos, plėtojimo problemos, metodai, koncepcijos, karkasai ir technologijos. Galiausiai įvardijamos ūko kompiuterijos orkestravimo problemos.

3 skyriuje pristatomas ūko kompiuterijos paslaugų dinaminis orkestravimo metodas. Aptariamas tiek sveikųjų skaičių daugiatikslio dalelių spiečiaus optimizavimo (IMOPSO) algoritmas, tiek analitinio hierarchijos proceso (AHP) algoritmas, nes šį metodą sudaro du algoritmai. Modeliavimas atliktas su „Matlab“.

4 skyriuje pristatomas prototipas, kuris JADE platformoje realizuotas kaip daugiaagentė sistema (MAS). Ūko mazgams naudojami „Raspberry Pi 4“ įrenginiai.

Dinaminiai orkestratoriai, kuriems naudojamas IMOPSO ir AHP algoritmas, įdiegti į visus ūko mazgus ir sudaro paskirstytą topologiją.

5 skyriuje pateikiamos bendrosios tyrimo išvados.

7. ŪKO KOMPIUTERIJOS PASLAUGŲ DINAMINIO ORKESTRAVIMO APŽVALGA

Šiame skyriuje pateikiama pati ūko kompiuterijos koncepcija ir jos problemos. Bendrai įvardijamos įvairios ūko kompiuterijos paradigmos, ūko kompiuterijos orkestravimas ir saugaus orkestravimo iššūkiai.

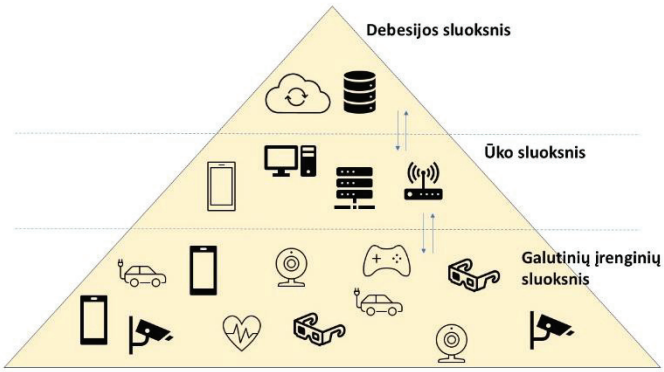
7.1. Ūko kompiuterijos paradigma

Ūko kompiuterija yra gana nauja paradigma, todėl, kaip Yousefpour ir kt. nurodo tyrime [18], „...ūko kompiuterijos ir ūko mazgų apibrėžimas yra nuolatinių diskusijų tema“. Visgi „OpenFog Consortium“ konsorciumas ėmėsi novatoriško vaidmens kurdamas šiuos apibrėžimus ir standartus. „OpenFog Consortium®“ konsorciumą 2015 m. įkūrė tokios organizacijos, kaip ARM, „Cisco“, „Dell“, „Intel“, „Microsoft“ ir Prinstono universitetas. „OpenFog“ rekomendacinės architektūros („OpenFog RA“) tikslas – padėti atitinkamoms įmonėms, programinės ir aparatinės įrangos kūrėjams kurti ir prižiūrėti bet kokius su ūko kompiuterija susijusius elementus. Ūko kompiuterija pagal apibrėžimą yra

Horizontali sistemos lygio architektūra, kuri paskirsto skaičiavimo, saugojimo, valdymo ir tinklo funkcijas arčiau naudotojų, vadovaujantis debesies-daiktų tęstinumu.

Ūko kompiuterija pagal nurodytą dokumentą yra tradicinio kompiuterijos modelio, kuris yra pagrįstas debesija, papildomas elementas. Bet kokia realizuota architektūra gali būti skirtinguose tinklo topologijos sluoksniuose. Ūko kompiuteriją papildo debesija, todėl yra prieinami visi tokie debesijos privalumai, kaip konteineriai, virtualizacija, orkestravimas ir kt. Rekomendacinės architektūros dokumente yra išsamiai aptariami architektūros ramsčiai, kuriuos sudaro „…saugumas, masto pritaikymas, atvirumas, autonomija, RAS (patikimumas, prieinamumas ir prižiūrimumas), judrumas, hierarchija ir programuojamumas“.

Nurodytame dokumente ūko kompiuterija prilyginama daiktų internetui (IoT). Jame ji atlieka tam tikrus specifinius vaidmenis. Bonomi ir kt. [21] siekia įrodyti, kad ūko kompiuterija dėl savo charakteristikų yra tinkama platforma įvairioms IoT paslaugoms ir programoms. Ji taip pat išplečia debesų kompiuterijos ribas iki tinklo krašto. Kaip nurodė „OpenFog Consortium“ [20], šis gebėjimas išplėsti ją iki tinklo krašto kartais gali kelti nesusipratimų, , nes ūko kompiuterija yra labai panaši į „periferinę kompiuteriją“ (angl. *Edge Computing*). Visgi yra keletas skirtumų. Ūkui galima naudoti debesiją, bet jos negalima prijungti prie periferinės kompiuterijos. Ūkui būdinga hierarchija, bet periferinę kompiuteriją sudaro tam tikras skaičius sluoksnių. Periferine kompiuterija taip pat siekiama pagerinti tinklų funkcionalumą, duomenų išsaugojimą, valdymą ir greitaveiką.



50 pav. Daugiasluoksnės ūko kompiuterijos struktūra

Kaip matyti iš Yousefpour ir kt. tyrimo [18], ūko kompiuterija yra daugiasluoksnė. Iš esmės ji turi 3 sluoksnius: A) debesį; b) ūką; c) IoT (žr. 50 pav.). Apatinis (IoT) sluoksnis apimtų visus galinius mazgus, pavyzdžiui, jutiklius, aktyviklius, transporto priemones arba net išmaniuosius įrenginius, jei jie naudojami duomenims rinkti, o ne jiems apdoroti. Antrasis sluoksnis, kuris, pasak autorių, yra ūko, sudarytas iš skirtingų prieigos taškų, ugniasienių, maršruto parinktuvų, komutatorių, tinklo sietuvų ir kt. O viršutinis sluoksnis priklauso debesijai, kurią sudaro didelio masto duomenų centrai.

7.2. Ūko kompiuterijos orkestravimo dinamika

Ūko kompiuterijos išteklių ir paslaugų prieinamumas dinamiškai keičiasi laiko atžvilgiu [34]. Paslaugos paprastai yra mobilios, paskirstytos, ir jos akimirksniu atsiranda ir išnyksta. Norint užtikrinti tinkamą paslaugų kokybę (QoS), reikalingas visapusiškas stebėsenos metodas. Stebėjimas yra esminė tinkamo ūko paslaugų orkestravimo funkcija [35]. Ji renka būsenos informaciją apie ūko mazgus ir komunikacijos kanalus. Kai orkestratorius gauna atnaujintus duomenis apie infrastruktūrą ir paslaugas, jis gali priimti tinkamus sprendimus, teikti paslaugas mazge su atitinkamais ištekliais ir optimizuoti paslaugų vietą. Visgi reikia balansuoti tarp informacijos atnaujinimo dažnio ir mazgų delsos. Paslaugų pasiekiamumą galima nustatyti stebint tam tikras metrikas, pavyzdžiui, paslaugų atsako laiką. O paslaugų sutrikimo priežastis galima nustatyti išanalizavus žurnalus.

Nesvarbu, ar tai yra statinė, ar dinaminė konfigūracija, tikimasi, kad stebėjimo sistema išmatuos reikiamus metrikos rodiklius, kurie yra svarbūs kontrolės procesams, juos saugos vietoje ir prireikus laiku perduos. Tokia metrika gali apimti saugumo lygio vertinimą, CPU užimtumą, RAM užimtumą, energijos naudojimą, atstumą [36] ir kt. Galima stebėti nuolat, kai orkestratorius gauna nuolatinį duomenų srautą, periodiškai, kai rodmenys siunčiami pasikartojančiu ciklu, ir pagal įvykius, pavyzdžiui, kai įvyksta aktyvinimo įvykis, gaunami ribiniai rodmenys arba prašymas [5].

7.3. Paskirstytas paslaugų orkestravimas

Kaip nurodoma tyrimuose, galima naudoti (a) centralizuotą, (b) decentralizuotą ir (c) paskirstytą orkestravimo topologiją. Centralizuotas orkestravimas yra labiau pažeidžiamas dėl SPOF (kritinės rizikos vietos). Decentralizuotai orkestravimui reikia specialaus algoritmo, kad būtų galima pasirinkti FOA (ūko orkestravimo agentą). O paskirstytasis orkestravimas yra mažiau populiarus tyrimuose, nes juo yra sudėtingiau valdyti įrenginius suderintai. Kaip apibendrinta apžvalgos tyrime [40], paskirstyto ūko tinklo kompiuterijos sistema yra būtina, siekiant sumažinti energijos sunaudojimą ir sutrumpinti galutinių įrenginių atsako laiką. Tai taip pat sumažina apkrovą debesyje ir leidžia realiuoju laiku lokaliai teikti paslaugą, skirtą duomenims apdoroti.

Lyginant du skirtingus metodus, [41] pavyzdžiui, paskirstytą ir centralizuotą, siekiant optimalaus išteklių paskirstymo tarp mazgų ir siekiant sumažinti bendrą užduočių apdorojimo vėlavimą ir energijos suvartojimą, buvo nuspręsta, kad paskirstytoje architektūroje paslaugos geriau perkeliamos, kai pirmenybė teikiama delsai. Centralizuoto metodo atveju sprendimus priima centrinis įrenginys, kuris, kaip tikimasi, žinos kiekvieno mazgo būseną, bet situacija yra tokia, kad ūko mazgas priima nepriklausomus sprendimus dėl bet kokios paslaugos, teikiamos paskirstytoje topologijoje.

Vienas iš reikalavimų, keliamų ūko kompiuterijos architektūroms, yra infrastruktūros atsparumas (angl. *resilience*). Firouzi ir kt. [42] naudojama našumo metrika, kurioje infrastruktūros atsparumui skiriamas aukščiausias balas. To priežastis – galimas sutrikimas, dėl kurio orkestratorius turi iš naujo nukreipti savo srautą į kaimyninius ūko mazgus, kad galėtų atnaujinti duomenų apdorojimą. Kaip teigiama tyrime, nors centralizuotas orkestratorius gali optimizuoti paslaugų teikimo procesą kartu su debesimi ir ūko sluoksniu, tarpiniai ūko orkestratoriai gali prisidėti prie didesnio atsparumo. Taip pat galima atsparumą papildomai padidinti leidžiant to paties sluoksnio prietaisams tiesiogiai komunikuoti tarpusavyje. Taip galima optimizuoti paslaugų teikimą tiek kolektyviai, tiek individualiai. Jei optimizuojama individualiai, taikoma buferizavimo strategija. Jei optimizuojama kolektyviai, ūko mazgai komunikuoja tarpusavyje saugiu kanalu, ieškodami netoliese esančių išteklių.

7.4. Išvados

1. Šiame skyriuje apžvelgiama ūko kompiuterija kaip paradigma, dinamiškas ir paskirstytas orkestravimas, jo iššūkiai, paslaugų įkėlimo problemos, siūlomi orkestravimo sprendimai ir identifikuoti ūko orkestravimo saugos iššūkiai. Kadangi ūko kompiuterija yra aplinka, kurioje veikia orkestratorius, orkestratoriaus saugumo iššūkiai priklauso nuo paties ūko kompiuterijos ir prieinamų orkestravimo metodų.
2. Tokie raktiniai žodžiai, kaip „Fog Orchestration“, „Fog Orchestrator“ ir „Fog Orchestrator“, buvo naudojami ieškant mokslinių tyrimų su atitinkama informacija. Iš gautų paieškos rezultatų buvo peržiūrėta apie 150 mokslinių tyrimų, tada 20 jų buvo atrinkti toliau analizuoti , o kiti atmesti dėl to, kad nebuvo reikiamos informacijos. Analizuojant šiuos tyrimus, ieškota nurodytų

orkestravimo iššūkių. Rasti iššūkiai tyrimo suklasifikuoti, siekiant nustatyti aktualiausius. Saugumas / privatumas, dinamika / mobilumas ir masto pritaikomumas / sudėtingumas buvo dažniausi, kuriuos tyrėjai buvo linkę įvardyti.

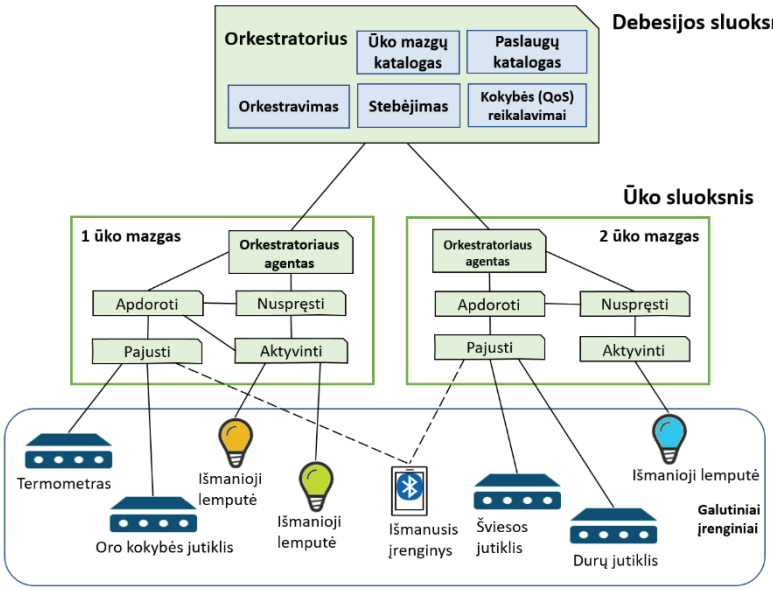
3. Apibendrinti naujausi paslaugų orkestravimo metodai, ieškant sprendžiamų problemų. Neretai pasitaikydavo dinaminių paslaugų orkestravimo metodų, bet tik keletas iš jų veikia kaip paskirstytieji. Dar mažiau jų atsižvelgia į atsparumą. Tai nulėmė tolesnį sprendimą suprojektuoti, susimuliuoti ir realizuoti paslaugų dinaminį orkestravimo metodą, kuris būtų paskirstytas ir orientuotas į atsparumą / saugumą.
4. Kai galima spręsti iš atliktos literatūros analizės, orkestravimo metodas turėtų būti kuriamas atsižvelgiant į tokius kriterijus, kaip saugumas, dinamika ir masto pritaikymas. Šios problemos yra dažniausiai pasitaikančios. Tolesnė metodų apžvalga nulemia paties orkestratoriaus tipą, nes aktualiausias kriterijus būtų paskirstytas, dinamiškas ir atsparus. Visgi į tokią kriterijų kombinaciją kituose metoduose nėra užtektinai atsižvelgiama.

8. ŪKO KOMPIUTERIJOS PASLAUGŲ DINAMINIO ORKESTRAVIMO METODO SIMULIACIJA

Šis skyrius skirtas ūko kompiuterijos paslaugų dinaminio orkestravimo metodui, pagrįstam IMOPSO ir AHP metodais, simuliuoti „Matlab“ programa.

8.1. Orkestratoriaus komponentai ir architektūra

Šiame skyriuje apžvelgiame ūko paslaugų dinaminio orkestratoriaus architektūrą ir komponentus, pateiktus 51 paveikslėlyje. Paslaugų orkestratorius veikia debesijos sluoksnyje, kuris naudojamas optimaliai paskirstyti paslaugas tarp kelių orkestruojamų ūko mazgų. Orkestruojamuose ūko mazguose teikiamos tam tikros paslaugos, kurios palaiko ryšį su galiniais prietaisais, renka ir apdoroja duomenis bei priima lokalius sprendimus dėl galinių prietaisų sluoksnyje esančių valdymo aktyviklių. Specialiosios paslaugos (orkestratoriaus agentai) fiziškai yra kiekviename ūko mazge ir komunikuoja su orkestratoriumi, pateikdamos visą būtiną informaciją, reikalingą priimant sprendimus dėl paslaugų įkėlimo.



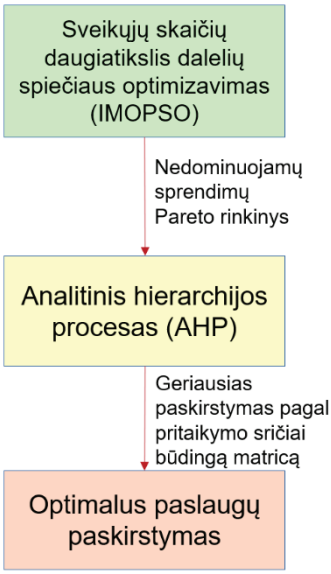
51 pav. Ūko dinaminio orkestratoriaus architektūra ir jos komponentai

Orkestratoriaus agentai lokaliai stebi ūko mazgų aparatinę ir programinę įrangą. Jie žino apie esančią CPU ir RAM apkrovą, energijos poreikį ir energijos lygį, galimus komunikacijos protokolus ir pralaidumą, saugumo galimybes, teikiamų paslaugų būklę ir kt. Jie apibendrina visą surinktą informaciją ir ją pateikia debesų sluoksnyje veikiančiam orkestratoriui. Orkestratorius žino esamą padėtį visuose ūko mazguose, be to, jam yra perduoti saugumo ir QoS reikalavimai, kurie keliami pagal IoT pritaikymo srities sprendimą, ir jis priima sprendimus dėl konkrečių paslaugų paleidimo, sustabdymo arba perkėlimo tarp orkestruojamų ūko mazgų. Orkestratoriaus priimti sprendimai perduodami orkestratoriaus agentams,

veikiantiems ūko mazguose, o orkestratoriaus agentai inicijuoja reikiamus veiksmus dėl paslaugų.

8.2. Dviejų etapų optimizavimo procesas

Mes siūlome naudoti daugiatikslį optimizavimą, kad būtų galima nuspręsti, kuris *n* prieinamų paslaugų paskirstymas *k* ūko mazguose yra geriausias pagal nustatytus apribojimus ir sąlygas [36]. Bendras siūlomų dviejų etapų optimizavimo proceso eigos grafikas pavaizduotas toliau pateiktame 52 paveikslėlyje.



52 pav. Siūlomo dviejų etapų optimizavimo proceso diagrama

Optimizavimo procesas turi du pagrindinius etapus – daugiatikslį optimizavimą ir daugiatikslį sprendimų priėmimą, bet užduotis turi būti išreikšta formaliu matematiniu modeliu, kad ją būtų galima naudoti su bet kokiais formaliais optimizavimo metodais. Tolesniuose poskyriuose išsamiai aprašomas šis optimizavimo procesas.

8.3. Paslaugų paskirstymo tikslo funkcijos

Pagrindinis šio optimizavimo proceso uždavinys yra rasti optimalų *n* paslaugų pasiskirstymą tarp galimų *k* ūko mazgų. Kiekvienas ūko mazgas gali turėti šiek tiek skirtingas charakteristikas, bet laikomasi nuomonės, kad visi mazgai gali paleisti visas paslaugas. Optimizavimo tikslas yra paskirstyti visas paslaugas taip, kad svarbių savybių rinkinys būtų optimalus. Galimos *i* paslaugų paskirstymo savybės X_i išreiškiamos tikslo funkcijų f_j (X_i), $j=1,2,\ldots,m$ ir apribojimo sąlygų reikšmėmis. Optimizavimo proceso objektas – rasti geriausią paslaugų paskirstymą X_{opt}, kuris minimalizuoja visas tikslo funkcijas f_j, dėl ko gauname daugiatikslio optimizavimo užduotį:

$$X_{opt} = \underset{i}{argmin}\, F(X_i); \qquad (1)$$

čia $F(x) = \{f_1(x), f_2(x), \dots, f_m(x)\}$ tikslo funkcijų rinkinys, o $x \in \{X_i\}$ yra rinkinio narys su visais galimais paslaugų paskirstymais.

Ribojimų sąlygos išreiškiamos lygtimis:

$$\begin{cases} g_j(X_i) \geq 0, \ j = 1,2, \dots, n_g \\ h_k(X_i) = 0, \quad k = 1,2, \dots, n_h \end{cases}. \qquad (2)$$

Skirtingi ūko mazgai pasižymi skirtingomis charakteristikomis, tinklo pralaidumu ir saugumo savybėmis. Dėl skirtingo paslaugų paskirstymo tarp ūko mazgų gali susidaryti šiek tiek skirtingas charakteristikas turinti darbo sistema. Pavyzdžiui, jei vienas ūko mazgas yra žemesnio saugumo lygio (dėl ribotų techninės įrangos galimybių), bet šiame mazge yra svarbi paslauga, tada bendras visos sistemos saugumas sumažinamas iki mažiausiai saugaus ūko mazgo saugumo. Naudojame įvairias tikslo funkcijas $f_j(.), \ j = 1,2, \dots, m$ norėdami įvertinti visas tokias situacijas, kurios apima: bendrą sistemos saugumą, procesoriaus apkrovą, RAM apkrovą, energijos sąnaudas, atstumą ir kt. Kai kurios tikslo funkcijos, kurios buvo naudojamos mūsų eksperimentuose, pateikiamos tolesniuose paragrafuose.

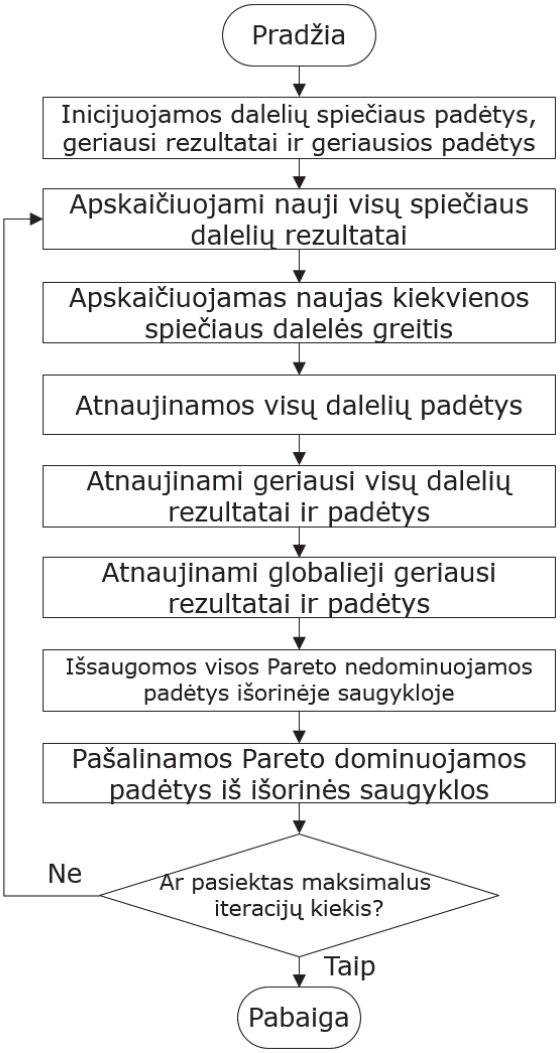
Visos sistemos saugumas naudojant *i* paslaugų paskirstymą $f_{sec}(X_i)$ yra apibrėžiamas žemiausiu visų paslaugų saugumu. Saugos lygiai (išreikšti saugumo bitais, remiantis NIST [100]) ūko mazgams priskiriami pagal jų galimybes veikti su atitinkamais saugumo protokolais. Laikomasi nuomonės, kad paslaugos gali veikti su visais ūko mazgais, tada saugumo kriterijų funkcijos reikšmė $f_{sec}(X_i)$ paslaugų paskirstymui X_i atitinka mažiausią visų ūko mazgų, kuriame yra bent viena paslauga, saugumo lygį. Pavyzdžiui, jei mes turime situaciją, kai trys ūko mazgai gali užtikrinti 128 bitų saugumą, bet vienas ūko mazgas yra apribotas ir jis gali užtikrinti tik 86 bitų saugumą, o jei bent viena paslauga yra įkeliama, tada bendras paslaugų paskirstymo saugumas tampa lygus 86 bitams, kas atitinka tikslo funkcijos reikšmę $f_{sec}(X_i) = 86$. Jei naudojame savo paslaugas taikymo srityje, kuriai reikalingas tam tikras saugumo lygis, toks reikalavimas išreiškiamas kaip apribojimo sąlyga, t. y. jei tam tikrai taikymo sričiai reikia ne mažiau kaip 128 bitų saugumo, tada mes turime atitinkamą apribojimą $g_{sec}(X_i) \geq 128$.

8.4. IMOPSO, skirtas Pareto galimų paslaugų paskirstymų rinkiniui rasti

Pirminis dalelių spiečiaus optimizavimo (PSO) algoritmas geriausiai tinka tęstinėms užduotims optimizuoti, bet [106] [107] yra keletas modifikacijų, leidžiančių jį naudoti sprendžiant diskretines užduotis. Jei yra keli vienas kitam prieštaraujantys tikslai, PSO algoritmas gali būti pritaikytas, kad būtų galima rasti optimalų Pareto sprendimų rinkinį [108] [109]. Mes naudojome Coello ir kt. pasiūlytą daugiatikslį dalelių spiečiaus optimizavimo metodą (MOPSO) [109], norėdami surasti Pareto galimų paslaugų paskirstymų rinkinį tarp ūko mazgų. Norint naudoti šį metodą, reikia jį šiek tiek pritaikyti, kad jis veiktų ribotų sveikųjų skaičių *n* matmenų erdvėje, naudojamai paskirstyti galimas paslaugas, kurios atvaizduojamos kaip spiečiaus dalelės (pradiniame metode naudojama tęstinė realiųjų skaičių erdvė).

Vektoriumi $X_i = (x_1, x_2, \dots, x_n)^T$, $x_j \in \{1,2,\dots,k\}$, $j = 1,2,\dots,n$ užkodavome i paslaugų paskirstymą, kur n yra paslaugų, kurios turi būti paskirstytos tarp k ūko mazgų, skaičius. Vektorinio elemento reikšmė $x_j = l$ yra ta, kad j paslauga turi būti įkelta į l ūko mazgą.

Toliau pateiktame 53 paveikslėlyje parodyta sveikojo skaičiaus daugiatikslio dalelių spiečiaus optimizavimo (IMOPSO) algoritmo eigos diagrama.



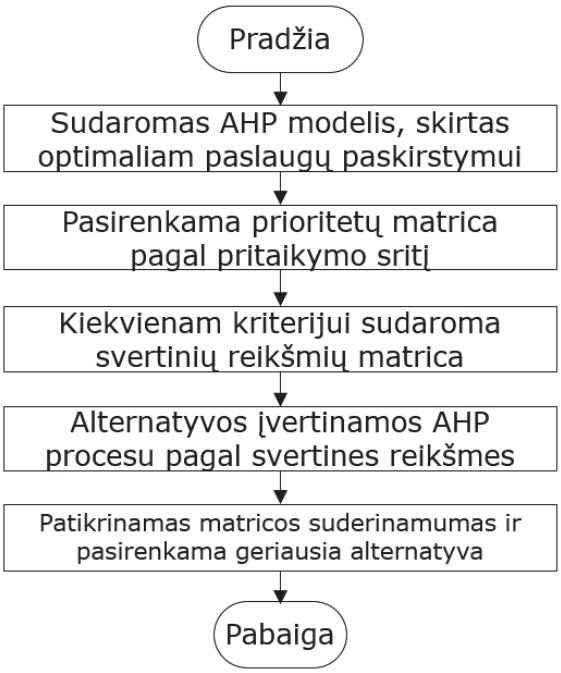
53 pav. Bendrojo IMOPSO algoritmo eigos diagrama

8.5. Optimalaus paslaugų paskirstymo paieška, naudojant AHP

Naudojome analitinės hierarchijos procesą (AHP) [105] [110] siekdami parinkti geriausią sprendimą iš Pareto optimalių reikšmių rinkinio, lygindami poromis visus nedominuojamus paslaugų paskirstymus pagal visas turimas tikslo funkcijas.

AHP paprastai naudojama situacijose, kai reikia priimti sprendimą su nedideliu kiekiu kiekybinių duomenų, naudojant išsamiąją analizę, kurią atlieka kelios sprendimų priėmimo šalys, taikant galimų sprendimų palyginimą poromis. Galima taip pritaikyti AHP, kad jį būtų galima naudoti mašininiam sprendimų priėmimui, per kurį įvertinamos sudėtingos kelių kriterijų užduotys [111] [112] [113]. AHP pasirinkimas vietoj kitų formalių daugiatikslių sprendimų priėmimo algoritmų grindžiamas toliau nurodytomis priežastimis [105]. AHP leidžia automatiškai patikrinti sprendimų priėmėjų pateiktų vertinimų suderinamumą. AHP naudoja normalizuotas kriterijų reikšmes, todėl skirtingiems kriterijams gali būti naudojamos nevienalytės matavimo skalės. Pavyzdžiui, galima naudoti tik kokybinę saugumo skalę (aukštas, žemas, vidutinis) ir tuo pačiu metu naudoti nenuoseklias skaitines skales bet kokiems energijos ir CPU reikalavimams. AHP naudoja tik porinius alternatyvų palyginimus, kurie leidžia supaprastinti daugiatikslį sprendimų priėmimą, siekiant geresnio rezultatų patikimumo.

54 paveikslėlyje pavaizduotas AHP sprendimų priėmimo algoritmas.



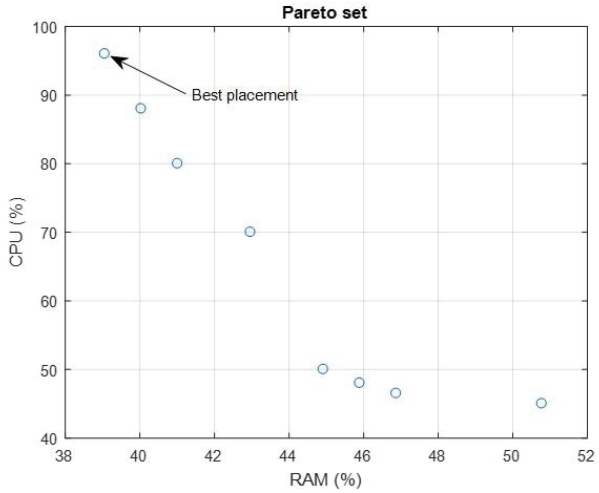
54 pav. Bendrasis AHP sprendimų priėmimo algoritmas

8.6. Vertinimas ir rezultatai

Pritaikytas supaprastintas scenarijus, kuriame naudojamos tik dvi tikslo funkcijos, kad būtų galima pademonstruoti, kaip veikia IMOPSO algoritmas ir kaip atrodo Pareto sprendimų rinkinys. Pareto rinkinys šiuo atveju gali būti atvaizduojamas dvimate diagrama. Prioritetų matrica, naudojama sprendimui, susideda tik iš 4 elementų:

$$J = \begin{pmatrix} 1 & 3 \\ \frac{1}{3} & 1 \end{pmatrix}. \quad (4)$$

Kai naudojamos dvi RAM ir CPU tikslo funkcijos, tada ši prioritetų matrica reiškia, kad tolygus RAM apkrovos pasiskirstymas tarp visų ūko mazgų yra svarbesnis nei procesoriaus apkrovimas. Pareto alternatyvų rinkinys, gautas pagal IMOPSO algoritmą, pateiktas 55 paveikslėlyje. Tada antrajame etape šiam Pareto sprendimų rinkiniui pritaikomas AHP procesas, siekiant rasti geriausią paslaugų išdėstymą. Geriausias išdėstymas yra apibendrintas 1 lentelėje.



55 pav. Pareto supaprastinto scenarijaus rinkinys

14 lentelė. Geriausias paslaugų išdėstymas supaprastintame scenarijuje

	1 ūkas	**2 ūkas**	**3 ūkas**	**4 ūkas**
Paslaugos	3 aktyviklis 4 procesas 5 procesas 6 procesas	1 jutiklis 2 jutiklis 2 aktyviklis 1 procesas	3 jutiklis 2 procesas	1 aktyviklis 3 procesas 7 procesas

Geriausias rezultatas (geriausios tikslo funkcijų reikšmės) šiuo scenarijumi yra (39, 96)^T, o tai reiškia, kad šis paslaugų išdėstymas užtikrina RAM užimtumą maksimaliai iki 39 % per visus keturis ūko mazgus. Maksimalus CPU užimtumas šiuo atveju yra 96 %.

8.7. Išvados

1. Pristatyta dinaminio orkestratoriaus architektūra ir jos komponentai. Aprašytas orkestravimo metodo valdymo ciklas ir dviejų etapų optimizavimo procesas, kuris naudojamas kaip paslaugos orkestravimo metodo dalis.

2. Dviejų etapų metodui naudojamas „Integer Multi-Objective Particle Swarm Optimization“ (IMOPSO) algoritmas, skirtas Pareto optimalų sprendimų rinkiniui rasti, ir Analitinės hierarchijos procesas (AHP) su konkrečiai sričiai pritaikyta matrica, skirta sprendimui dėl optimalių sprendimų atrinkimo priimti. Toks paslaugų paskirstymo optimizavimas leidžia įvertinti skirtingus heterogeninius kriterijus, kurie yra su skirtingais matavimo vienetais ir skirtingu pobūdžiu (kokybiniu arba kiekybiniu). Šis metodas, nepaisant to, kad išrenka geriausią sprendimą, taip pat eilės tvarka išdėsto visus Pareto optimalius sprendimus, leidžiančius juos palyginti tarpusavyje (atsakant į klausimą,kiek vienas sprendimas yra geresnis už kitą) ir, jei reikia, pasirinkti antrąjį geriausią, trečiąjį geriausią ir kt. sprendimą.
3. Modeliavimas atliktas su „Matlab“. Dėl prieinamų bibliotekų buvo paprasčiau atlikti metodo simuliaciją. Gaunant simuliacijos rezultatus, sukuriami grafikai, kurie vaizdžiai papildo duomenis. Ši simuliacija taip pat leido sutaupyti pinigų, išbandant koncepcijos veiksmingumą, nes turint kompiuterį nereikėjo įsigyti jokios papildomos aparatinės įrangos. Pritaikyti trys scenarijai, iliustruojantys skirtingas situacijas. Pirmasis scenarijus buvo supaprastintas, turintis tik dvi tikslo funkcijas, kuriuo siekta patikrinti metodo veiksmingumą. Antrasis scenarijus skirtas optimaliam paslaugų, turinčių skirtingus matricos prioritetus, išdėstymui. O per trečiąjį scenarijų reikėjo perskirstyti įkeliamas paslaugas dėl atstumui keliamų reikalavimų, imituojant ūko mazgo mobilumą.

9. DAUGIAAGENČIO ŪKO KOMPIUTERIJOS PASLAUGŲ DINAMINIO ORKESTRAVIMO METODO PROTOTIPO KŪRIMAS

Šis skyrius skirtas ūko kompiuterijos paslaugų dinaminio orkestravimo metodo, pagrįsto IMOPSO ir AHP algoritmais, fiziniam prototipui realizuoti. Pasiūlytas paslaugų įkėlimo orkestravimas, kuris veikia kaip daugiaagentė sistema. Ūko kompiuterijos paslaugos perteikiamos agentais, kurie gali funkcionuoti tiek savarankiškai, tiek bendradarbiaujant. Paslaugų perkėlimo vieta apskaičiuojama taikant dviejų etapų optimizavimo procesą. Šis paslaugų įkėlimo orkestratorius yra paskirstytas, paslaugos surandamos dinamiškai, ištekliai gali būti stebimi, o komunikacijos pranešimus tarp ūko mazgų galima pasirašyti ir užšifruoti, siekiant išspręsti daugiaagenčių sistemų silpnąsias vietas dėl stebėjimo priemonių ir saugumo trūkumo.

9.1. Dizaino motyvacija

Šio orkestratoriaus projektavimo proceso tikslas yra parodyti, kaip orkestratoriai priima sprendimus, leidžiančius kontroliuoti paslaugas, atsižvelgiant į QoS ir saugumo reikalavimus. Kiekvienas sprendimas, kurį orkestratoriai priima, siekdami paleisti / sustabdyti / perkelti savo paslaugas, turi būti dinamiškai patikrintas, ar minimalūs reikalavimai, susiję su įvairiais aparatinės ir programinės įrangos apribojimais, taip pat reikalavimai dėl taikymo srities ypatumų (pvz., neskelbtini duomenys turėtų būti geriau apsaugoti nei aplinkos stebėjimo duomenys), yra įvykdyti.

Projektuojant orkestratorių, reikėtų atsižvelgti į tris pagrindinius etapus:

- Kiekvienas orkestratorius pirmame etape turėtų įvertinti tokius reikalavimus, kaip saugumas, CPU, RAM ir energija, atsižvelgiant į taikymo sritį ir įvairias ūko mazgo aparatinės / programinės įrangos galimybes, kad nuspręstų, ar galima paleisti visas reikalingas paslaugas nepažeidžiant šių reikalavimų.
- Antrasis etapas – rasti optimalų paskirstymą dislokuojamoms paslaugoms tarp skirtingų ūko mazgų. Gali būti itin svarbu taupyti energiją ir skaičiavimo išteklius tais atvejais, kada kai kurios paslaugos turi būti panaikintos, pristabdytos arba perkeltos į kitus ūko mazgus.
- Trečiąjį etapą sudaro dinamiškas paslaugų perkėlimas, kai aplinkybės pasikeičia ir orkestratoriai turi perskirstyti savo paslaugas prieinamuose ūko mazguose pagal naujas aplinkybes.

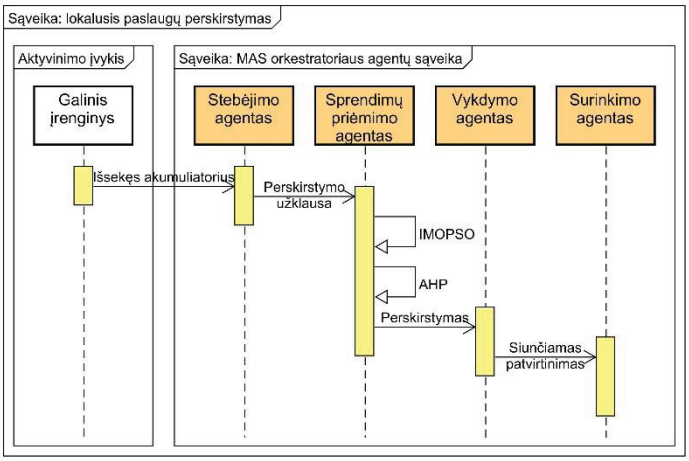
9.2. Išteklių stebėjimas

Išteklių stebėjimas vykdomas naudojant stebėjimui skirtą agentą. Jis laukia pranešimų, naudodamas ciklinį elgesį. Šių pranešimų turinys filtruojamas naudojant metodą „startsWith()“, kad būtų galima atlikti atitinkamą veiksmą. Jis gali gauti pranešimus iš akumuliatoriaus įtampos agento („BattVoltAgent“), šviesos agento („LightingAgent“) arba jutiklio agento („AbstractSensorAgent“). Akumuliatoriaus

įtampos agentas stebi „Raspberry Pi 4“ įrenginio akumuliatoriaus įkrovos kaip ūko mazgo lygį. „Pi 4“ neturi lokalios analoginės-skaitmeninės (ADC) konversijos galimybės. Reikia išorinio MCP3424 18 bitų ADC-4 kanalų keitiklio. Prototipui naudojama Pi4J biblioteka, tikrinant 3 kontaktą, ar įtampa yra aukšta, ar žema.

Jutiklio agentu stebimi tokie išoriniai ištekliai, kaip šviesos intensyvumas ar temperatūra ir kt. 5 sekundes trunka jutiklio atnaujinimo intervalas, kuris apima komunikaciją tarp galinių įrenginių ir tam tikrų agentų, ir jutiklio agento apklausos intervalas, kurį sudaro komunikacija tarp agentų ir orkestratoriaus. Yra keletas komandų, kurias jutiklio agentas gali gauti, pavyzdžiui, „Get“, „Set“, „Move“, „Changeip“. Jei jutiklio agentas gauna „Get“ pranešimą, jis identifikuoja siuntėją ir reaguoja pagal reikšmę, kuri gaunama naudojant „getValue()“ metodą ACL INFORM pranešime. Šie pranešimai siunčiami per įprastas jutiklio apklausas iš anksto nustatytu intervalu, pavyzdžiui, kas 5 sekundes. Taip pat įrenginį galima suaktyvinti, kai pasiekiama ribinė vertė.

Komunikacija su galutiniais įrenginiais, kad gautų jų reikšmes, atliekama COAP protokolu. Apšvietimo ir temperatūros agentai nuolat apklausia savo galinius įrenginius, kad sužinotų reikšmes, naudojantis „onTick()“ įvykiu, kuris periodiškai suaktyvinamas tam tikru laiko intervalu. Reikšmė iš galutinio punkto (angl. *end-point*) įrenginio gaunama COAP užklausos metodu GET. O PUT metodas naudojamas vertei nustatyti.

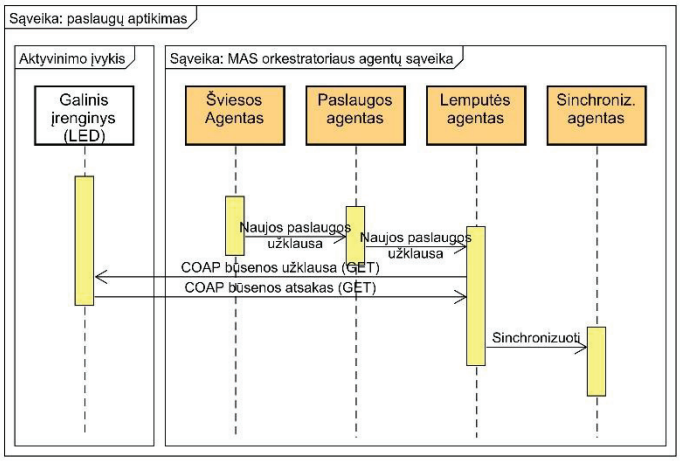


56 pav. Stebėjimo rodmenų inicijuotas paslaugų perskirstymas

Kaip matyti pirmiau pateiktame 55 paveikslėlyje, galutinis įrenginys periodiškai komunikuoja su savo ūko mazgu. Kai stebėjimo agentas nustato, kad jo akumuliatoriaus įtampa yra mažesnė nei reikalaujamas lygis, jis siunčia paslaugos perskirstymo užklausą ūko mazgo sprendimo priėmimo agentui. Nuotolinis paslaugų perskirstymas apskaičiuojamas naudojant IMOPSO ir AHP, o jo sprendimas perduodamas vykdymo agentui. Paslaugos dabartiniame ūko mazge sustabdomos. Perskirstymo sprendimą sinchronizuoja dabartinis ūko mazgas ir tas ūko mazgas, į kurį paslaugos turi būti perkeltos. Sinchronizavimo agentas siunčia užklausą savo vykdymo agentui, o galutinio įrenginio paslauga priskiriama ūko mazgui 1.

9.3. Paslaugų suradimas

Paslaugų užklausos generuojamos, kai atsiranda naujas galutinis įrenginys arba dabartinis galutinis įrenginys juda iš vienos vietos į kitą. Dabartiniai ūko mazgo galutiniai įrenginiai atjungiami, ir jiems reikia išsiųsti užklausą į artimesnį ūko mazgą. Šie galutiniai įrenginiai identifikuojami pagal jų galutinio punkto adresą, pavyzdžiui, coap://192.168.0.24/temp temperatūros įrenginiui arba coap://192.168.0.24/led apšvietimui, kurį galima pakeisti arba pritaikyti pagal poreikį (žr. 56 pav.).



57 pav. Paslaugų suradimas

Apšvietimo agentas veikia kaip šviesos jutimo elementas. Jei šviesos lygis lauke nukrenta žemiau tam tikros ribos, jis siunčia užklausą savo paslaugų agentui, kad paleistų naują paslaugą. Paslaugų agentas nustato, kokia turėtų būti ši paslauga. Kai reikia apšvietimo paslaugos, ji gali apibrėžti, koks šviesos intensyvumas yra reikalingas. Tada paslaugos agentas siunčia nustatytą užklausą „šviestuvo“ agentui. Su galutiniu įrenginiu komunikuojama PUT metodu, siekiant nustatyti reikiamą šviesos lygį. Kai lygis pakoreguojamas, rezultatas sinchronizuojamas.

9.4. Duomenų sinchronizavimas

Šiuo metu sinchronizavimas tarp orkestratoriaus agentų ir komunikacija tarp ūko mazgų yra įgyvendinami kaip trikryptė komunikacijos topologija. Vertikalioji sinchronizacija su „motininiu“ agentu yra nukreipiama aukštyn, einant hierarchijoje vienu lygiu aukštyn. Komunikacija žemyn yra skirta siųsti pranešimus vienu lygiu žemyn į „vaikų“ agentą. Tai atliekama siunčiant ACL INFORM pranešimus vidiniais ūko mazgų agentais. Šoninė komunikacija yra horizontalus ryšys tarp netoliese esančių ūko mazgų, kurį palaiko sinchronizacijos agentai. Visi ūko mazgai turi nuolat atnaujinti savo duomenis apie išteklius ir būsenas, kad galėtų priimti pagrįstus sprendimus dėl geriausio galimo paslaugų įkėlimo, paleidžiant naują paslaugą. Tai taip pat būtina, kai tuo metu teikiamos paslaugos perkeliamos dėl išteklių ar saugumo apribojimų. Šiuo metu visa horizontalioji komunikacija vyksta „Wi-Fi“ protokolu, bet tai taip pat galima atlikti per „Bluetooth“, „ZigBee“ arba net eternetu.

```
 :sender  ( agent-identifier :name FogOrchestrator4
@fog4  :addresses (sequence http://192.168.0.90:7774 ))
 :receiver  (set ( agent-identifier :name FogOrchestrator4
@fog4  :addresses (sequence http://192.168.0.90:7774 )) )
 :content  "AgentList:4:9:FogOrchestrator4:sniffer0:sniffer0-
on-fog4-cnt:Temp4:sniffer1:Light4:Decision4:Bulb4:sniffer1-
on-fog4-cnt"
```

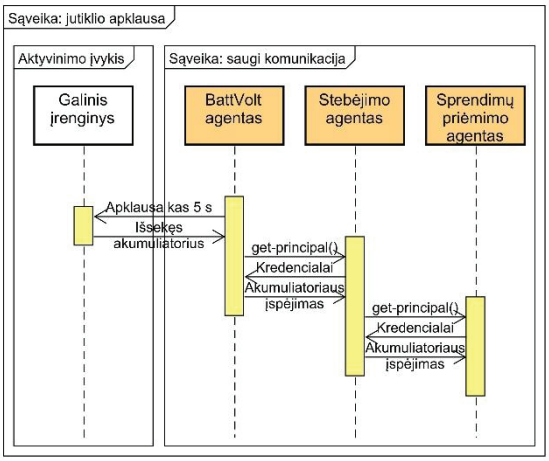
58 pav. Agentų sąrašo sinchronizavimo pranešimai

Kad agentų sąrašas būtų sinchronizuotas, jis atnaujinamas naudojant AMS paslaugų komponentą. Komunikacijos pranešimai pavaizduoti 57 paveikslėlyje. Šis komponentas leidžia pradėti paiešką, pagrįstą tam tikrais apribojimais ir aprašymais. Jis stebi agentų registravimą, išregistravimą ir seka juos. Kad būtų gautas sąrašas, ūko mazguose siunčiamas „GetAgentList“ pranešimas visiems žinomiems orkestratoriams. Saugomas galimų orkestratorių sąrašas su jų IP adresais, bet jų buvimas ūko tinkle patvirtinamas atsakymu.

9.5. Saugumo priežiūra

Pagal numatytąją konfigūraciją agentų komunikacijos pranešimai nėra nei užšifruoti, nei pasirašyti. Tai gali suteikti galimybę įsilaužėliui, naudojančiam kenkėjišką agentą, šnipinėti pranešimų turinį arba net jį modifikuoti. Gavimo agentas gautų kenkėjiškas instrukcijas. AMS galima siųsti įvairius prašymus, kad būtų pašalinti kai kurie agentai, jei nėra saugumo patikrinimų. Siekiant išlaikyti saugumą, panaudotas JADE saugumo priedas JADE-S. Įdiegus priedą, turi būti įjungtos tokios paslaugos, kaip „SecurityService“, „PermissionService“, „SignaureService“ ir „EncryptionService“. „SecurityService“ yra pagrindinė paslauga, o kitos yra pasirenkamos, jas galima įjungti pagal poreikį.

Vietoj to, kad būtų naudojamas metodas „send()“, naudojamas „sendMessage()“. Jis leidžia nurodyti, ar pranešimas turi būti pasirašytas ir užšifruotas. Prisijungimo duomenys tikrinami pagal metodą „retrievePrincipal()“. Jis paleidžiamas prieš išsiunčiant atitinkamą pranešimą. Pirmiausia jis išsiunčia užklausos pranešimą su turiniu „get-principal“, o atsakymą suformuoja gavėjas, kurį išsiunčia kaip informacinį pranešimą (žr. 58 pav.).



59 pav. Prisijungimo duomenų nuskaitymas saugiu ryšiu

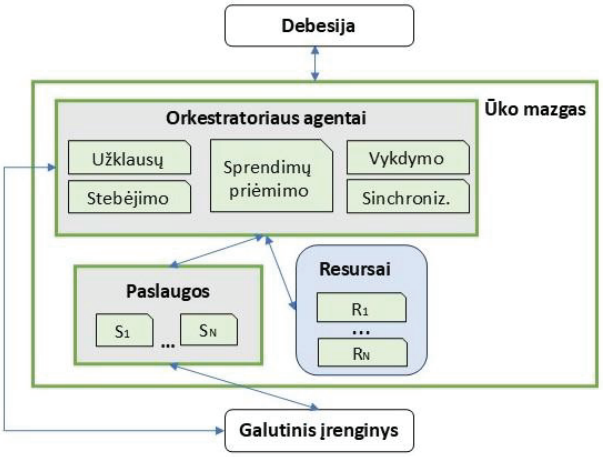
Be to, reikia aktyvinti papildomus modulius, kad numatytieji saugos parametrai būtų pritaikyti pagal pageidavimus. „SignAlgorithm“ leidžia pasirinkti algoritmą, kuris bus naudojamas pranešimams pasirašyti, pavyzdžiui, MD5withRSA arba DSA. Viešųjų ir privačių raktų dydį galima apibrėžti modulyje „AsimKeySize“, kuris gali kisti nuo 512 kaip numatytoji reikšmė iki 2048. Taip pat esama kitų modulių, nepaisant pirmiau minėtų, leidžiančių užtikrinti papildomas saugumo parinktis.

9.6. Dinaminio orkestratoriaus architektūra

Paslaugų dinaminis orkestratorius buvo realizuotas kaip daugiaagentės sistemos (MAS) „Java“ programa, naudojant JADE karkasą. Karkasas leidžia kurti MAS programas, atitinkančias FIPA specifikacijas. Jis siūlo tokias funkcijas, kaip agentų abstrakcija, asinchroniniai pranešimai, paslaugų suradimas geltonųjų puslapių metodu. Orkestratorius veikia kaip paskirstytas stebėsenos, sprendimų priėmimo ir vykdymo metodas, kuris yra įdiegiamas kiekvieno ūko mazgo atitinkamoje ūko kompiuterijos sistemoje. Nuolatinis išteklių stebėjimas ir prašymų tvarkymas leidžia dinamiškai priimti informuotus sprendimus. Paslaugų paskirstymas tarp kelių mazgų prisideda prie optimalesnių skaičiavimo rezultatų, energijos naudojimo ir atsparumo didinimo. Paskirstytas orkestratorius yra atsparesnis nei centralizuotas dėl to, kad nėra kritinės rizikos vietos (SPF). Dėl judumo, kintančių išteklių lygio ir ūko mazgų sutrikimų pirmenybė teikiama dinaminiam ir paskirstytam sprendimų priėmimui.

Kai ūko mazgai prisijungia prie to paties tinklo, kad sudarytų bendrą infrastruktūrą, jie pradeda stebėti savo išteklius, tokius kaip CPU, RAM, akumuliatorius ir saugumas. Stebėjimo agentas laukia su ištekliais susijusių pranešimų, naudodamas ciklinį elgesį. Šie pranešimai klasifikuojami pagal panaudojimo atvejus ir juose nurodomi atitinkami išteklių lygiai. Užklausų agentas laukia paslaugų užklausos pranešimų. Jie taip pat klasifikuojami pagal panaudojimo atvejus. Dėl saugumo naudojamas priedas JADE-S leidžia pranešimus pasirašyti, užšifruoti arba tiek pasirašyti, tiek užšifruoti. Dalyvaujančiuose agentuose yra

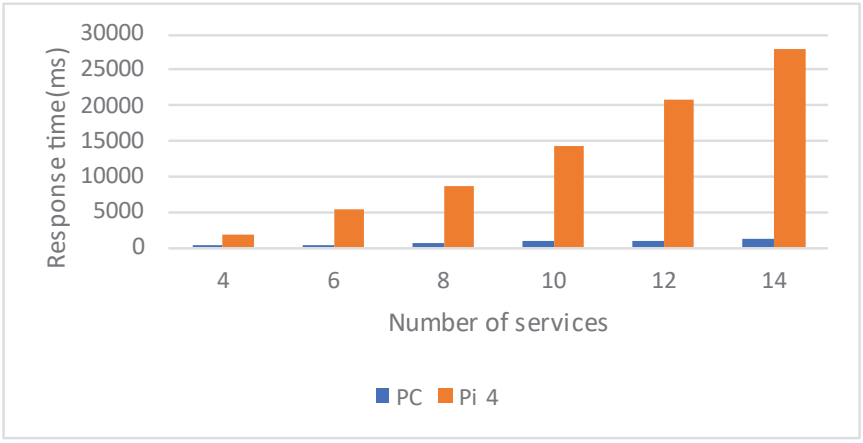
nustatomos būtinos saugumo priemonės ir naudojamos saugumo priemonės, o pranešimus galima atmesti, jei šie reikalavimai nėra įvykdyti (žr. 59 pav.) [121].



60 pav. Paskirstyto orkestratoriaus architektūra

9.7. Masto pritaikymo įvertinimas

Toliau pateiktoje diagramoje parodytas paslaugų išdėstymo efektyvumas, kai mastas didėja [128]. Ūko mazgų, paslaugų, dalelių ir epochų (iteracijų) skaičius laipsniškai didinamas. Kiekvienas diagramos taškas buvo apskaičiuotas, kiekvienam parametrų rinkiniui atliekant vidutiniškai 20 bandymų. Naudojami 6 parametrų rinkiniai, kurie išvardyti toliau lentelėje. 61 paveikslėlyje pavaizduota masto diagrama.



61 pav. Paslaugų išdėstymo radimo algoritmo priklausymas nuo masto

2 lentelėje apibendrinti naudoti parametrai.

15 lentelė. Masto įvertinimo parametrų rinkiniai

Ūko mazgai	Paslaugos	Dalelės	Epochos	Atsako laikas (ms) su kompiuteriu	Atsako laikas (ms) su „Pi 4“
4	12	200	300	121	1802
6	18	300	300	305	5432
8	24	400	400	498	8644
10	30	500	600	684	14401
12	36	600	700	747	20923
14	42	600	800	1161	27923

Iš toliau atliktų eksperimentų galima nustatyti šias priklausomybes:

- Paslaugų išdėstymo trukmė pasižymi linijine priklausomybe nuo dalelių ir epochų (iteracijų) skaičiaus. Naujų paslaugų paleidimo ir paslaugų perskirstymo eksperimentuose pagal nutylėjimą buvo naudojama 200 dalelių ir 400 epochų. Padidinus bet kokius šiuos parametrus, linijiniu būdu padidėja išdėstymo trukmė.
- Paslaugų išdėstymo trukmė pasižymi linijine priklausomybe nuo uždavinio sudėtingumo, kurį apibrėžia ūko mazgų ir paslaugų skaičius. Naujų paslaugų paleidimo ir paslaugų perskirstymo eksperimentuose pagal nutylėjimą buvo naudojami 2 ūko mazgai ir 12 paslaugų. Padidinus bet kokius šiuos kompleksiškumo parametrus, linijiniu būdu padidėja išdėstymo trukmė.
- Iš paslaugų išdėstymo delsos matyti, kad sprendžiamo uždavinio kompleksiškumą galima apibrėžti iki kelių dešimčių ūko mazgų arba paslaugų. Ypač su santykinai mažos skaičiavimo galios įrenginiais, tokiais kaip „Raspberry Pi“.

9.8. Išvados

1. Kaip Paslaugų dinaminis orkestravimo metodas realizuotas kaip JADE platformos prototipas. Aprašyta prototipo sąveika su jo aplinka, įskaitant išteklių stebėjimą, naujų paslaugų paleidimą, duomenų sinchronizavimą, komunikacijos saugumo užtikrinimą ir architektūrą. MAS agentų sąveika perteikiama sekų diagramomis.
2. Buvo atlikti įvairūs eksperimentai, siekiant įvertinti metodo efektyvumą skirtingomis sąlygomis. Šie eksperimentai gali būti klasifikuojami kaip parametrų testai, perkrovos testai, saugumo testai, energijos sąnaudų testai ir masto pritaikymo testai.
3. Visgi daugiausiai skaičiavimo resursų reikalauja dviejų pakopų optimizavimo procesas. Bendroji delsa, kai naudojamas kompiuteris, gali siekti iki 50 %, bet ji gali pasiekti 70 % ir daugiau, kai naudojami mažiau galingi įrenginiai, pavyzdžiui, „Raspberry Pi 4“. Optimizavimas gali sudaryti dar didesnę laiko dalį, jei dislokuojamų paslaugų ir ūko mazgų skaičius didėja.

BENDROSIOS TYRIMO IŠVADOS

1. Siekiant nustatyti ūko kompiuterijos iššūkius, 2 skyriuje peržiūrėta daugiau nei 150 tyrimų. Iššūkiai buvo suklasifikuoti, siekiant išskirti populiariausius. Kaip matyti iš rezultatų, dažniausiai kalbama apie saugumą / privatumą, dinamiką / mobilumą, ir masto pritaikymą / sudėtingumą.
2. Naujas optimizavimo procesas, remiantis „Matlab“ platformoje atliktu modeliavimu, optimaliai paskirsto paslaugas. Dviejų etapų procesas yra sudarytas iš daugiatikslio dalelių spiečiaus optimizavimo (IMOPSO) algoritmo, skirto dalelių rinkiniams generuoti, ir analitinio hierarchijos proceso (AHP) su konkrečia prioritetų matrica, skirta paslaugų paskirstymo prioritetiniams sprendimams. Optimizavimo procesas įvertina skirtingus heterogeninius kriterijus su skirtingais kokybiniais ir kiekybiniais matavimo vienetais.
3. Ūko kompiuterijos paslaugų dinaminio orkestravimo prototipas sąveikauja su aplinka kaip daugiaagentė sistema (MAS). Orkestratoriai veikia visuose ūko mazguose, kuriuose jie priima sprendimus dėl optimizuoto paslaugų įkėlimo, kai įvertina tokius išteklius, kaip procesorius, atmintis, akumuliatorius ir saugumas. Šis paslaugų dinaminis orkestravimo metodas yra svarbus tuo, kad jis nėra pagrįstas centriniu valdymo bloku. Sprendimą dėl paslaugų optimizuoto įkėlimo gali priimti bet kuris orkestratorius, kai jie sinchronizuoja savo prieinamus išteklius su kitais ūko tinklo mazgais. Tai itin pagerina masto pritaikymą ir atsparumą ūko mazgų sutrikimams. Paslaugų dinaminio orkestravimo metodo prototipas pritaikytas judumui, jis taip pat prisitaiko prie išteklių prieinamumo pokyčių.

REFERENCES

[1] J. H. Nord, A. Koohang, and J. Paliszkiewicz, "The Internet of Things: Review and theoretical framework," *Expert Systems with Applications*, vol. 133, pp. 97–108, Nov. 2019, doi: 10.1016/j.eswa.2019.05.014.

[2] P. Hosseinioun, M. Kheirabadi, S. R. Kamel Tabbakh, and R. Ghaemi, "aTask scheduling approaches in fog computing: A survey," *Transactions on Emerging Telecommunications Technologies*, vol. 33, no. 3, p. e3792, 2022, doi: 10.1002/ett.3792.

[3] Z. Ali, S. A. Chaudhry, K. Mahmood, S. Garg, Z. Lv, and Y. B. Zikria, "A clogging resistant secure authentication scheme for fog computing services," *Computer Networks*, vol. 185, p. 107731, Feb. 2021, doi: 10.1016/j.comnet.2020.107731.

[4] K. E. Srinivasa Desikan, V. J. Kotagi, and C. Siva Ram Murthy, "Topology Control in Fog Computing Enabled IoT Networks for Smart Cities," *Computer Networks*, vol. 176, p. 107270, Jul. 2020, doi: 10.1016/j.comnet.2020.107270.

[5] A. Liutkevičius, N. Morkevičius, A. Venčkauskas, and J. Toldinas, "Distributed Agent-Based Orchestrator Model for Fog Computing," *Sensors*, vol. 22, no. 15, Art. no. 15, Jan. 2022, doi: 10.3390/s22155894.

[6] F. A. Salaht, F. Desprez, and A. Lebre, "An Overview of Service Placement Problem in Fog and Edge Computing," *ACM Comput. Surv.*, vol. 53, no. 3, pp. 1–35, May 2021, doi: 10.1145/3391196.

[7] 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," *Internet of Things*, vol. 12, p. 100273, Dec. 2020, doi: 10.1016/j.iot.2020.100273.

[8] B. Costa, J. Bachiega, L. R. de Carvalho, and A. P. F. Araujo, "Orchestration in Fog Computing: A Comprehensive Survey," *ACM Comput. Surv.*, vol. 55, no. 2, p. 29:1-29:34, Jan. 2022, doi: 10.1145/3486221.

[9] G. Lee, W. Saad, and M. Bennis, "An Online Optimization Framework for Distributed Fog Network Formation With Minimal Latency," *IEEE Transactions on Wireless Communications*, vol. 18, no. 4, pp. 2244–2258, Apr. 2019, doi: 10.1109/TWC.2019.2901850.

[10] K. Toczé and S. Nadjm-Tehrani, "ORCH: Distributed Orchestration Framework using Mobile Edge Devices," in *2019 IEEE 3rd International Conference on Fog and Edge Computing (ICFEC)*, May 2019, pp. 1–10. doi: 10.1109/CFEC.2019.8733152.

[11] C. Guerrero, I. Lera, and C. Juiz, "A lightweight decentralized service placement policy for performance optimization in fog computing," *Journal of Ambient Intelligence and Humanized Computing*, vol. 10, no. 6, pp. 2435–2452, Jun. 2019, doi: 10.1007/s12652-018-0914-0.

[12] A. Aral and T. Ovatman, "A Decentralized Replica Placement Algorithm for Edge Computing," *IEEE Transactions on Network and Service Management*, vol. 15, no. 2, pp. 516–529, Jun. 2018, doi: 10.1109/TNSM.2017.2788945.

[13] P. Wang, S. Liu, F. Ye, and X. Chen, "A Fog-based Architecture and Programming Model for IoT Applications in the Smart Grid," Apr. 04, 2018, *arXiv*: arXiv:1804.01239. Accessed: Sep. 03, 2022. [Online]. Available: http://arxiv.org/abs/1804.01239

[14] A. Rayes and S. Salam, "Fog Computing," in *Internet of Things from Hype to Reality: The Road to Digitization*, A. Rayes and S. Salam, Eds., Cham: Springer International Publishing, 2022, pp. 153–178. doi: 10.1007/978-3-030-90158-5_6.

[15] S. N. Srirama, "A decade of research in fog computing: Relevance, challenges, and future directions," *Software: Practice and Experience*, vol. 54, no. 1, pp. 3–23, 2024, doi: 10.1002/spe.3243.

[16] X. Mu and M. F. Antwi-Afari, “The applications of Internet of Things (IoT) in industrial management: a science mapping review,” *International Journal of Production Research*, vol. 62, no. 5, pp. 1928–1952, Mar. 2024, doi: 10.1080/00207543.2023.2290229.

[17] A. Hazra, P. Rana, M. Adhikari, and T. Amgoth, “Fog computing for next-generation Internet of Things: Fundamental, state-of-the-art and research challenges,” *Computer Science Review*, vol. 48, p. 100549, May 2023, doi: 10.1016/j.cosrev.2023.100549.

[18] A. Yousefpour *et al.*, “All one needs to know about fog computing and related edge computing paradigms: A complete survey,” *Journal of Systems Architecture*, vol. 98, pp. 289–330, Sep. 2019, doi: 10.1016/j.sysarc.2019.02.009.

[19] “THE INDUSTRIAL INTERNET CONSORTIUM AND OPENFOG CONSORTIUM JOIN FORCES | Industrial Internet Consortium.” Accessed: Dec. 19, 2019. [Online]. Available: https://www.iiconsortium.org/press-room/12-18-18.htm

[20] O. C. A. W. Group, “OpenFog reference architecture for fog computing,” *OPFRA001*, vol. 20817, p. 162, 2017.

[21] 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*, in MCC ’12. New York, NY, USA: ACM, 2012, pp. 13–16. doi: 10.1145/2342509.2342513.

[22] R. Das and M. M. Inuwa, “A review on fog computing: Issues, characteristics, challenges, and potential applications,” *Telematics and Informatics Reports*, vol. 10, p. 100049, Jun. 2023, doi: 10.1016/j.teler.2023.100049.

[23] S. Mirampalli, R. Wankar, and S. N. Srirama, “Evaluating NiFi and MQTT based serverless data pipelines in fog computing environments,” *Future Generation Computer Systems*, vol. 150, pp. 341–353, Jan. 2024, doi: 10.1016/j.future.2023.09.014.

[24] M. Aldossary, “Multi-Layer Fog-Cloud Architecture for Optimizing the Placement of IoT Applications in Smart Cities,” *Computers, Materials and Continua*, vol. 75, pp. 633–649, Feb. 2023, doi: 10.32604/cmc.2023.035414.

[25] M. Aqib, D. Kumar, and S. Tripathi, “Machine Learning for Fog Computing: Review, Opportunities and a Fog Application Classifier and Scheduler,” *Wireless Pers Commun*, vol. 129, no. 2, pp. 853–880, Mar. 2023, doi: 10.1007/s11277-022-10160-y.

[26] H. Tran-Dang and D.-S. Kim, “Fog Computing: Fundamental Concepts and Recent Advances in Architectures and Technologies,” in *Cooperative and Distributed Intelligent Computation in Fog Computing: Concepts, Architectures, and Frameworks*, H. Tran-Dang and D.-S. Kim, Eds., Cham: Springer Nature Switzerland, 2023, pp. 1–18. doi: 10.1007/978-3-031-33920-2_1.

[27] A. V. Dastjerdi, H. Gupta, R. N. Calheiros, S. K. Ghosh, and R. Buyya, “Chapter 4 - Fog Computing: principles, architectures, and applications,” in *Internet of Things*, R. Buyya and A. Vahid Dastjerdi, Eds., Morgan Kaufmann, 2016, pp. 61–75. doi: 10.1016/B978-0-12-805395-9.00004-6.

[28] M. Waqas, S. Tu, J. Wan, T. Mir, H. Alasmary, and G. Abbas, “Defense scheme against advanced persistent threats in mobile fog computing security,” *Computer Networks*, vol. 221, p. 109519, Feb. 2023, doi: 10.1016/j.comnet.2022.109519.

[29] S. Aggarwal and N. Kumar, “Fog Computing for 5G-Enabled Tactile Internet: Research Issues, Challenges, and Future Research Directions,” *Mobile Netw Appl*, vol. 28, no. 2, pp. 690–717, Apr. 2023, doi: 10.1007/s11036-019-01430-4.

[30] Q. Vu Khanh, N. Vi Hoai, A. Dang Van, and Q. Nguyen Minh, “An integrating computing framework based on edge-fog-cloud for internet of healthcare things applications,” *Internet of Things*, vol. 23, p. 100907, Oct. 2023, doi: 10.1016/j.iot.2023.100907.

[31] M. Burhan *et al.*, "A Comprehensive Survey on the Cooperation of Fog Computing Paradigm-Based IoT Applications: Layered Architecture, Real-Time Security Issues, and Solutions," *IEEE Access*, vol. 11, pp. 73303–73329, 2023, doi: 10.1109/ACCESS.2023.3294479.

[32] A. Dorri, S. S. Kanhere, and R. Jurdak, "Multi-Agent Systems: A Survey," *IEEE Access*, vol. 6, pp. 28573–28593, 2018, doi: 10.1109/ACCESS.2018.2831228.

[33] J. Wang, X. Deng, J. Guo, and Z. Zeng, "Resilient Consensus Control for Multi-Agent Systems: A Comparative Survey," *Sensors*, vol. 23, no. 6, Art. no. 6, Jan. 2023, doi: 10.3390/s23062904.

[34] H. G. Abreha, C. J. Bernardos, A. D. L. Oliva, L. Cominardi, and A. Azcorra, "Monitoring in fog computing: state-of-the-art and research challenges," *International Journal of Ad Hoc and Ubiquitous Computing*, Feb. 2021, Accessed: Oct. 05, 2022. [Online]. Available: https://www.inderscienceonline.com/doi/10.1504/IJAHUC.2021.113384

[35] B. Costa, J. Bachiega, L. R. Carvalho, M. Rosa, and A. Araujo, "Monitoring fog computing: A review, taxonomy and open challenges," *Computer Networks*, vol. 215, p. 109189, Oct. 2022, doi: 10.1016/j.comnet.2022.109189.

[36] N. Morkevicius, A. Venčkauskas, N. Šatkauskas, and J. Toldinas, "Method for Dynamic Service Orchestration in Fog Computing," *Electronics*, vol. 10, no. 15, p. 1796, Jul. 2021, doi: 10.3390/electronics10151796.

[37] A. Miele, H. Zárate, L. Cassano, C. Bolchini, and J. E. Ortiz, "A Runtime Resource Management and Provisioning Middleware for Fog Computing Infrastructures," *ACM Trans. Internet Things*, vol. 3, no. 3, p. 17:1-17:29, Apr. 2022, doi: 10.1145/3506718.

[38] A. Katal and V. Sethi, "Communication Protocols in Fog Computing: A Survey and Challenges," 2022, pp. 153–170. doi: 10.1201/9781003188230-11.

[39] J. Dizdarević, F. Carpio, A. Jukan, and X. Masip-Bruin, "A Survey of Communication Protocols for Internet of Things and Related Challenges of Fog and Cloud Computing Integration," *ACM Comput. Surv.*, vol. 51, no. 6, pp. 1–29, Feb. 2019, doi: 10.1145/3292674.

[40] N. S. Mohd Pakhrudin, M. Kassim, and A. Idris, "A review on orchestration distributed systems for IoT smart services in fog computing," *IJECE*, vol. 11, no. 2, p. 1812, Apr. 2021, doi: 10.11591/ijece.v11i2.pp1812-1822.

[41] A. Bozorgchenani, D. Tarchi, and G. E. Corazza, "Centralized and Distributed Architectures for Energy and Delay Efficient Fog Network-Based Edge Computing Services," *IEEE Transactions on Green Communications and Networking*, vol. 3, no. 1, pp. 250–263, Mar. 2019, doi: 10.1109/TGCN.2018.2885443.

[42] F. Firouzi, B. Farahani, and A. Marinšek, "The convergence and interplay of edge, fog, and cloud in the AI-driven Internet of Things (IoT)," *Information Systems*, vol. 107, p. 101840, Jul. 2022, doi: 10.1016/j.is.2021.101840.

[43] M. Whaiduzzaman, A. Barros, A. R. Shovon, M. R. Hossain, and C. Fidge, "A Resilient Fog-IoT Framework for Seamless Microservice Execution," in *2021 IEEE International Conference on Services Computing (SCC)*, Sep. 2021, pp. 213–221. doi: 10.1109/SCC53864.2021.00034.

[44] A. A. Kirsanova, G. I. Radchenko, and A. N. Tchernykh, "Fog computing state of the art: concept and classification of platforms to support distributed computing systems," Jun. 22, 2021, *arXiv*: arXiv:2106.11726. Accessed: Oct. 01, 2022. [Online]. Available: http://arxiv.org/abs/2106.11726

[45] S. Forti, M. Gaglianese, and A. Brogi, "Lightweight self-organising distributed monitoring of Fog infrastructures," *Future Generation Computer Systems*, vol. 114, pp. 605–618, Jan. 2021, doi: 10.1016/j.future.2020.08.011.

[46] H. K. Apat, R. Nayak, and B. Sahoo, “A comprehensive review on Internet of Things application placement in Fog computing environment,” *Internet of Things*, vol. 23, p. 100866, Oct. 2023, doi: 10.1016/j.iot.2023.100866.

[47] M. Trabelsi, N. Bendaoud, and S. Ben Ahmed, “A Dynamic Service Placement in Fog Infrastructure:,” in *Proceedings of the 18th International Conference on Evaluation of Novel Approaches to Software Engineering*, Prague, Czech Republic: SCITEPRESS - Science and Technology Publications, 2023, pp. 444–452. doi: 10.5220/0011852400003464.

[48] M. Qin, M. Li, and R. Othman Yahya, “Dynamic IoT service placement based on shared parallel architecture in fog-cloud computing,” *Internet of Things*, vol. 23, p. 100856, Oct. 2023, doi: 10.1016/j.iot.2023.100856.

[49] Y. Abofathi, B. Anari, and M. Masdari, “A learning automata based approach for module placement in fog computing environment,” *Expert Systems with Applications*, vol. 237, p. 121607, Mar. 2024, doi: 10.1016/j.eswa.2023.121607.

[50] M. Zare, Y. Elmi Sola, and H. Hasanpour, “Towards distributed and autonomous IoT service placement in fog computing using asynchronous advantage actor-critic algorithm,” *Journal of King Saud University - Computer and Information Sciences*, vol. 35, no. 1, pp. 368–381, Jan. 2023, doi: 10.1016/j.jksuci.2022.12.006.

[51] F. A. Saif, R. Latip, Z. M. Hanapi, and K. Shafinah, “Multi-Objective Grey Wolf Optimizer Algorithm for Task Scheduling in Cloud-Fog Computing,” *IEEE Access*, vol. 11, pp. 20635–20646, 2023, doi: 10.1109/ACCESS.2023.3241240.

[52] M. Mokni, S. Yassa, J. E. Hajlaoui, M. N. Omri, and R. Chelouah, “Multi-objective fuzzy approach to scheduling and offloading workflow tasks in Fog–Cloud computing,” *Simulation Modelling Practice and Theory*, vol. 123, p. 102687, Feb. 2023, doi: 10.1016/j.simpat.2022.102687.

[53] F. A. Saif, R. Latip, Z. M. Hanapi, M. A. Alrshah, and S. Kamarudin, “Workload Allocation Toward Energy Consumption-Delay Trade-Off in Cloud-Fog Computing Using Multi-Objective NPSO Algorithm,” *IEEE Access*, vol. 11, pp. 45393–45404, 2023, doi: 10.1109/ACCESS.2023.3266822.

[54] H. Sulimani *et al.*, “Reinforcement optimization for decentralized service placement policy in IoT-centric fog environment,” *Transactions on Emerging Telecommunications Technologies*, vol. 34, no. 11, p. e4650, 2023, doi: 10.1002/ett.4650.

[55] M. M. Islam, F. Ramezani, H. Y. Lu, and M. Naderpour, “Optimal placement of applications in the fog environment: A systematic literature review,” *Journal of Parallel and Distributed Computing*, vol. 174, pp. 46–69, Apr. 2023, doi: 10.1016/j.jpdc.2022.12.001.

[56] S. Azizi, M. Shojafar, P. Farzin, and J. Dogani, “DCSP: A delay and cost-aware service placement and load distribution algorithm for IoT-based fog networks,” *Computer Communications*, vol. 215, pp. 9–20, Feb. 2024, doi: 10.1016/j.comcom.2023.12.016.

[57] K. Ostrowski, K. Małecki, P. Dziurzański, and A. K. Singh, “Mobility-aware fog computing in dynamic networks with mobile nodes: A survey,” *Journal of Network and Computer Applications*, vol. 219, p. 103724, Oct. 2023, doi: 10.1016/j.jnca.2023.103724.

[58] S. O. Ogundoyin and I. A. Kamil, “Optimal fog node selection based on hybrid particle swarm optimization and firefly algorithm in dynamic fog computing services,” *Engineering Applications of Artificial Intelligence*, vol. 121, p. 105998, May 2023, doi: 10.1016/j.engappai.2023.105998.

[59] H. S. Ali and R. Sridevi, “Mobility and Security Aware Real-Time Task Scheduling in Fog-Cloud Computing for IoT Devices: A Fuzzy-Logic Approach,” *The Computer Journal*, p. bxad019, Apr. 2023, doi: 10.1093/comjnl/bxad019.

[60] K. M. Matrouk and A. D. Matrouk, “Mobility Aware-Task Scheduling and Virtual Fog for Offloading in IoT-Fog-Cloud Environment,” *Wireless Pers Commun*, vol. 130, no. 2, pp. 801–836, May 2023, doi: 10.1007/s11277-023-10310-w.

[61] Y. Jiang, Z. Huang, and D. H. K. Tsang, “Challenges and Solutions in Fog Computing Orchestration,” *IEEE Network*, vol. 32, no. 3, pp. 122–129, May 2018, doi: 10.1109/MNET.2017.1700271.

[62] N. Šatkauskas, A. Venčkauskas, N. Morkevičius, and A. Liutkevičius, “Orchestration Security Challenges in the Fog Computing,” in *International Conference on Information and Software Technologies*, Springer, 2020, pp. 196–207. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-030-59506-7_17

[63] A. Venčkauskas, V. Štuikys, J. Toldinas, and N. Jusas, “A Model-Driven Framework to Develop Personalized Health Monitoring,” *Symmetry*, vol. 8, no. 7, p. 65, Jul. 2016, doi: 10.3390/sym8070065.

[64] M. S. de Brito *et al.*, “A service orchestration architecture for Fog-enabled infrastructures,” in *2017 Second International Conference on Fog and Mobile Edge Computing (FMEC)*, May 2017, pp. 127–132. doi: 10.1109/FMEC.2017.7946419.

[65] K. Velasquez *et al.*, “Fog orchestration for the Internet of Everything: state-of-the-art and research challenges,” *Journal of Internet Services and Applications*, vol. 9, no. 1, p. 14, Jul. 2018, doi: 10.1186/s13174-018-0086-3.

[66] H. Zahmatkesh and F. Al-Turjman, “Fog computing for sustainable smart cities in the IoT era: Caching techniques and enabling technologies - an overview,” *Sustainable Cities and Society*, vol. 59, p. 102139, Aug. 2020, doi: 10.1016/j.scs.2020.102139.

[67] C. Zhang, “Design and application of fog computing and Internet of Things service platform for smart city,” *Future Generation Computer Systems*, vol. 112, pp. 630–640, Nov. 2020, doi: 10.1016/j.future.2020.06.016.

[68] R. Yang, Z. Wen, D. McKee, T. Lin, J. Xu, and P. Garraghan, “Fog Orchestration and Simulation for IoT Services,” in *Fog and Fogonomics*, Y. Yang, J. Huang, T. Zhang, and J. Weinman, Eds., Wiley, 2020, pp. 179–212. Accessed: May 10, 2021. [Online]. Available: https://eprints.lancs.ac.uk/id/eprint/128717/

[69] Z. Wen, R. Yang, P. Garraghan, T. Lin, J. Xu, and M. Rovatsos, “Fog Orchestration for Internet of Things Services,” *IEEE Internet Computing*, vol. 21, no. 2, pp. 16–24, 2017, doi: 10.1109/MIC.2017.36.

[70] O. Skarlat, M. Nardelli, S. Schulte, M. Borkowski, and P. Leitner, “Optimized IoT service placement in the fog,” *Service Oriented Computing and Applications*, vol. 11, no. 4, pp. 427–443, Dec. 2017, doi: 10.1007/s11761-017-0219-8.

[71] R. K. Naha, S. Garg, A. Chan, and S. K. Battula, “Deadline-based dynamic resource allocation and provisioning algorithms in Fog-Cloud environment,” *Future Generation Computer Systems*, vol. 104, pp. 131–141, Mar. 2020, doi: 10.1016/j.future.2019.10.018.

[72] K. Khebbeb, N. Hameurlain, and F. Belala, “A Maude-Based rewriting approach to model and verify Cloud/Fog self-adaptation and orchestration,” *Journal of Systems Architecture*, vol. 110, p. 101821, Nov. 2020, doi: 10.1016/j.sysarc.2020.101821.

[73] S. Tuli, S. Poojara, S. N. Srirama, G. Casale, and N. R. Jennings, “COSCO: Container Orchestration using Co-Simulation and Gradient Based Optimization for Fog Computing Environments,” *arXiv:2104.14392 [cs]*, Apr. 2021, Accessed: May 11, 2021. [Online]. Available: http://arxiv.org/abs/2104.14392

[74] D. Rahbari and M. Nickray, “Task offloading in mobile fog computing by classification and regression tree,” *Peer-to-Peer Netw. Appl.*, vol. 13, no. 1, pp. 104–122, Jan. 2020, doi: 10.1007/s12083-019-00721-7.

[75] H. R. Arkian, A. Diyanat, and A. Pourkhalili, “MIST: Fog-based data analytics scheme with cost-efficient resource provisioning for IoT crowdsensing applications,” *Journal of Network and Computer Applications*, vol. 82, pp. 152–165, 2017, doi: https://doi.org/10.1016/j.jnca.2017.01.012.

[76] K. Velasquez, D. P. Abreu, M. Curado, and E. Monteiro, “Service placement for latency reduction in the internet of things,” *Annals of Telecommunications*, vol. 72, no. 1, pp. 105–115, Feb. 2017, doi: 10.1007/s12243-016-0524-9.

[77] A. Brogi, S. Forti, and A. Ibrahim, “How to Best Deploy Your Fog Applications, Probably,” in *2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC)*, 2017, pp. 105–114. doi: 10.1109/ICFEC.2017.8.

[78] R. Urgaonkar, S. Wang, T. He, M. Zafer, K. Chan, and K. K. Leung, “Dynamic Service Migration and Workload Scheduling in Edge-Clouds,” *Perform. Eval.*, vol. 91, no. C, pp. 205–228, Sep. 2015, doi: 10.1016/j.peva.2015.06.013.

[79] J. Bellendorf and Z. Á. Mann, “Classification of optimization problems in fog computing,” *Future Generation Computer Systems*, vol. 107, pp. 158–176, Jun. 2020, doi: 10.1016/j.future.2020.01.036.

[80] X. Huang, S. Ganapathy, and T. Wolf, “Evaluating Algorithms for Composable Service Placement in Computer Networks,” in *2009 IEEE International Conference on Communications*, 2009, pp. 1–6. doi: 10.1109/ICC.2009.5199007.

[81] Z. Á. Mann, “Optimization in computer engineering – Theory and applications,” in *Optimization in computer engineering – Theory and applications*, Scientific Research Publishing, 2011. Accessed: May 14, 2021. [Online]. Available: https://www.scirp.org/book/DetailedInforOfABook.aspx?bookID=486

[82] R. Mahmud, K. Ramamohanarao, and R. Buyya, “Latency-Aware Application Module Management for Fog Computing Environments,” *ACM Trans. Internet Technol.*, vol. 19, no. 1, p. 9:1-9:21, Nov. 2018, doi: 10.1145/3186592.

[83] M. Chen, W. Li, G. Fortino, Y. Hao, L. Hu, and I. Humar, “A Dynamic Service Migration Mechanism in Edge Cognitive Computing,” *ACM Trans. Internet Technol.*, vol. 19, no. 2, p. 30:1-30:15, Apr. 2019, doi: 10.1145/3239565.

[84] S. Filiposka, A. Mishev, and K. Gilly, “Mobile-aware dynamic resource management for edge computing,” *Transactions on Emerging Telecommunications Technologies*, vol. 30, no. 6, p. e3626, 2019, doi: 10.1002/ett.3626.

[85] A. Mseddi, W. Jaafar, H. Elbiaze, and W. Ajib, “Joint Container Placement and Task Provisioning in Dynamic Fog Computing,” *IEEE Internet of Things Journal*, vol. 6, no. 6, pp. 10028–10040, Dec. 2019, doi: 10.1109/JIOT.2019.2935056.

[86] S. Jošilo and G. Dán, “Decentralized Algorithm for Randomized Task Allocation in Fog Computing Systems,” *IEEE/ACM Transactions on Networking*, vol. 27, no. 1, pp. 85–97, Feb. 2019, doi: 10.1109/TNET.2018.2880874.

[87] C. Zhu *et al.*, “Folo: Latency and Quality Optimized Task Allocation in Vehicular Fog Computing,” *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4150–4161, Jun. 2019, doi: 10.1109/JIOT.2018.2875520.

[88] H. Zhao, S. Wang, and H. Shi, “Fog-computing based mobility and resource management for resilient mobile networks,” *High-Confidence Computing*, p. 100193, Nov. 2023, doi: 10.1016/j.hcc.2023.100193.

[89] M. Ebrahim and A. Hafid, “Resilience and load balancing in Fog networks: A Multi-Criteria Decision Analysis approach,” *Microprocessors and Microsystems*, vol. 101, p. 104893, Sep. 2023, doi: 10.1016/j.micpro.2023.104893.

[90] S. Pallewatta, V. Kostakos, and R. Buyya, “MicroFog: A framework for scalable placement of microservices-based IoT applications in federated Fog environments,”

Journal of Systems and Software, vol. 209, p. 111910, Mar. 2024, doi: 10.1016/j.jss.2023.111910.
[91] C. Núñez-Gómez, C. Carrión, B. Caminero, and F. M. Delicado, "S-HIDRA: A blockchain and SDN domain-based architecture to orchestrate fog computing environments," *Computer Networks*, vol. 221, p. 109512, Feb. 2023, doi: 10.1016/j.comnet.2022.109512.
[92] J. Dogani, A. Yazdanpanah, A. Zare, and F. Khunjush, "A two-tier multi-objective service placement in container-based fog-cloud computing platforms," *Cluster Comput*, Nov. 2023, doi: 10.1007/s10586-023-04183-8.
[93] R. C. Sofia, D. Dykeman, P. Urbanetz, A. Galal, and D. A. Dave, "Dynamic, Context-Aware Cross-Layer Orchestration of Containerized Applications," *IEEE Access*, vol. 11, pp. 93129–93150, 2023, doi: 10.1109/ACCESS.2023.3307026.
[94] J. Cheng, D. T. Nguyen, and V. K. Bhargava, "Resilient Edge Service Placement Under Demand and Node Failure Uncertainties," *IEEE Transactions on Network and Service Management*, vol. 21, no. 1, pp. 558–573, Feb. 2024, doi: 10.1109/TNSM.2023.3290137.
[95] S. Singh and D. P. Vidyarthi, "An integrated approach of ML-metaheuristics for secure service placement in fog-cloud ecosystem," *Internet of Things*, vol. 22, p. 100817, Jul. 2023, doi: 10.1016/j.iot.2023.100817.
[96] H. Chouat, I. Abbassi, M. Graiet, and M. Südholt, "Adaptive configuration of IoT applications in the fog infrastructure," *Computing*, vol. 105, no. 12, pp. 2747–2772, Dec. 2023, doi: 10.1007/s00607-023-01191-9.
[97] M. Zare, Y. E. Sola, and H. Hasanpour, "Imperialist competitive based approach for efficient deployment of IoT services in fog computing," *Cluster Comput*, Mar. 2023, doi: 10.1007/s10586-023-03985-0.
[98] S. Amjad *et al.*, "Orchestration and Management of Adaptive IoT-Centric Distributed Applications," *IEEE Internet of Things Journal*, vol. 11, no. 3, pp. 3779–3791, Feb. 2024, doi: 10.1109/JIOT.2023.3306238.
[99] I. S. M. Isa, T. E. H. El-Gorashi, M. O. I. Musa, and J. M. H. Elmirghani, "Resilient Energy Efficient IoT Infrastructure with Server and Network Protection for Healthcare Monitoring Applications," *IEEE Access*, pp. 1–1, 2024, doi: 10.1109/ACCESS.2024.3352024.
[100] E. Barker, "Nist special publication 800-57 part 1 revision 4, recommendation for key management part 1: General," *NIST*, 2016.
[101] H. Langhnoja and P. Joshiyara, "Multi-Objective Based Integrated Task Scheduling In Cloud Computing," Jun. 2019, pp. 1306–1311. doi: 10.1109/ICECA.2019.8821912.
[102] B. Yu, S. Wu, Z. Jiao, and Y. Shang, "Multi-Objective Optimization Design of an Electrohydrostatic Actuator Based on a Particle Swarm Optimization Algorithm and an Analytic Hierarchy Process," *Energies*, vol. 11, no. 9, 2018, doi: 10.3390/en11092426.
[103] T. Yang, Z. Huang, H. Pen, and Y. Zhang, "Optimal Planning of Communication System of CPS for Distribution Network," *Journal of Sensors*, vol. 2017, p. 9303989, Mar. 2017, doi: 10.1155/2017/9303989.
[104] Q.-K. Pan, M. Fatih Tasgetiren, and Y.-C. Liang, "A Discrete Particle Swarm Optimization Algorithm for the No-Wait Flowshop Scheduling Problem," *Comput. Oper. Res.*, vol. 35, no. 9, pp. 2807–2839, Sep. 2008, doi: 10.1016/j.cor.2006.12.030.
[105] A. Khaira and R. K. Dwivedi, "A State of the Art Review of Analytical Hierarchy Process," *Materials Today: Proceedings*, vol. 5, no. 2, Part 1, pp. 4029–4035, 2018, doi: https://doi.org/10.1016/j.matpr.2017.11.663.
[106] B. Z. Wang, X. Deng, W. C. Ye, and H. F. Wei, "Study on Discrete Particle Swarm Optimization Algorithm," in *Advances in Manufacturing Technology*, in Applied

Mechanics and Materials, vol. 220. Trans Tech Publications Ltd, 2012, pp. 1787–1794. doi: 10.4028/www.scientific.net/AMM.220-223.1787.
[107] S. Strasser, R. Goodman, J. Sheppard, and S. Butcher, "A New Discrete Particle Swarm Optimization Algorithm," in *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, in GECCO '16. New York, NY, USA: Association for Computing Machinery, 2016, pp. 53–60. doi: 10.1145/2908812.2908935.
[108] L. Wang, W. Ye, X. Fu, and M. I. Menhas, "A Modified Multi-objective Binary Particle Swarm Optimization Algorithm," in *Advances in Swarm Intelligence*, Y. Tan, Y. Shi, Y. Chai, and G. Wang, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 41–48.
[109] C. A. C. Coello, G. T. Pulido, and M. S. Lechuga, "Handling multiple objectives with particle swarm optimization," *IEEE Transactions on Evolutionary Computation*, vol. 8, no. 3, pp. 256–279, 2004, doi: 10.1109/TEVC.2004.826067.
[110] T. L. Saaty and L. G. Vargas, "The Seven Pillars of the Analytic Hierarchy Process," in *Models, Methods, Concepts & Applications of the Analytic Hierarchy Process*, Boston, MA: Springer US, 2001, pp. 27–46. doi: 10.1007/978-1-4615-1665-1_2.
[111] M. Higgins and H. Benaroya, "Utilizing the Analytical Hierarchy Process to determine the optimal lunar habitat configuration," *Acta Astronautica*, vol. 173, pp. 145–154, 2020, doi: https://doi.org/10.1016/j.actaastro.2020.04.012.
[112] O. Cheikhrouhou, A. Koubâa, and A. Zaard, "Analytical Hierarchy Process based Multi-objective Multiple Traveling Salesman Problem," in *2016 International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, May 2016, pp. 130–136. doi: 10.1109/ICARSC.2016.26.
[113] S.-J. Chang and T.-H. S. Li, "Design and Implementation of Fuzzy Parallel-Parking Control for a Car-Type Mobile Robot," *Journal of Intelligent and Robotic Systems*, vol. 34, no. 2, pp. 175–194, Jun. 2002, doi: 10.1023/A:1015664327686.
[114] M. Suárez-Albela, T. M. Fernández-Caramés, P. Fraga-Lamas, and L. Castedo, "A Practical Evaluation of a High-Security Energy-Efficient Gateway for IoT Fog Computing Applications," *Sensors*, vol. 17, no. 9, p. 1978, 2017.
[115] M. Aazam and E.-N. Huh, "Dynamic resource provisioning through Fog micro datacenter," in *2015 IEEE International Conference on Pervasive Computing and Communication Workshops (PerCom Workshops)*, 2015, pp. 105–110. doi: 10.1109/PERCOMW.2015.7134002.
[116] A. Venčkauskas, N. Morkevicius, V. Jukavičius, R. Damaševičius, J. Toldinas, and Š. Grigaliūnas, "An Edge-Fog Secure Self-Authenticable Data Transfer Protocol," *Sensors*, vol. 19, no. 16, p. 3612, 2019.
[117] M. Suárez-Albela, P. Fraga-Lamas, and T. M. Fernández-Caramés, "A Practical Evaluation on RSA and ECC-Based Cipher Suites for IoT High-Security Energy-Efficient Fog and Mist Computing Devices," *Sensors*, vol. 18, no. 11, 2018, doi: 10.3390/s18113868.
[118] A. A. Mutlag *et al.*, "Multi-Agent Systems in Fog–Cloud Computing for Critical Healthcare Task Management Model (CHTM) Used for ECG Monitoring," *Sensors*, vol. 21, no. 20, Art. no. 20, Jan. 2021, doi: 10.3390/s21206923.
[119] M. Jain, V. Saihjpal, N. Singh, and S. B. Singh, "An Overview of Variants and Advancements of PSO Algorithm," *Applied Sciences*, vol. 12, no. 17, Art. no. 17, Jan. 2022, doi: 10.3390/app12178392.
[120] D. S. Costa, H. S. Mamede, and M. M. da Silva, "A method for selecting processes for automation with AHP and TOPSIS," *Heliyon*, vol. 9, no. 3, Mar. 2023, doi: 10.1016/j.heliyon.2023.e13683.

[121] N. Šatkauskas and A. Venčkauskas, “Multi-Agent Dynamic Fog Service Placement Approach,” *Future Internet*, vol. 16, no. 7, Art. no. 7, Jul. 2024, doi: 10.3390/fi16070248.
[122] M. Hailan, B. Albaker, and M. Alwan, “Transformation to a smart factory using NodeMCU with Blynk platform,” *Indonesian Journal of Electrical Engineering and Computer Science*, vol. Vol 30, pp. 237–245, Jan. 2023, doi: 10.11591/ijeecs.v30.i1.pp237-245.
[123] M. A. Sudo, S. R. B. D. Santos, A. M. de Oliveira, and S. N. Givigi, “Design and Analysis of a Low-Cost Weather Monitoring System based on Standard IoT Data Protocols,” in *2022 IEEE International Systems Conference (SysCon)*, Apr. 2022, pp. 1–7. doi: 10.1109/SysCon53536.2022.9773845.
[124] “NodeMcu -- An open-source firmware based on ESP8266 wifi-soc.” Accessed: May 14, 2024. [Online]. Available: https://www.nodemcu.com/index_en.html#fr_54745c8bd775ef4b99000011
[125] “ESP8266 Wi-Fi SoC | Espressif Systems.” Accessed: May 14, 2024. [Online]. Available: https://www.espressif.com/en/products/socs/esp8266
[126] “System (Java Platform SE 8).” Accessed: Jul. 09, 2023. [Online]. Available: https://docs.oracle.com/javase/8/docs/api/java/lang/System.html
[127] A. Modiri and K. Kiasaleh, “Modification of Real-Number and Binary PSO Algorithms for Accelerated Convergence,” *IEEE Trans. Antennas Propagat.*, vol. 59, no. 1, pp. 214–224, Jan. 2011, doi: 10.1109/TAP.2010.2090460.
[128] N. Šatkauskas, A. Venčkauskas, and N. Morkevičius, “Fog Computing Service Placement Orchestrator,” in *2024 11th International Conference on Electrical and Electronics Engineering (ICEEE)*, IEEE, 2024, pp. 371–376. Accessed: Jan. 02, 2025. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/10779266/?casa_token=a6Kmdyn3VQYAAAAA:2_uoX-RObTYTEyhcVU0D4wQI5HS1_I6yvxgpwZKcorkKBjAidcMZKkrgjsk_SAZOvrKs8KzcQw
[129] K. R. Harrison, A. P. Engelbrecht, and B. M. Ombuki-Berman, “Inertia weight control strategies for particle swarm optimization,” *Swarm Intell*, vol. 10, no. 4, pp. 267–305, Dec. 2016, doi: 10.1007/s11721-016-0128-z.
[130] J. Wang, X. Wang, X. Li, and J. Yi, “A Hybrid Particle Swarm Optimization Algorithm with Dynamic Adjustment of Inertia Weight Based on a New Feature Selection Method to Optimize SVM Parameters,” *Entropy*, vol. 25, no. 3, Art. no. 3, Mar. 2023, doi: 10.3390/e25030531.

CURRICULUM VITAE

Nerijus Šatkauskas
nerijus.satkauskas@ktu.lt

Išsilavinimas:

1999 – 2003	Kėdainių Jonušo Radvilos aukštesnioji mokykla. Kvalifikacija: anglų kalbos vertėjas.
2009 – 2015	Kauno technologijos universitetas. Kvalifikacija: elektronikos inžinerijos bakalauras.
2017 – 2019	Kauno technologijos universitetas. Kvalifikacija: informacijos ir informacinių technologijų saugos magistras.
2019 – 2024	Kauno technologijos universitetas. Kvalifikacija: informatikos inžinerijos mokslo krypties doktorantūros studijos.

Profesinė patirtis:

2019 – 2021	KTU kompiuterių katedra. Pareigos: asistentas.
2021 – 2022	KTU kompiuterių katedra. Pareigos: akademinis padėjėjas.
2024 – dabar	KTU kompiuterių katedra. Pareigos: jaunesnysis asistentas.

Mokslinių interesų sritys:
Ūko kompiuterija, informacinių technologijų sauga, daiktų internetas, kompiuterių architektūra.

Mokslinės konferencijos:

1. „Orchestration Security Challenges in the Fog Computing“, „26th International Conference on Information and Software Technologies“ (ICIST 2020), Kaunas, Lietuva.
2. „Fog Computing Service Placement Orchestrator“, „11th International Conference on Electrical and Electronics Engineering“ (ICEEE 2024), Marmaris, Turkija.

LIST OF PUBLICATIONS BY THE AUTHOR OF THE THESIS

Indexed in the *Web of Science* with Impact Factor

All the results which are provided in this thesis are original. They have been presented as publications in the following scientific journals with the *Web of Science* index:

1. [S1; CH; OA] Morkevičius, Nerijus; Venčkauskas, Algimantas; **Šatkauskas, Nerijus**; Toldinas, Jevgenijus. Method for Dynamic Service Orchestration in Fog Computing // *Electronics*. Basel : MDPI. ISSN 2079-9292. 2021, vol. 10, iss. 15, art. No. 1796, p. 1–22. DOI: 10.3390/electronics10151796. [Science Citation Index Expanded (Web of Science); Scopus; DOAJ] [IF: 2.690; AIF: 4.505; IF/AIF: 0.597; Q3 (2021, InCites JCR SCIE)] [Field: T 007] [Contribution: 0.250].
2. [S1; CH; OA] **Šatkauskas, Nerijus**; Venčkauskas, Algimantas. Multi-Agent Dynamic Fog Service Placement Approach // *Future Internet*. Basel : MDPI. ISSN 1999-5903. 2024, vol. 16, iss. 7, art. No. 248, p. 1–25. DOI: 10.3390/fi16070248. [Emerging Sources Citation Index (Web of Science); Scopus] [Field: T 007] [Contribution: 0.500].

Reviewed Scientific Publications in Other International Databases

1. [P1b; CH] **Šatkauskas, Nerijus**; Venčkauskas, Algimantas; Morkevičius, Nerijus; Liutkevičius, Agnius. Orchestration Security Challenges in the Fog Computing // *Information and Software Technologies: 26th international conference*, *ICIST 2020*, Kaunas, Lithuania, October 15–17, 2020: proceedings / A. Lopata, R. Butkienė, D. Gudonienė, V. Sukackė (eds.). Cham : Springer, 2020. ISBN 9783030595050. eISBN 9783030595067. p. 196–207. (Communications in Computer and Information Science, ISSN 1865-0929, eISSN 1865-0937; Vol. 1283). DOI: 10.1007/978-3-030-59506-7_17. [Scopus] [Field: T 007] [Contribution: 0.250].
2. [P1b; DE; OA] **Šatkauskas, Nerijus**. Composition of the Information Security Methods for a Smart Environment and the Research // *CEUR Workshop Proceedings*: IVUS 2019 international conference on information technologies: proceedings of the international conference on information technologies, Kaunas, Lithuania, April 25, 2019 / edited by: Robertas Damaševičius, Tomas Krilavičius, Audrius Lopata, Dawid Połap, Marcin Woźniak. Aachen : CEUR-WS. ISSN 1613-0073. 2019, vol. 2470, p. 130–135. [Scopus] [Field: T 007] [Contribution: 1.000].

3. **Šatkauskas, N.**, Venčkauskas, A., Morkevičius, N., IEEE leidėjas., & IEEE leidėjas. (2024). Fog computing service placement orchestrator / Nerijus Šatkauskas, Algimantas Venčkauskas, Nerijus Morkevičius. In: 2024 11th International Conference on Electrical and Electronics Engineering (ICEEE 2024), Marmaris, Turkey, April 2022–2024, 2024. https://doi.org/10.1109/ICEEE62185.2024.10779266

Other Non-Reviewed Lithuanian Scientific Publications

1. [P1d; LT; OA] **Šatkauskas, Nerijus**. Composition of the Information Security Methods for a Smart Environment and the Research *// Informacinės technologijos = Information Technology*: XXIV tarpuniversitetinės tarptautinės magistrantų ir doktorantų konferencijos „Informacinė visuomenė ir universitetinės studijos“ (IVUS 2019) medžiaga, 2019 m. balandžio 25 d., Kaunas, Lietuva. Kaunas : Vytauto Didžiojo universiteto leidykla. ISSN 2029-249X. eISSN 2029-4824. 2019, p. 7-12. [Field: T 007] [Contribution: 1.000].

UDK 004.7+004.738.5](043.3)

SL344. 2025-03-10, 15,75 leidyb. apsk. l. Tiražas 14 egz. Užsakymas 30
Išleido Kauno technologijos universitetas, K. Donelaičio g. 73, 44249 Kaunas
Spausdino leidyklos „Technologija“ spaustuvė, Studentų g. 54, 51424 Kaunas