



Bulgarian Academy of Sciences
Institute of Mathematics and Informatics

Tsvetan Krasimirov Tsokov

IoT platforms and protocols

Dissertation

for conferring of academic and scientific degree "Doctor"
in professional field 4.6. "Informatics and Computer Sciences"
scientific specialty "Informatics"

Supervisor:

Assoc. Prof. Hristo Kostadinov

Sofia, 2024

Contents

Title Page	1
Abstract	1
Introduction	1
Relevancy of the problem	2
Dissertation structure	5
1 Overview of Edge/Fog distributed systems	7
2 EcoLogic IoT application	11
2.1 Overview of EcoLogic application	11
2.2 System design	12
2.3 System implementation elements	15
2.3.1 Embedded hardware system	15
2.3.2 Cloud software applications	18
2.4 Results from data analytics algorithm	20
2.4.1 Algorithm results on a dataset with known anomalies	21
2.4.2 Algorithm results on a real dataset	22
2.5 Discussion	23
3 Comparative analysis of resource allocation in Edge/Fog platforms	24
3.1 Network latency	24
3.2 Virtualization type and microservice dependencies	25
3.3 Infrastructure topology awareness	25
3.4 Dynamicity and mobility	26
4 Mixed-Integer Linear Programming (MILP) model	29
4.1 Model description and variables	29
4.1.1 Input variables for nodes	30
4.1.2 Input variables for workloads	31

4.1.3	Decision variables	32
4.2	Objectives and constraints	34
4.2.1	Maximize workload deployments (MAX_WD)	34
4.2.2	Minimize replica movements (MIN_RM)	36
4.2.3	Minimize workload end-to-end network latency (MIN_WL)	37
5	Results and discussion	44
5.1	Discussion	48
6	Conclusion	51
6.1	Future work	51
	Acknowledgments	53
	A Public contribution references	54
	B Abbreviations	55
	C Terminology	56
	Dissertation contributions	57
	Approbation of the results	59
	Presentations at scientific forums	60
	Bibliography	70

List of Figures

1.1	EcoLogic application [19] microservice architecture.	9
1.2	Movement of nodes, part of geo-distributed (multi-region) edge cluster.	10
2.1	EcoLogic application detailed architecture.	13
2.2	Raspberry Pi architecture.	16
2.3	Identified clusters in dataset with known anomalies.	21
2.4	Identified clusters in real dataset ingested from vehicle.	22
4.1	MILP model orchestration and execution.	30
5.1	Example E1 with small movement distance $\Delta x = 10\%(80km)$ per step. CS optimizes end-to-end latency by 20% in last step 20, compared to existing K8s schedulers.	46
5.2	Example E2 with big movement distance $\Delta x = 20\%(160km)$ per step. CS optimizes end-to-end latency by up to 48% starting early from step 9, compared to existing K8s schedulers.	47
5.3	Execution time of CS with MILP is much bigger (27 to 117 min), compared to existing K8s schedulers (0.06 to 0.13 sec).	49

List of Tables

3.1	Comparison between related works	28
4.1	Mixed-Integer Linear Programming (MILP) input variables for infrastructure nodes.	41
4.2	Mixed-Integer Linear Programming (MILP) input variables for workloads.	42
4.3	Mixed-Integer Linear Programming (MILP) decision variables	43
5.1	EcoLogic application Pods with Replica counts.	44
5.2	MILP input variables used in examples (E1, E2) and execution time results.	50

Abstract

The massive adoption of moveable hardware devices like sensors, actuators and microcontrollers, created a new use cases such as Internet of Things (IoT), autonomous vehicles, spacecraft computing, Virtual Reality (VR), Augmented Reality (AR), etc. Billions of devices are sensing and acting upon the vital physical world and exchange information. Their computational power is increasing, while their footprint, electricity consumption and price are reducing. The networks are undergoing a similar evolution path - their latency and throughput are improving. Not only end users, but also computing infrastructures can change their location. The applications are composed of many heterogeneous interdependent containerized microservices with specific computational and network resource requirements. Latency demanding applications have stringent Quality of Service (QoS) and Quality of Experience (QoE) necessities. Latest state of the art in Cloud/Edge/Fog infrastructures is considering only the computational (CPU, memory, storage) and network (latency, bandwidth) resources during scheduling of microservices, but in a static way. The network variability when the nodes are moving in space is not considered. This leads to increased total latency, hindering the QoS and network costs, due to not optimal traffic routes. Many researchers are focused only on a theoretical level, using simulations, which makes the adoption in the real-world practical platforms difficult, slow or even impossible. This dissertation proposes a novel technique for network-aware dynamic allocation of interdependent microservices on moving infrastructure nodes, applicable in practice. It is composed of a generic Mixed-Integer Linear Programming (MILP) optimization model and implementation in a Cloud/Edge/Fog platform. Also the dissertation provides a novel Edge-native hardware and software application, called EcoLogic, for monitoring and control of vehicle carbon emissions in smart cities. It serves as a sample application for comprehensive evaluation of the proposed solution model in practical environment. The results show reduction in the total end-to-end network latency in Cloud/Edge/Fog platforms, composed of resource-constrained mobile nodes, in comparison to the latest state of the art. The contributions can be used for future research of distributed systems with mobile parties.

Introduction

The aim of the dissertation is to provide solution for optimal management of computational (CPU, memory, storage) and network (latency, bandwidth) resources in Cloud/Edge/Fog distributed system platforms, which involve dynamic moveable infrastructure nodes and end-users. It is aware of the infrastructure topology, which means that it is monitoring and adapting to the infrastructure state continually. It is tailored towards modern microservice applications with complex interdependencies that usually are used in IoT scenarios and incorporate infrastructure clusters with constraint ARM64 hardware devices. In such dynamic environment the state of the art in Cloud/Edge/Fog platforms available in the scientific field and in the practical world are failing to manage the QoS and QoE of the latency-demanding business applications, leading to big end-to-end network latencies and degraded user experience at runtime. Proposing a solution to this problem the dissertation makes possible the support of real-time IoT applications with mobile nodes such as autonomous vehicles, virtual/augmented reality, spacecraft computing, smart city, etc.

The dissertation makes a comprehensive analysis of the available related works in the field and identifies several major parameters, which should be supported by one Cloud/Edge/Fog platform in order to handle effectively moveable infrastructure nodes and application's QoS/QoE dynamically. All works, including the proposed solution in the dissertation, are compared against them and at the end the benefits of the proposal are evaluated and highlighted.

The methodology of the dissertation is the following. The stated issue is formulated as a mathematical optimization problem and a MILP model for finding the most optimal solutions is provided. Its architecture is designed and implemented in the most popular Cloud/Edge platform used in the practice, called Kubernetes (K8s) [1]. The implementation is done in the Golang programming language. A testbed cluster containing resource-constrained devices (Raspberry Pi) is built and the implemented Edge/Fog platform is deployed on it. The testbed is emulating dynamic environment composed of geo-distributed mobile Edge/Fog nodes moving in space. Its aim is to evaluate how the proposed model reduces total end-to-end network latency of a business application in this environment. The business application used for the evaluation is also implemented as part of the dissertation. It is called EcoLogic and represents a practical edge-native IoT application. Its functionality is to measure and control carbon emissions from vehicles and is suitable for execution in smart city scenarios.

The obtained results show that the model is scheduling and moving resources on proximal infrastructure nodes by coping to the movements of the nodes. In this way the business application end-to-end network latency is kept at minimum on runtime and in continual manner. One of the most important outcomes of the dissertation is that the solution is suitable for real-world practical infrastructures with constrained ARM64 devices.

Relevancy of the problem

For the last two decades we are seeing a massive adoption of many, heterogeneous (complex and constrained), stationary and mobile hardware devices across the globe, which are connected between each other with different types of networks, forming the nowadays distributed systems such as Cloud/Edge/Fog/Swarm computing platforms. They are used not only by roughly all economic sectors, like manufacturing, transportation, agriculture, food industry, health, education, tourism, etc, but also in everyday people life, for necessity or not. Many applications are natively requiring low network latency and big throughput, especially the real-time ones, like self driving cars, AR, VR, health-related applications and others. Their QoS and QoE are very stringent. The adaptation to all these devices results to changed sensitivity and demands in humans. For example everybody is liking low latency and big throughput for the applications, which use. On the other hand the price for manufacturing of the devices and servers, for maintaining the networks, for their energy consumptions, for the software on top of the hardware and for the management and processing of the huge amount of generated data should be reduced and kept at minimum continually. The optimal management of the computational (CPU, GPU, memory, storage) and network (latency, bandwidth) resources in the latest Cloud/Edge/Fog platforms is difficult task with NP-hard complexity [2]. This problem is tackled by different approaches such as mathematical optimization (Integer-Linear Programming), artificial intelligence (reinforcement learning, deep neural networks) and heuristics-based algorithms. There is high demand for innovation and introduction of automation and digitalization in novel scenarios, which have even harder requirements and incorporate constrained devices, which move in space alone or attached to other living or inanimate objects. It seems that we as humans have reached the moment in evolution, where we possess enough technology for manufacturing of such capable hardware devices, but the optimal algorithms and software are missing in the latest state of the art in science and practice. The main existing problems of the current distributed platforms regarding their support for novel mobile scenarios are the following:

- **Infrastructure topology awareness.** The nowadays distributed systems, in the face of Cloud/Edge/Fog platforms, have to be aware of the topology of the underlying computational and network infrastructure and in the same time they have to be agnostic to it. This means that the platforms should continually monitor and manage the underlying computational resources in terms of CPU,

memory, storage and network capabilities - mainly the latency and bandwidth. At the same time the platform should not depend on the concrete type of the underlying architecture - its hardware architecture, manufacturing type, age, etc, and should support as much as possible types of hardware.

- **Virtualization type.** The effective isolation and management of resources in modern Operating Systems, is enabled by virtualization. It is mandatory for the Cloud/Edge/Fog platforms to support virtualization on the infrastructure host devices. The containerization-based approach for virtualization is easier for integration in the software stack of the platforms, compared to the Virtual Machine (VM) one, so preferably it has to be supported by the platforms. The containerization is achieved natively and in performant way in the Linux-based Operating Systems by Control Groups (cgroups) and Namespaces.
- **Application microservice dependencies support.** The business applications evolved in the last two decades from singular big monoliths to loosely-coupled small and atomic microservices, following the Cloud-native [3] and recently the Edge-native [4] software design principles. In this way they are built in an object-oriented way, preserving their encapsulation, single responsibility and communicating with abstractions and interfaces. In this way they become easier for functional extension, but their lifecycle management become harder at runtime, because their count become huge and their inter-dependencies - much more complex. On the other hand in dynamic environment, as it is in the novel scenarios with mobile infrastructure nodes and users, the management of their performance and availability in runtime become with NP-hard complexity. Current distributed platforms do not support preservation of QoS and QoE for applications with microservice dependencies in dynamic environments.
- **Dynamicity.** The dynamicity of the environment contains two parameters: dynamicity of infrastructure and dynamicity of applications. The infrastructure is dynamic in cases where devices join and leave the network due to different instabilities. Device resources can also vary over time due to hardware and network issues. The dynamicity of applications means that the application's demand or workload, which is generated by the end-users, can be dynamic. This often happens in scenarios where users are mobile or their operations vary in time. For example in scenarios where humans are following daily schedule, the application's workload will be increased in the time periods when people are going home, mainly in the afternoon and evening hours. It's crucial one platform to support dynamicity on level of infrastructure and on level of applications.
- **Mobility.** The mobility is related to level of infrastructure and to level of end-users. Both can move in space in different time, which means that the platform should continually adapt to this mobility. This can be achieved by replacement and scheduling of computational and network resources on different more optimal

infrastructure devices in order to preserve the critical application's QoS and QoE in continual manner. This problem is the biggest one, which is not supported in the latest practical Cloud/Edge/Fog platforms. They simply do not work especially with mobile infrastructure nodes, leaving the application state as it is, resulting to degraded QoS and QoE for the applications.

- **Network latency awareness.** The network latency characteristic is the most important one for maintaining the QoS and QoE of the business applications. The round trip time of a client network request to reach a server application, located in a cloud is around 40 ms over wired internet connection. There are latency measurements for round trip time over 4G and 5G wireless networks in Non Standalone (NSA) and Standalone (SA) deployments in America with the following median values, according to the Rochman et. al. survey [5]. For AT&T 4G and 5G-NSA, they are 30.5 ms and 30.7 ms, respectively. For T-Mobile 4G, 5G-SA and 5G-NSA: 44.1 ms, 48.4 ms, and 74.8 ms, respectively. For Verizon 4G and 5G-NSA: 44.1 ms and 54.4 ms, respectively. It's known that applications like AR and VR require total latency of 20 ms at maximum [6]. This round trip times can be significantly reduced by using Edge/Fog platforms, in which the server devices are placed in the proximity of end-users, in contrast to Cloud platforms, where the server is far away from users. In dynamic and mobile environments this Edge/Fog platforms should adapt to the movements of the infrastructure and end-users in order to minimize the total end-to-end latency and preserve QoS and QoE. Again this automatic adaptation to node movements for minimization of the total latency is not supported currently in practice.
- **ARM64 CPU architecture support.** The above mentioned real-time business scenarios, which incorporate moveable objects are requiring lightweight, low-footprint and constrained hardware devices, which often are connected to wireless networks and use low-power sources or batteries. In practice such constrained devices are usually with ARM CPU architecture. The modern Cloud/Edge/Fog platforms are barely supporting ARM64 CPU architecture. It's necessary to fully support the whole range of ARM-based devices (ARM32 and ARM64) in order to cover wider use cases and scenarios.
- **Evaluation type.** The above problems are barely handled in the latest Cloud/Edge/Fog platforms available in practice in the computer engineering field, despite being open source or proprietary. Some of these problems are researched in different publications from the scientific field, but mainly on a theoretical level and are evaluated by simulations, instead of real practical implementations [7]–[10]. A lot of works are using the simulators like CloudSim [11], EdgeCloudSim [12] and iFogSim [13]. Constrained devices with ARM architecture, which are important in the practical field, are not taken into account. Such theoretical models from the scientific field, which are not adequately evaluated in real-world devices and platforms will be applied into the practical field really hard, after

spending a lot of time and efforts. That's why it's necessary to be researched feasible theoretical models, which are thoroughly evaluated in a real-world environment with constrained hardware devices and Edge/Fog platforms used widely in practice.

Due to all these issues and the promising potential of the novel Edge/Fog platforms and distributed systems algorithms, the aim of the dissertation is to analyze and solve the above problems not only on a theoretical and simulation level, but in practice - on a real-world hardware clusters, using the latest technology stack. This will bring improvements in the applications end-to-end network latency in practical Edge/Fog infrastructures with moving nodes.

Dissertation structure

The dissertation is structured in introduction and 6 chapters. It contains 76 pages, 10 figures, 6 tables, 86 references and 3 appendices. There is 1 research article and 1 conference paper published in international journals, related to the dissertation. The results are presented in 3 scientific forums.

In Chapter 1 an overview of the Edge/Fog computing platforms is made. The evolution of the Cloud to Edge/Fog systems is described. Important recent developments and future directions of the distributed systems used in practice are highlighted. The main problems for supporting novel real-time application scenarios are stated. Also major features and parameters of the platforms necessary for solving the stated problems are identified. These features are used in the next chapters for comparative analysis and system design.

In Chapter 2 the EcoLogic IoT application is described. It has functionality for monitoring and control of carbon emissions from vehicles in smart city scenarios. It serves as a real-world practical edge-native application used for evaluation of the proposed solution for container scheduling in Edge/Fog platforms, in next chapters. EcoLogic microservice architecture, algorithms, publicly available implementations and results are presented.

In Chapter 3 a comparative analysis is made for some major related results available in the scientific field. These results are summarized and compared against the identified platform parameters, towards support of the stated problem for dynamic resource allocation in Edge/Fog platforms with moveable nodes. The chapter highlights the importance of the problem topic in the research community and in practice.

In Chapter 4 the developed MILP model is shown. It provides optimal solution for the problem of dynamic container allocation in Cloud/Edge/Fog platforms. Its variables, constraints and objective functions are defined and explained. Also the overall model execution and algorithms are described.

In Chapter 5 two examples and an execution time result are shown, to demonstrate how the described MILP model handles the identified major features of Edge/Fog

computing platforms in a realistic mobile environment. The examples are conducted in a testbed and are aiming to evaluate realistic workload represented by the EcoLogic sample IoT application. The results show reduction in the total end-to-end network latency compared to other state of the art solutions.

In Chapter 6 conclusion remarks and open problems are given, related to the presented research problem, methodology, solution and results, regarding the Edge/Fog platform and EcoLogic application. Also the detailed dissertation contributions are listed. Additionally, the approbations of the results, presentations on scientific forums and publicly shared implementations and resources are attached.

Chapter 1

Overview of Edge/Fog distributed systems

The development of hardware in the last years brought a massive adoption of hardware devices with computing and network capabilities into many economy sectors like electronics, machine industry, transport, agriculture, healthcare and others. This massive adoption of hardware devices enables many processes which were previously executed solely by humans or electro-mechanical machines to be assisted or fully-replaced by smart IoT devices. Due to their limited resources they cannot execute complex computing tasks on big amounts of data, so most frequently in practice they execute only simple computing tasks, filter data and delegate the complex tasks with filtered data to Cloud platforms [14] via exposed software services. Cloud computing offers almost unlimited computing resources, which are provided by centralized infrastructure nodes located in small number of data centers around the globe. However the IoT devices are frequently located far away from the Cloud data centers and the delivery of data is introducing big overhead and operational efforts on the long-distance networks. This leads to reduced network bandwidth and increased latency, especially when the devices are moving. It is known that low network latency and mobility are crucial for large range of applications in order to support certain level of QoS and QoE [15]–[18]. Such kinds of applications are for example virtual and augmented reality, autonomous driving vehicles and traffic emissions control in smart cities [19]. These requirements are addressed in the novel Edge and Fog computing platforms, which extend the computational resources of Cloud platforms with additional infrastructure nodes located on the Edge layer of the network and closer to the IoT devices and end-users [20]–[23], providing lower latency and bigger bandwidth.

Major Cloud providers are building their Edge computing platforms, including Multi-access Edge Computing (MEC) [24], [25], like Azure Edge Zone [26], Google Distributed Cloud [27], AWS Local Zones [28] and Alibaba Link IoT Edge [29], but they are at early stage due to the difficulties related to less computational resources and geo-distribution of Edge nodes [30]. On the other hand major content providers like Google [31] and Facebook [32] have Edge infrastructure based on Content Delivery

Network (CDN) to deliver their contents to end users in more effective way. But CDNs are used mainly only for caching (storage) and not for processing tasks. However there is a tendency by some major CDN providers like Cloudflare [33], Akamai [34] and Fastly [35] to begin to provide not only caching, but also computational processing on Edge by leveraging their global CDN infrastructure [36]. Macrometa [37] provide innovative highly-distributed global database and possibility to execute computational tasks like stream processing on Edge nodes, using the Akamai infrastructure, enabling users to build global stateful applications.

Cloud platforms rely on virtualization techniques to support resource utilization and multi-tenancy. Container-based virtualization has become the standard and most used type in Cloud platforms in contrast to hypervisor-based type, because it is more lightweight and have better support for microservice architecture [38]. On the hardware level, ARM CPU architecture [39] is becoming increasingly used for servers, but its virtualization support is more limited than the prevalent x86 architecture [40]. It's known that Edge network layer can be composed of nodes with low-power and ARM CPU architecture [41]. From the stated above follows that constrained devices with ARM CPU architecture and container-based virtualization can be widely used in Fog computing platforms and thus being highly important research topic.

On the other hand microservice architecture become widely adopted and has established principles for building software applications suitable to run in Cloud platforms - Cloud-native [3], [42] principles and Twelve-factor App methodology [43]. Right now a new novel principles are emerging in practice called Edge-native [4], [44]. Edge-native principles are an extension and evolution of the Cloud-native ones. They describe how to build a set of microservices in the most isolated, scalable and resilient way such that they can run reliably in Edge environment. Edge-native principles should be applied for Fog platforms. It is known that microservices can form a graph of interconnected dependencies, which communicate between each other, called Service Function Chains (SFCs). For example EcoLogic [19] is a typical Edge-native application for monitoring and control of vehicle emissions in cities. It is composed of four interdependent microservices depicted in Fig. 1.1. The Database microservice (P1) contains data related to vehicle parameters and emissions, while the Backend microservice (P2) provides an interface and analytics of this data. The VehicleAgent microservice (P3) instances are running on a hardware nodes located in all registered vehicles. It collects low-level vehicle parameters and sends them to the Backend microservice (P2). The Frontend microservice (P4) serves a web-based Application Programming Interface (API) and user interface. The Database (P1), Backend (P2) and Frontend (P4) microservice instances can run on stationary or non-stationary nodes. All microservices support multiple number of replicas running on different machines. In such dynamic environments the infrastructure nodes are forming a geo-distributed cluster and can move in time and space. An example cluster with nodes, which change their location in 2 different states in time is shown in Fig. 1.2. Deploying dependent microservices on distant nodes without latency awareness can impact the application's response time and overall QoS and QoE. If the nodes are mobile, the initial deployment of dependent

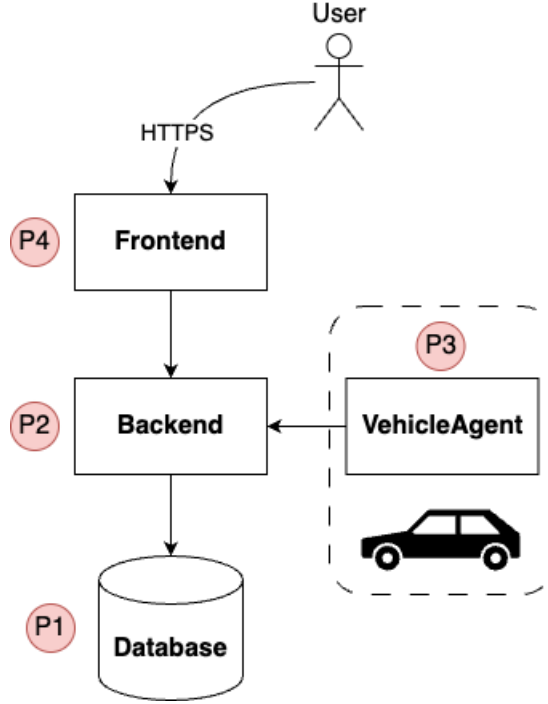


Figure 1.1: EcoLogic application [19] microservice architecture.

microservices can be on near nodes, but when the nodes change their location and move away from each other, the end-to-end application response time will be increased and overall QoS/QoE - degraded. It is necessary the dependent microservices to be re-deployed on different closer nodes if such exist and thus dynamically adapting to the mobility of nodes in time and space. The platform should keep track of the underlying infrastructure topology composed of heterogenous nodes with different resources and varying links in terms of network latency and bandwidth.

Previous research on latency-aware scheduling emphasize mostly on theoretical definitions (e.g. [45]–[48]) or heuristic-based algorithms (e.g. [49], [50]) that usually are assessed on simulators like CloudSim [11], EdgeCloudSim [12] and iFogSim [13]. Often if the evaluation is based on real devices they are with big resource capacity and have x86 or x64 CPU architectures. Real world constrained devices with ARM architecture are not considered. Additionally the devices are stationary in space, lacking mobility. In the domain of latency-demanding applications it's important for one Fog platform to support allocation of SFCs on mobile Edge nodes which change their location in time and space.

According to the above exposition several major features of Edge/Fog computing platforms are identified such as:

- **Infrastructure topology awareness**
- **Virtualization type**

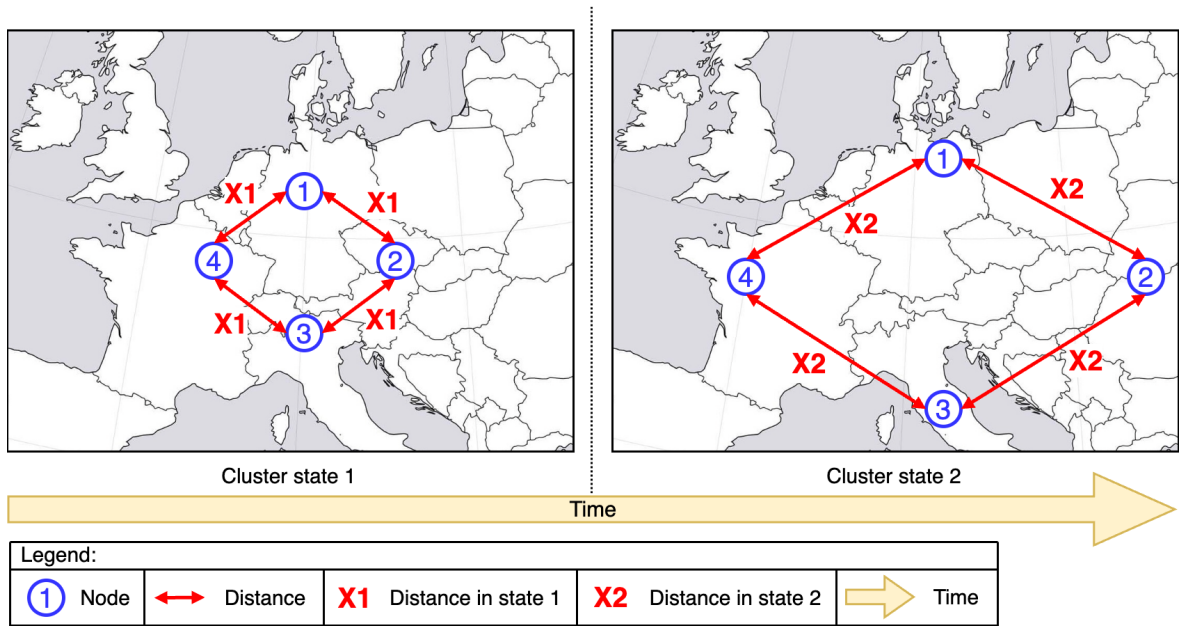


Figure 1.2: Movement of nodes, part of geo-distributed (multi-region) edge cluster.

- Application microservice dependencies support
- Dynamicity
- Mobility
- Network latency awareness
- ARM64 CPU architecture support
- Evaluation type

These features will be used as a pillars for comparison with other related works and in system design and evaluation.

In the next Chapter 2 the EcoLogic application will be described, representing a sample edge-native IoT application, suitable to run in real-world practical smart city environments and used for solution validation in the following chapters.

Chapter 2

EcoLogic IoT application

Nowadays, the smart physical devices are part of the everyday people life and generate huge amount of data, which drives applications and services on top of them, establishing the IoT ecosystem [51]. Applications are developed to ingest, store and analyze large amounts of data generated by the IoT devices in all economic sectors such as transportation, agriculture, health and education. According to CISCO Internet Business Solutions Group (CISCO IBSG), the total number of interconnected devices by 2015 was 25 billions and it's expected to be 50 billion by 2020 [52]. For the same time period it is expected 5.8 billion IoT endpoints and global economic revenue from endpoint electronics to total 389 billion US dollars.

One of the most impacted sectors by the IoT is the automotive industry. Recently, tens of millions of cars are said to be connected to the Internet and their number is expected to become hundreds of millions in the near future.

2.1 Overview of EcoLogic application

According to a global industry analysis, the number of connected vehicles will become 125 million by 2022. In order to enable such a massive amount of connected vehicles to communicate with other IoT endpoints in real time, new network bridges and protocols are developed [53]. At the same time, mobile network technology is recognized to have considerable potential to enable carbon emissions reduction across a wide range of sectors. Currently, 70% of the carbon savings come from the use of machine-to-machine (M2M) technologies, according to Global e-Sustainability Initiative. The survey data shows that 68% of smartphone users are willing to adopt behaviors that could result in even more future reductions to personal carbon emissions. According to another study the power consumption of the radio access networks (RAN) and the production of mobile devices are major contributors in the carbon emissions [54]. There it is concluded that the usage of green technologies to reduce the power consumption offer big potential for reduction of the emissions. IoT is pointed as a key lever to reduce the carbon emissions.

The current hardware and software solutions for tracking vehicles give evidence for the efforts of using IoT technologies in automotive industry. Geotab provides a service for monitoring and analysis of vehicles using integrated hardware module that sends data to private cloud. The hardware module is connected to the onboard diagnostic bus of the vehicle and collects data about fuel consumption, traveled distance and other parameters. The data analysis, which is made in the cloud allows identification of vehicles with not optimal fuel consumption. Unfortunately, Geotab solution does not detect increased rate of carbon emissions and provide control over vehicle’s parameters. Madgetech is a data logger, which provides functionality for regular monitoring of carbon dioxide levels (Data Loggers). It measures the carbon emissions by exhaust gas sensors in vehicles and sends data to private cloud. The measured data is visualized by mobile application, but analysis and control of the carbon emissions are not supported. In addition to these existing solutions, there are mandatory emissions inspection and certification procedures in many countries. But these procedures usually are done only one or several times per year, which is a small frequency and can not guarantee that the vehicles are working with optimal emissions for big period of time.

Inspired by the low-carbon roadmap of European union and the grate potential provided by the IoT technologies for reducing the carbon emissions, in this dissertation a solution for real-time monitoring and detection of rising levels of carbon emissions from vehicles, called EcoLogic, is proposed. It is used also for evaluation of the presented framework for resource scheduling in dynamic Cloud/Edge/Fog platforms with moveable infrastructure nodes, described in next chapters. The proposed EcoLogic application includes hardware module, which collects sensor data related to vehicle’s carbon emissions such as air pressure, air temperature and fuel mixture. The data is transferred to a cloud-based applications, where it’s stored and analyzed. The results from the analysis are used to control the carbon emissions through driver notifications and vehicle’s power limitations. The obtained results, resources and source code of the main software components of the solution are publicly available and can be accessed and downloaded from the following locations, described in Appendix A. The repositories contain information on how to setup the projects and deploy them on hardware device and server or in the cloud. The source code and hardware modules are with open-source licenses, because the purpose of the solution is to be freely reviewed, used and extended by everyone who is interested to track and reduce emissions at global scale. Additionally the system can be integrated with the countries tax system supporting drivers with small emissions footprint to pay smaller taxes.

2.2 System design

This section presents the architecture of EcoLogic system with its components, communication paths and protocols.

The EcoLogic system is composed of two big parts: edge-native hardware modules and cloud microservice applications. The hardware modules are located in vehicles

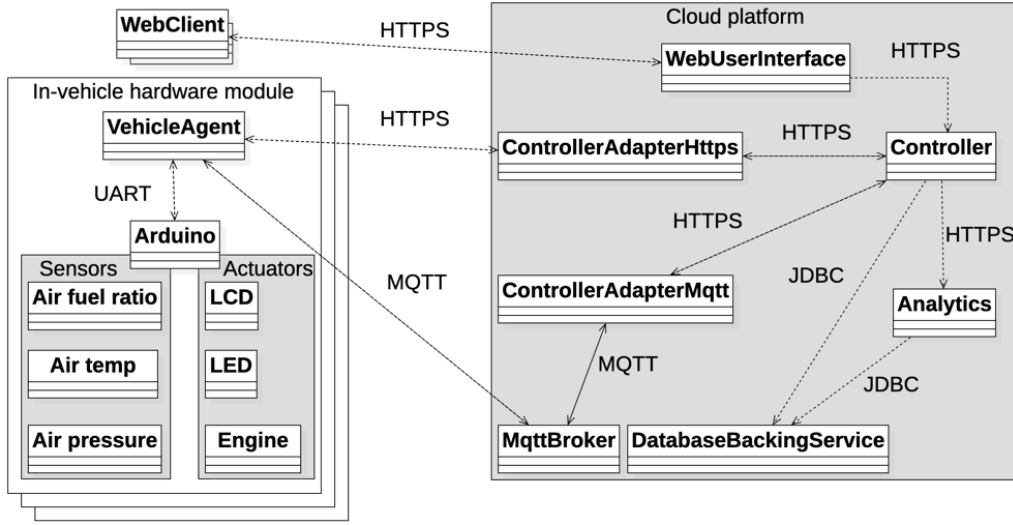


Figure 2.1: EcoLogic application detailed architecture.

and the microservice applications are deployed on a cloud platform. The architecture of the system is shown in Fig. 2.1. The history of physical computing technology showed us that each physical computing system contains at least one hardware device, but nowadays the IoT paradigm, which extends the physical computing technology, is showing us that except hardware devices each IoT system should use also Cloud or Fog platform, which enables massive scale, reliability and efficiency. It becomes a standard for IoT systems to incorporate Cloud and Fog computing platforms in its architectures [55] as main components that give a lot of benefits like enormous computing power, high availability, disaster recovery, storage, scalability and near real-time speeds.

The hardware module of the system has sensors and measures several physical parameters. It can also extract parameters from the onboard diagnostic system of the vehicle. The data is sent to applications in the cloud platform. The measured physical parameters are:

- *Air/fuel ratio*, which is measured by lambda sonde sensor, which is located into the exhaust system of the vehicle.
- *Absolute pressure of the air* that is consumed by the engine, which is measured by sensor located in the air intake manifold of the vehicle.
- *Temperature of the air* that is consumed by the engine, which is measured by sensor located in the air intake manifold of the vehicle.

The cloud applications are implemented as microservices, which are designed in a platform agnostic way in order to have the possibility for deployment on different cloud platforms. The cloud applications store data in a relational database, which is represented by backing service from the cloud platform. They process the incoming data, store it into the database and analyze it. The hardware modules communicate with the cloud with wireless network via Hypertext Transfer Protocol Secure (HTTPS) or Mes-

sage Queue Telemetry Transport Protocol (MQTT). The following physical parameters are calculated on top of the incoming sensor data:

- Mass of the consumed air by the engine.
- Mass of the consumed fuel by the engine.
- Mass of the carbon dioxide emissions, exposed into the atmosphere.

The database contains all measured and calculated physical parameters. A cloud Analytics application executes an anomaly detection on the streamed data and outlines vehicles that have not optimal amount of carbon dioxide emissions or system failures. Clustering analysis is made in order to detect anomalies. The hardware module is notified when the vehicle is detected by the system as an anomaly, with suboptimal amount of emissions. Then the hardware actuator is started automatically to reduce the amount of emissions. This comprises a feedback control loop. In this way the system monitors and controls the amount of carbon dioxide emissions in the atmosphere in real time. The hardware modules are equipped with three actuators:

- *Liquid crystal display (LCD)*, which visualize the measured and calculated physical parameters to the driver.
- *Light-emitting diode (LED)*, which indicates to the driver that the amount of carbon dioxide emissions is not optimal or there is a system failure (not optimal parameters).
- *Actuator*, which controls the amount of injected fuel in the engine and regulates the amount of emissions.

Currently, only the display and LED actuator are implemented in EcoLogic. The purpose of the LED actuator is to notify the driver to intentionally reduce the acceleration and change the driving behavior, which leads to reduction of the amount of burned fuel and emissions.

The cloud applications provide web user interface, on the base of HTML5, JavaScript and CSS resources. Its endpoint is publicly available and accessible by clients via HTTPS protocol.

The user management of the system is composed of two roles: driver and operator. The lifecycle of the system is the following:

- Dealer sells a hardware module to a driver.
- The driver installs the hardware module into vehicle. This step can also be accomplished by an authorized service.
- The driver registers the vehicle with the hardware module and sensors via the web user interface. All components have unique identifiers.
- Drivers are authorized to monitor and control their own registered vehicles.
- Operators are authorized to monitor and control all registered vehicles by regions.
- Each driver gets score points proportional to the amount of carbon dioxide emissions exposed in the atmosphere from their vehicles.
- The score points can be integrated with tax systems of countries and city halls. In this way the taxes can be reduced proportionally to the score points. Drivers can also participate in greenhouse gas trading schemes with their score points.

Important aspect of the design of the system is its security. Security is major problem in all IoT applications, because of the big number of heterogenous devices, the data, which is distributed across many locations and the need for high-speed communication. New protocols and schemes that incorporate symmetric and asymmetric cryptography are developed [56] in order to resolve these problems. The current system uses TLS v1.2 (TLS protocol) with its built-in asymmetric and symmetric cryptography for the network communication between components.

2.3 System implementation elements

This section outlines the main components of EcoLogic system. First, the hardware module with its components and the algorithm for calculation of carbon emissions are presented. Later, the cloud applications are described and the algorithm for data analysis.

2.3.1 Embedded hardware system

The embedded hardware system contains two embedded subsystems: Arduino Uno and Raspberry Pi B+. The Arduino operates on the low-level hardware sensors and actuators in the vehicles, while the Raspberry Pi works on higher level. It aggregates the data from the low-level hardware and communicates with the cloud platform in bidirectional way.

Arduino embedded hardware system

The Arduino Uno embedded system has the following capabilities:

- Measure physical parameters of internal combustion engine.
- Visualize the measured parameters on 4x16 liquid crystal display.
- Control of actuator (light emitting diode).
- Communication with Raspberry Pi embedded system.

The physical parameters are measured in two ways: by sensors or extracted from the onboard diagnostic system (OBD2), which is provided by the electronic control module of the vehicle. If the onboard diagnostic interface exposes the necessary parameters in the Application Programming Interface (API), no additional sensors will be installed in the vehicle. If the onboard diagnostic interface API does not expose the necessary parameters, additional sensors, which measure these parameters, will be integrated. There is also a possibility to reuse the already existing default sensors of the vehicle. In this way the hardware module is flexible and can be installed on huge amount of vehicles.

The Arduino embedded system has deployed application in its local storage and executes it. The application is implemented in the C++ programming language. It is publicly available and can be downloaded from the following location A. 2. The

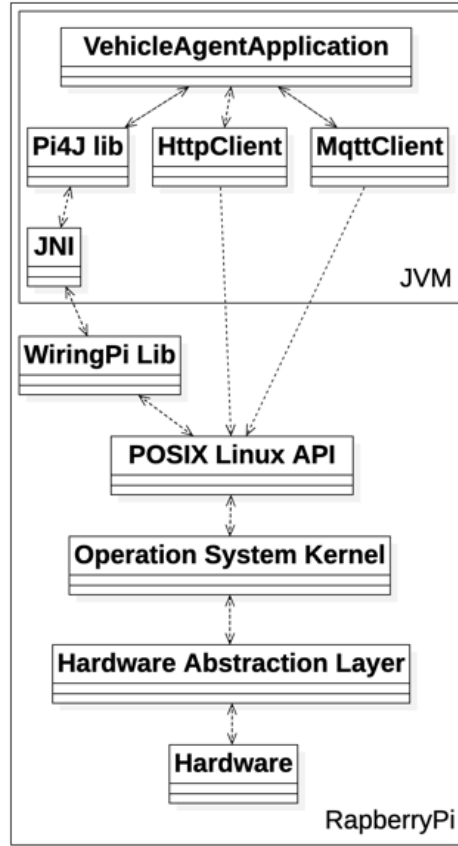


Figure 2.2: Raspberry Pi architecture.

repository from the link contains information for the build, deployment and execution on local Arduino-based hardware module. The application consumes Wiring library, which is part of the Arduino Software Development Kit (SDK). The Wiring library executes on the concrete Arduino microcontroller via drivers. Currently, the Arduino Uno hardware is used, which has Microchip ATmega328 microcontroller based on RISC architecture. This hardware module executes a critical calculations from physical sensors in real time. This is the reason why it's a separate module with own independent computational resources and with software application written in fast low-level programming language like C++.

Raspberry Pi embedded hardware system

The Raspberry Pi B+ embedded system delegates the communication between the Arduino embedded system and the applications in the cloud platform. Its architecture is shown in Fig. 2.2.

The communication between the Raspberry Pi B+ embedded system and the Ar-

duino embedded system is via serial Universal Asynchronous Receiver and Transmitter (UART) protocol with a custom application messaging protocol on top of it. The Raspberry Pi B+ embedded system is connected to the cloud platform via 4G/5G broadband cellular network. The measured physical parameters by the Arduino embedded system are ingested into the Raspberry Pi B+ embedded system, which caches them in a local cache storage for further processing and sends them to the cloud platform. It sends the data to the cloud platform via HTTPS or MQTT protocol. The Adapter cloud-application is the facade, which is the only endpoint seen by the Raspberry Pi B+ embedded system. When it sends the data to the cloud platform, it receives response that contains information about the state of the vehicle, including the amount of the carbon dioxide emissions. There are two possible states: optimal (eco) and not optimal (not eco). The Raspberry Pi module notifies the Arduino embedded system once the emissions are not optimal. Then the Arduino embedded system activates the low-level hardware actuator in order to reduce the quantity of emissions. The main component inside the Raspberry Pi embedded hardware is the System on a Chip (SoC), which is using ARM architecture and Linux based operation system, currently Raspbian. It executes the VehicleAgent application, which is implemented in the Java programming language and it runs on top of a Java Virtual Machine. The VehicleAgent application is publicly available and can be downloaded from the following location [A. 3](#). The repository from the link contains information on how to build the application locally, how to deploy it on any Raspberry Pi-based hardware module and how to execute it.

The VehicleAgent application depends on the Pi4J and WiringPi libraries, which support the serial communication between the two embedded systems. An USB WiFi adapter is used for the 802.11n communication between the Raspberry Pi embedded hardware and a hotspot, which provides the 4G/5G broadband cellular network to the cloud platform. The adapter is connected to one of the USB ports of the Raspberry Pi. The communication with the cloud platform can be achieved by several application layer protocols: HTTPS or MQTT. The decision which protocol will be used is taken according to the supported protocol by the cloud platform. The modular architecture of the application enables easy extension with other application layer protocols. Because the communication network to the cloud platform is constrained it is expected to exist glitches and that's why the application caches the latest data in the local cache storage. After successful reestablishment of the connection with the cloud platform, the application retries to sent the locally cached data. The application can be configured by configuration file (config.xml), which is located into the local file system storage. It contains unique identifiers for the vehicle, sensors and type of the communication with the cloud platform (HTTPS, MQTT, etc). The driver or operator, who registers the vehicle into the system, should populate the configuration file and save it.

This hardware module serves as a communication proxy and need to have capabilities to work with a lot of communication networks and protocols. This is the reason why it's a separate hardware module, which work with many hardware adapters or antennas supporting different communication networks. The software application is written in high-level programming language like Java, because it needs to work with

many communication protocols and serialize data in different formats.

2.3.2 Cloud software applications

The EcoLogic is composed of several Edge-native microservice applications deployed in the cloud, namely Controller application, Adapters applications, Web user interface and Analytics application. Each application is described in the following subsections.

Controller application

The Controller application is the main cloud application, which manages all vehicles with their hardware modules and sensors, make calculations, persists data into database, executes artificial intelligence algorithms on the data and provides HTTP REST API. It is implemented using the Java Enterprise Edition programming language using JPA and Apache CXF services framework. The Controller application is publicly available and can be downloaded from the following location [A. 4](#). The repository from the link contains information on how to build the application locally, how to deploy it on a web container in any cloud platform or local server and how to execute it. The application has the following functionality:

- Compose a data model, which has the following relations: users, which have vehicles, which have sensors, which have measurements of physical parameters.
- Orchestrates the lifecycle of all users, vehicles, sensors and measurements.
- Persists the data into a relational database, which is provided as a backing service exposed by the cloud platform. The application is database-agnostic, which means that it can work with any relational database, which provides Java connectivity. If it does not provide Java connectivity, an adapter can be used. The database is composed of four tables: User, Vehicle, Sensor and Measurement.
- Calculates the total mass of the carbon dioxide emissions disposed into the atmosphere by vehicles.
- Manages the state of each vehicle: optimal (eco) state and not optimal (not eco) state. This is a simple state machine with two states.
- Calls the API of an Analytics application, which executes clustering analysis and anomaly detection. The purpose is to find vehicles, which have not optimal amount of carbon dioxide emissions per region.
- Provides HTTP REST API which is used by the Adapter applications and web user interface.

Adapter application

The data coming from the vehicle hardware modules is intercepted and adapted by the Adapter cloud applications. Then it is routed to the Controller application. The Adapter applications are implemented using the Java programming language. Currently there are two types of Adapter applications that handle HTTPS and MQTT

network protocols: ControllerAdapterHttps and ControllerAdapterMqtt, respectively. ControllerAdapterMqtt application communicates with MQTT broker. It is broker independent, so: Mosquitto, HiveMQ, Mosca or other type of MQTT broker can be used. The MQTT broker and the Adapter applications have publicly available URL endpoints, which are called by the vehicle hardware modules. The traffic is routed by the cloud platform to the concrete application depending on the application layer protocol that is used. The traffic is routed to the ControllerAdapterHttps application if HTTPS protocol is used. The traffic is routed to the MQTT broker if MQTT protocol is used. This routing behavior of the cloud platform is based on TCP routing mechanism. TCP routing enables cloud platforms to support applications, which communicate with different non-HTTP protocols.

The flow of the MQTT traffic is the following:

1. On creation of new vehicle, the Controller application registers two topics with names *vehicles/{id}/sensors/{id}/measurements* and *vehicles/{id}/state*. The ControllerAdapterMqtt application subscribes for that topics.
2. The concrete hardware module also subscribes to both topics and start to publish new measurements to *vehicles/{id}/sensors/{id}/measurements* topic.
3. On receiving of new message with measurement by the ControllerAddapterMqtt application, it adapts the measurement and sends it to the Controller application via the HTTP protocol.
4. The ControllerAddapterMqtt application receives the state of the concrete vehicle and publishes it to the *vehicles/{id}/state* topic.
5. The concrete hardware module is subscribed to the state topic and receives a response with the state of the vehicle, whether it is in optimal state or not. In this way vehicle is notified.

This flow represents how ControllerAddapterMqtt application adapts the data from MQTT to HTTP in both directions.

Web user interface

A web server serves static HTML5, JavaScript and CSS resources, which are assembling the web user interface. The web resources do not contain any back-end logic, but only front-end code, which assembles a responsive and user-friendly interface. This functionality for serving of static web resources is lightweight and it's supported by the most cloud platform providers. The concrete web user interface in the EcoLogic is based on the open-source JavaScript front-end web application framework OpenUI5. It makes many simultaneous Asynchronous JavaScript and XML (AJAX) requests to the HTTP

REST API endpoint exposed by the Controller application. Public access is provided and the drivers and operators of the system are using it. The web user interface and the Controller application have different origins, because they have different domain names. The problem of the same-origin policy (OpenUI5), which postulates that one web application can access web resources from the same origin or only permitted web resources from another origin is resolved by most of the cloud platforms, usually by tokens.

The web user interface is based on the model-view-controller architecture and supports the following capabilities:

- Handles the whole lifecycle of the users, vehicles, sensors and measurements: supports create, read, update, and delete operations.
- Display all historic and live measured parameters in real-time.
- Static manual control of the state of all vehicles by appropriate user interface controls, which leads to control of the vehicle's actuator. This is achieved with manually configured threshold.
- Automatic dynamic control of the state of the vehicles, which is based on the value of emissions or parameters that are not optimal per region.
- Visualization of all vehicles with not optimal emissions or parameters - anomalies (outliers).

Analytics application

The Analytics application executes a clustering analysis algorithm on the stored data. The algorithm takes two parameters as input: engine capacity of the vehicles and total mass of carbon dioxide emissions exposed in the atmosphere. In this way it partitions vehicles that have similar engine capacity and emissions amount in clusters and finds the vehicles which are anomalies. The Analytics application uses K-Means algorithm for clustering analysis. The number of engine capacities of the registered vehicles represent the number of clusters (K) in the algorithm. The application is connected to the backing service and consumes the stored relational data.

2.4 Results from data analytics algorithm

This section proves the relevancy of the EcoLogic algorithm by presenting a case study with two separate datasets: test dataset and real dataset. The goal of the case study is to test the capability of the algorithm to find outliers in the dataset, which represent vehicles with not optimal carbon dioxide emissions in the context of a region with many vehicles. The experiment is carried out with one real vehicle with installed hardware module. The cloud platform and services that are configured for the case study are as follows:

- All described cloud applications are deployed into SAP Cloud platform.

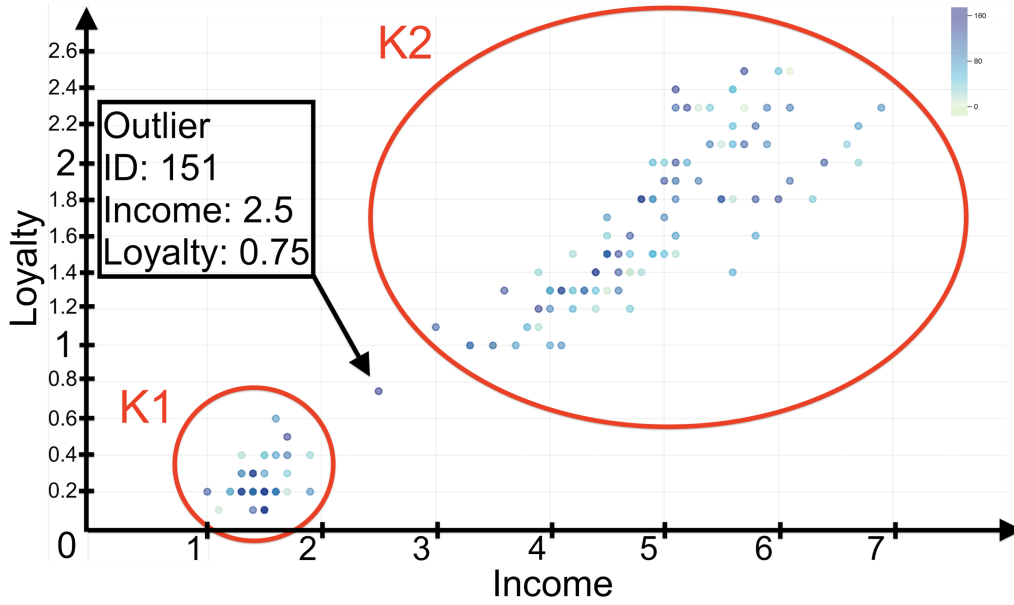


Figure 2.3: Identified clusters in dataset with known anomalies.

- The data is stored in a HANA relational database, provided as a backing service from the cloud platform [57].
- The Analytics application is represented by a SAP cloud platform predictive service, which provides an algorithm for clustering analysis used for anomaly detection.

2.4.1 Algorithm results on a dataset with known anomalies

First for validation of the correctness of the anomaly detection algorithm a test dataset, which contains known anomalies is used. The algorithm is executed on the test dataset and the returned result is compared with the known result. An official test dataset is used for that purpose [58]. It has data for customers with the following parameters: id, name, lifespand, newspend, income and loyalty. The test dataset contains 152 rows. The returned results from execution of the clustering analysis with anomaly detection on the test dataset are shown in Fig. 2.3.

The x-axis contains the income of the customers. The y-axis contains the loyalty of the customers. The algorithm successfully make two clusters (K1 and K2) and detects one anomaly. The clusters represent two kinds of customers: customers with low income and low loyalty and customers with high income and high loyalty. The anomaly, marked in Fig. 2.3 as Outlier, represents a data point, which is located too far from the centers of the clusters K1 and K2, according to other data points. The customer, which is detected as an outlier has ID: 151, income: 2.5 and loyalty: 0.75. The result obtained from the algorithm is the same as the known result from the test dataset, which proves the correctness of the algorithm for this dataset.

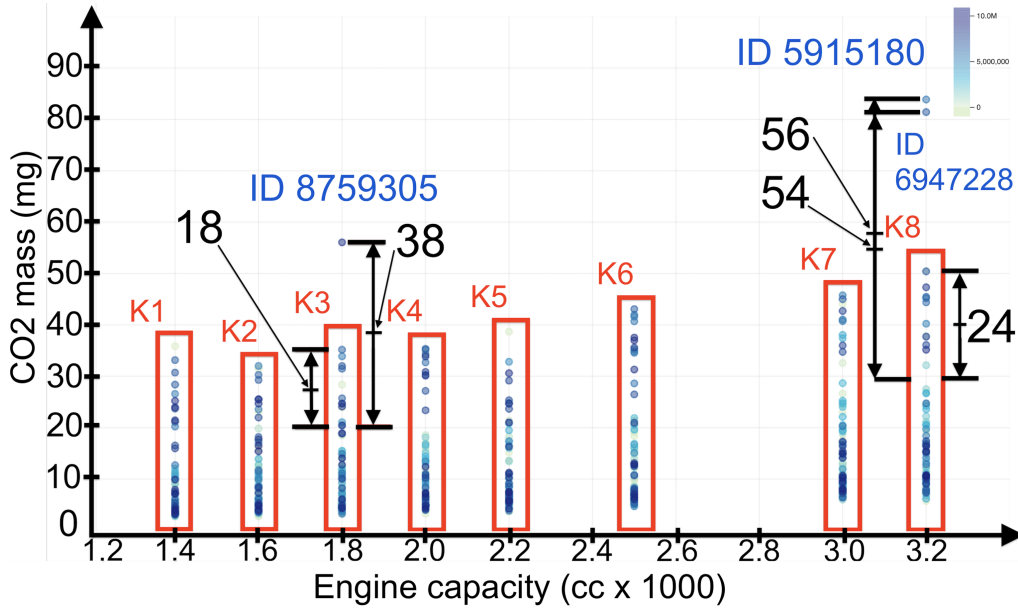


Figure 2.4: Identified clusters in real dataset ingested from vehicle.

2.4.2 Algorithm results on a real dataset

The hardware module of EcoLogic is installed into a real vehicle with internal combustion engine with a capacity of 1800 cubic centimetres, working on petrol fuel. In order to acquire bigger dataset, the collected real data is expanded with proportional simulated data. The final operative dataset contains data for vehicles with many engine capacities and carbon emission amounts. The x-axis contains the engine capacity measured in cubic centimetres (cc). The y-axis contains the mass of the carbon dioxide emissions, measured in milligrams (mg). The outcome clusters obtained after execution of the clustering algorithm are shown in Fig. 2.4. The data points, which have unique IDs, correspond to the vehicles.

The clustering algorithm should place vehicles, which have equal engine capacity and different amount of emissions preferably in the same cluster. In this way, the vehicles with not optimal emissions will not be placed in any cluster and should be detected as outliers. The algorithm returns 8 clusters (K1-K8) for the current dataset, which represent vehicles grouped in 8 different engine capacities – 1400, 1600, 1800, 2000, 2200, 2500, 3000 and 3200 cubic centimetres, respectively. They have different carbon dioxide emissions values, which vary between 2.70 and 50.44 milligrams. Vehicle with ID 8759305 has engine capacity of 1800 cubic centimetres and emissions of 56.07 milligrams. The nearest cluster center located to vehicle with ID 8759305 is those of cluster K3, with distance of 38 milligrams. The maximal internal cluster distance in cluster K3 is 18 milligrams. Vehicle with ID 6947228 has engine capacity of 3200 cubic centimetres and emissions of 81.42 milligrams. Vehicle with ID 5915180 has engine capacity of 3200 cubic centimetres and emissions of 83.86 milligrams. The

nearest cluster center located to vehicles with ID 6947228 and 5915180 is those of cluster K8, with distance of 54 and 56 milligrams respectively. The maximal internal cluster distance in cluster K8 is 24 milligrams. The vehicles with IDs 8759305, 6947228 and 5915180 are detected as outliers, because the distance from them to their nearest clusters is bigger than the internal cluster distance. These vehicles don't have optimal amount of carbon dioxide emissions in the context of the rest of the vehicles, which are located into clusters K1-K8. Important notice for the outlier with ID 8759305 is that this is the real vehicle with parameter values measured during the idle period on cold engine. The outliers with IDs 6947228 and 5915180 have simulated parameters on the base of the real vehicle's parameters on cold engine. The result obtained from the algorithm is feasible and proves its correctness on the real dataset. The hardware modules of the detected anomalies are successfully notified and actuators are activated.

2.5 Discussion

This chapter presented the EcoLogic IoT application for real-time monitoring, analysis and control of carbon dioxide emissions exposed into the atmosphere by vehicles with internal combustion engines.

The application has the following advantages:

- High availability and velocity during work with big amounts of data due to edge-native architecture.
- Platform-agnostic – possibility to work with different vehicles and clouds. The hardware modules are flexible and easy extendable to work with variety of sensors and/or collect data from the default onboard diagnostic bus or sensors of the vehicles. The implemented cloud applications are microservices, which are not vendor locked and can work on different clouds.
- Fully completed and validated solution for monitoring and control of carbon dioxide emissions from vehicles, which is ready to contribute in the efforts against climate change and reduction of the carbon dioxide emissions in the atmosphere.
- EcoLogic represents a typical practical IoT application and thus it is natively suitable to be deployed and used for evaluation of the Edge/Fog platform, presented in Chapter 4 and Chapter 5.

The results described in this chapter are published in [19].

In the next Chapter 3 a comparative analysis is made for some major related results available in the scientific field.

Chapter 3

Comparative analysis of resource allocation in Edge/Fog platforms

This chapter makes a comparative analysis of some important results for resource allocation in Edge/Fog platforms related to their major features, identified in Chapter 1. The features are infrastructure topology awareness, virtualization type, support for application microservice dependencies, dynamicity, mobility, network latency awareness, ARM64 CPU architecture support and evaluation type.

In Fog computing platforms, Edge-layer resources (processing, memory, storage and network) are restricted in size and amount, but being essential for latency-sensitive applications. This has made resource provisioning in Fog computing an important research topic. Previously it has been studied in the domain of local machines and later in the network management [59] and Cloud computing domains [60]. The resource provisioning techniques evolved from localness to remoteness. According to a recent surveys [7]–[10] we are seeing reverse movement with the research of novel resource allocation solutions in Edge/Fog domain, which favors both localness and remoteness.

3.1 Network latency

The resource provisioning problem in the Edge/Fog computing domain is studied and classified in the Bachiega et al. survey [7]. It identifies several metrics including resource utilization, cost, energy and network latency. The latency is two types: node-to-node and end-user-to-node. Additionally this survey classifies the resource allocation by several techniques: Integer Linear Programming (ILP)/Nonlinear Programming (NLP), heuristics, fit-based approaches, Multiple Criteria Decision Making (MCDM), game-based approaches and Machine Learning. Many related works usually consider ILP models to find the optimal provisioning solution based on an objective metric. The main limitation of these modeling approaches is the impossibility to find solution

in an acceptable period of time, thus limiting their applicability in real production environments. However, they can serve as an optimum indicator for heuristics-based techniques.

3.2 Virtualization type and microservice dependencies

The availability and isolation of resources (processing, memory, storage and network) in Fog computing platforms are crucial, because the devices are constrained. The resource allocation is usually achieved by virtualization model. It is classified in [7] by two parameters including Virtual Machines and containerization. Many papers consider Virtual Machines. There are works that accentuate on container placement optimizations by considering different metrics, including multi-tenant resource fair scheduling and waiting time [61] and reducing end-to-end (E2E) tail latency [62]–[64]. Additionally, the orchestrated entities are classified in Costa et al. survey [8] by task and service (microservice). The microservice dependencies are important topic when deploying big production-ready containerized applications in Fog, because the dependencies should be deployed on appropriate nodes such that the resources and latencies are optimal. They are addressed mainly in the area of batch-job scheduling [65]–[67]. They target on resource utilization [66], categorization of inter-job dependencies [65] and modeling machine learning job flow by considering training stages [67]. The dependencies are ephemeral and consecutive, meaning that later jobs depend on successfully finished execution of earlier ones.

3.3 Infrastructure topology awareness

Another important topic in Fog is the infrastructure topology awareness. It is considered in Li et al. [68], where Microservice-Oriented Topology-Aware Scheduling Framework (MOTAS) is presented. It utilizes the topologies of microservices and clusters to optimize the network overhead of microservice applications and available cluster computing resources. Another topology aware scheduler, named Gavel, is proposed by Narayanan et al. [69]. Gavel improves the execution time of deep learning jobs by focusing on the heterogeneity of cluster resources like GPUs, TPUs, FPGAs and custom ASICs. Rao et al. A topology aware placement scheme which maps microservice containers to cores on cluster machines in order to maximize performance is presented in [70]. Also a mechanism for dynamic coalescing of hot services into a single container is shown there. Additional topology aware scheduler, named RackSched, is proposed by Zhu et al. [71]. RackSched schedules resources on rack-scale computers (rack of servers with hundreds to thousands of cores) and provides high throughput at microsecond-scale tail latency with network-system co-design. Haja et al. A network-aware scheduling algorithms for big data tasks on hybrid geo-distributed Edge-Cloud

infrastructure are shown in [72]. It optimizes the infrastructure network latency and bandwidth. Its evaluation is performed in simulation environment. In general, the most works that consider data center clusters emphasize mostly on network bandwidth instead of latency.

The problem of resource provisioning in Fog-Cloud environments with containerized services is solved by Santos et al. in [73], where Mixed-Integer Linear Programming (MILP) formulation is described. The formulation considers SFC concepts, several LP-WAN technologies and objectives among which is also network latency. Four IoT smart city use case scenarios are evaluated via simulator and testbed [74]. This theoretical model is further implemented in a network-aware framework for the Kubernetes platform called Diktyo as part of subsequent work [75], where its full practical applicability is validated. Diktyo manages the placement of interdependent microservices composed of long-running applications by reducing end-to-end network latency and guaranteeing bandwidth reservations. Evaluations are done on K8s cluster built on top of 16 virtual machines provided by IBM Cloud [76]. The nodes are stationary, non-constrained and have CPU architecture different than ARM [39]. The results show that network latency is significantly reduced compared to default Kubernetes scheduling plugins, used in the latest versions of the platform (v1.22.4).

Additionally, a network-aware MILP model and system for reduction of overall network expenses in Edge Computing is studied in [77].

3.4 Dynamicity and mobility

The dynamicity and mobility support in Fog is considered in Salaht et al. survey [78]. It identifies the dynamicity of the environment in two aspects: dynamicity of Fog infrastructure and dynamicity of applications. The Fog infrastructure is dynamic, because nodes can join and leave the network due to instability of links or malfunctions. Nodes resources can also vary over time. Application dynamicity is related to changes in the workload generated by users, because users can appear or disappear and they can change their requests and responses. One Fog system supports dynamicity if it adapts to the dynamicity of the environment, for example to reschedule application containers when Fog node disappear or workload is changed.

Additionally it considers the mobility of Fog nodes and/or end-users, which means that their location can change in time and space, leading to changes in the network latency and packet loss. One Fog system supports mobility if it moves the application containers transparently from previous nodes to new ones in order to ensure application QoS/QoE. It concludes that the mobility support is important challenge that has to be addressed due to the limited number of current works researching it. The system can be not only reactive, meaning to take action in response to node or end-user movement, but also to proactively predict movements and adapt beforehand.

Dynamic replica placement algorithms for data services are presented in Aral et al. [45], Shao et al. [46] and Li et al. [79]. They achieve enhancement of the data

availability. These three works use mean latency as a metric and do not tackle the problems related to load distribution. Placement algorithm for replicated VMs in Fog Radio Access Network (F-RAN) is proposed by Yu et al. [80]. This work uses as a metric the end-user-to-edge proximity instead of edge-to-edge proximity.

Cross-edge service migration schemes for minimizing the end-user latency under constraint of a long-term migration cost budget in Mobile Edge Computing is presented by Ouyang et al. [81]. The problem of optimal service placement sequence in mobile micro clouds for minimization of the average cost for a fixed time period is studied by Wang et al. [82].

Algorithms for dynamic replica placement and maintaining application end-to-end tail latency within predefined threshold is proposed by Fahs et al. in two works, called Hona [63] and Voila [64]. They support stationary and mobile nodes. Hona provides a heuristic placement solution according to tail latency and load balancing, ensuring an optimal load distribution over a fixed-size replica set. Voila has the ability to heuristically scale up and down the replica set size based on tail latency and saturation. Both works are validated in two Kubernetes clusters: one in real testbed consisting of 22 Raspberry Pi 3 B+ devices and the other - in a simulator.

A distributed Edge orchestration framework, called ORCH, is presented by Toczé et al. [83]. It tackles the problem for task placement in distributed dynamic Edge environment. The framework manages to place end-user tasks on specific Edge device, part of pool of devices in geographical area, in order to be executed. It also manages to add additional mobile Edge devices to an area with high task load if currently such mobile devices are available in the area. It is implemented in Java as an extension of EdgeCloudSim simulator [12]. The framework however has only been validated with a proof-of-concept simulation and the model does not support resource elasticity. It also does not support migration of tasks, meaning that once a task has been placed on an Edge device for execution, all the execution will be performed there and the Edge device is blocked until finish the task. When the result is ready, if the end-user or Edge device is not in the same area because one of them moved, the task is considered as failed. The model can be improved with the concept of store-and-forward from delay-tolerant networking topic.

An extension of the two related works [64], [83], called VioLinn, is provided by Toczé et al. [84]. An optimization formulation for task placement, service placement and Edge device provisioning in Edge/Fog scenario has been presented. When end-user sends time-constraint task to closest Edge device the system determines on which devices to deploy service replicas and which replica to execute the task such that latency requirements (QoS) are met. There can be situations with sudden task load spikes and the available Edge nodes can not handle all tasks in a deadline. Currently this problem is often addressed in practice by naïve overprovisioning of Edge devices. However VioLinn introduces the concept of 'spare Edge device', which can be rented until the task load is handled completely, without having to continuously overprovision. The optimization formulation and algorithms are implemented in external Python component called Voila [64] on top of popular Cloud orchestration engine Kubernetes [85],

Reference	Scheduling Objective	Infra. Topology	µ-service deps.	Dynamicity	Mobility	Latency	ARM64	Evaluation
Aral et al.[45]	L	✗	✗	✓	✗	✓	✗	S
Beltre et al.[61]	R & LU	✗	✗	✗	✗	✓	✗	T
Chung et al.[65]	R	✗	✓	✗	✗	✗	✗	S
Crankshaw et al.[62]	R & LU	✓	✓	✓	✗	✓	✗	T
Fahs et al.[63]	R & LU	✗	✗	✓	✓	✓	✓	S & T
Fahs et al.[64]	R & LU	✗	✗	✓	✓	✓	✓	S & T
Haja et al.[72]	L & B	✓	✓	✗	✗	✓	✗	S
He et al.[67]	L & B	✓	✓	✓	✗	✓	✗	S & T
Li et al.[68]	R & L & B	✓	✓	✓	✗	✓	✗	S
Li et al.[79]	R & LU	✗	✗	✓	✓	✓	✗	T
Narayanan et al.[69]	L & B	✓	✓	✓	✗	✓	✗	S & T
Ouyang et al.[81]	L	✗	✗	✓	✓	✓	✗	S
Qiao et al.[66]	R	✓	✓	✓	✗	✗	✗	S & T
Rao et al.[70]	L & B	✓	✓	✓	✗	✓	✗	S
Santos et al.[75]	L & B	✓	✓	✗	✗	✓	✓	S & T
S.-González et al.[77]	B & C	✓	✗	✗	✗	✗	✗	S
Shao et al.[46]	L	✗	✓	✓	✗	✓	✗	S
Toczé et al.[84]	R & LU	✗	✗	✓	✓	✓	✓	S & T
Wang et al.[82]	R	✗	✗	✓	✓	✗	✗	S
Yu et al.[80]	B	✗	✗	✓	✓	✗	✗	S
Zhu et al.[71]	L & B	✓	✗	✗	✗	✓	✗	T
Dissertation	L	✓	✓	✓	✓	✓	✓	T

Table 3.1: Comparison between related works

Scheduling Objective: R = Resources, AD = App. Dependencies, T = Topology-aware, L = Latency (node to node), B = Bandwidth, LU = User latency (end-user to replica), C = Cost

Infrastructure Topology Awareness, Microservice Dependencies, Latency, Mobility, ARM64 CPU: ✓ = supported, ✗ = not supported

Evaluation: S = Simulation, T = Testbed

but not the latest version. The provided solution is validated in real testbed consisting of 22 Raspberry Pi 3B+ devices and in simulated testbed using Open-Radio Access Network (O-RAN) use case.

Table 3.1 shows comparison of the papers discussed above with the proposed model concerning the scheduling objective, infrastructure topology awareness, microservice dependencies, dynamicity, mobility, network latency, ARM64 support and evaluation type. The big amount of related research works proves the importance of the topic.

Later we will see that the investigation from the dissertation rely on containerization due to its better usage in practice. It manages long-living microservices with dependencies, which means that all containers are running permanently in a tandem as a whole application. It is aware of the infrastructure topology by using network latency characteristics of the links between infrastructure nodes. It is dynamic (update placements based on infrastructure topology changes) and supports movement of nodes by monitoring the inter-node latencies. Furthermore it supports constrained devices with ARM64 CPU architecture.

In next Chapter 4 a MILP model with the above listed features will be formulated.

Chapter 4

Mixed-Integer Linear Programming (MILP) model

The problem for dynamic network-aware allocation of microservice containers in Cloud/Fog infrastructures composed of set of moveable nodes, which change their computational and network resource capabilities can be considered as an optimization problem with NP-hard complexity [2]. Many ILP models providing optimal solutions for this problem are examined, but their variables and objectives do not correlate with the model of the real Cloud/Fog platforms and could not be applied in practice. They are not suitable for IoT constrained devices with ARM architecture and use simulation-based evaluation.

A novel MILP optimization model for network-aware microservice container provisioning in dynamic Cloud/Fog infrastructures composed of moveable and constrained nodes with ARM architecture is proposed in the dissertation. Its objective functions maximize the total number of deployed workloads, minimize total replica movements across nodes and minimize the workload's network latency in order to provide the most optimal placement solution. The variables, constraints and objective functions of the model are explained in the next sections.

4.1 Model description and variables

The MILP model is tailored towards the K8s deployment model and can be used for wide variety of Cloud/Fog infrastructures. It is treating the Cloud/Fog infrastructure as a set of nodes with limited computational resources such as CPU and memory. The nodes are interconnected with network links, which have specific latency and bandwidth characteristics which may vary between iterations. The microservices, running on the cluster, are represented as a set of workloads with specific dependencies and requirements in terms of computational resources, network latency and bandwidth. The placement of the microservice containers on the nodes is directly affecting the final end-to-end latency and bandwidth characteristics of the application. In order to pro-

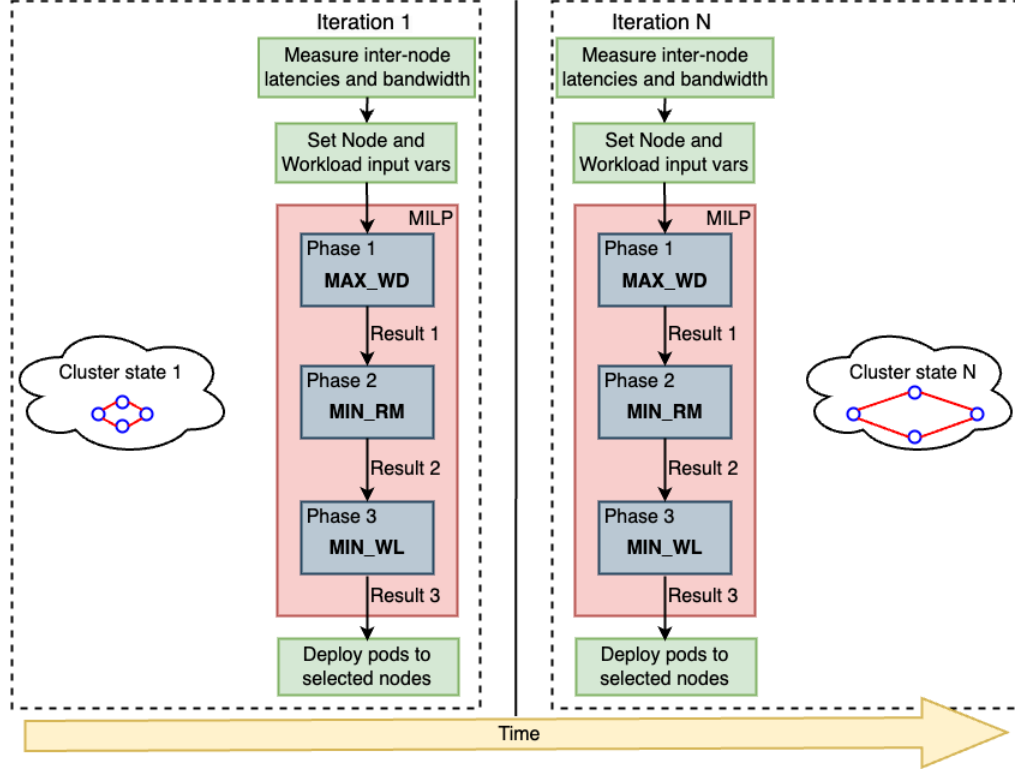


Figure 4.1: MILP model orchestration and execution.

duce placement with minimal end-to-end latency in dynamic environment the model has objectives for maximization of the workload deployments and minimization of the replica movements and network latency. These objective functions are executed one after another. The overall MILP model orchestration and its execution are illustrated in Fig. 4.1.

4.1.1 Input variables for nodes

The model formulates the set of nodes in a Kubernetes-based Cloud/Fog cluster as a set $N = \{n_1, n_2, \dots\}$. Each node n has the following computational and network resources:

- C_n : Total CPU cores capacity in CPU units, where $C_n \in \mathbb{Q}^+$, $C_n > 0$.
- E_n : Total memory capacity in Megabytes (Mbyte), where $E_n \in \mathbb{N}$.
- B_n : Total bandwidth capacity in Megabits/second (Mbit/sec), where $B_n \in \mathbb{N}$.

The network latency and bandwidth parameters between each two nodes are defined with the following matrices:

- B_{n_u, n_v} : Bandwidth matrix representing network bandwidth capacity from source node n_u to target node n_v in Megabits/second (Mbit/sec).
- T_{n_u, n_v} : Latency matrix representing network latency from source node n_u to target node n_v in milliseconds (ms).

Nodes are separated in infrastructure regions and zones forming a specific grouping. Each region is composed of zones. Each zone z is located in only one region u . A node $n \in N$ can be located in exactly one region and zone. The set of infrastructure regions and zones are defined as:

- $U = \{u_1, u_2, \dots\}$: Set of infrastructure regions, where each region u is a string name.
- $Z = \{z_1, z_2, \dots\}$: Set of infrastructure zones, where each zone z is a string name.

The node membership matrix $M_{n,u,z}$ (binary) represents whether node n is member of region u and zone z in that region. We will write $M_{n,u,z} = 1$ if node n is member of region u and zone z .

4.1.2 Input variables for workloads

Each business application is defined as workload w . All workloads are forming a set of workloads $W = \{w_1, w_2, \dots\}$.

Microservices are called pods in the K8s model. Each workload w is composed of set of pods $P = \{p_1, p_2, \dots\}$. The pod membership matrix M_p^w (binary), defines whether pod p is member of workload w . We will denote $M_p^w = 1$ if pod p is member of workload w .

Each pod has one or more instances, called replicas in the K8s model. Each pod is composed of set of replicas $R = \{r_1, r_2, \dots\}$. The number of requested replicas for each pod p is defined with R_p^{req} . The maximal limit of replica instances for each pod p is defined with R_p^{max} .

Each pod p has the following resource requirements necessary for its proper functioning:

- c_p : CPU requirement in CPU units, where $c_p \in \mathbb{Q}^+$, $c_p > 0$.
- e_p : Memory requirement in Megabytes (Mbyte), where $e_p \in \mathbb{N}$.
- b_p : Bandwidth requirement in Megabits/sec (Mbit/sec), where $b_p \in \mathbb{N}$.

As an example a CPU requirement that is equal to 0.1 cpu (i.e. 100 millicpu) means that the pod needs 10% of a CPU core to function properly.

Inter-pod dependency requirements for network bandwidth and latency are defined in the following matrices:

- β_{p_i, p_j} : Bandwidth matrix defining network bandwidth requirement in the link from source pod p_i to target pod p_j .
- τ_{p_i, p_j} : Latency matrix defining network latency requirement in the link from source pod p_i to target pod p_j .

Additionally the requirements predicate matrices define whether the corresponding inter-pod latency and bandwidth requirements are guaranteed or not:

- Ω_{p_i, p_j}^β : Workload bandwidth requirements predicate matrix (binary). We will write $\Omega_{p_i, p_j}^\beta = 1$ if network bandwidth requirements from source pod p_i to target pod p_j must be assured.
- Ω_{p_i, p_j}^τ : Workload latency requirements predicate matrix (binary). We will denote $\Omega_{p_i, p_j}^\tau = 1$ if network latency requirements from source pod p_i to target pod p_j must be assured.

Furthermore the replica movement factor δ is used for hard limitation of the total number of possible replica movements for pods across nodes in the cluster, affecting the scheduling and descheduling decisions. The input variables of the model related to the infrastructure nodes and workloads are described in Table 4.1 and Table 4.2, respectively. They have to be set in each iteration, before the execution of the objective functions, as shown in Fig. 4.1. Node input variables require measurements of node characteristics. The computational characteristics, like CPU, Memory and Storage are measured by the infrastructure platform (K8s), while the inter-node latency and bandwidth are measured by an external component, which could be integrated into the platform. These measurements are taken and the node input variables are set automatically in the beginning of each iteration. It is possible to set (overwrite) node-related input variables manually, if specific values are required. The workload input variables are set manually by the application developers or platform engineers, because they have knowledge related to the workload requirements. All input variables represent the latest cluster state at each iteration (nodes movement). Their values can be static or dynamic between iterations, depending on the node movements. The MILP orchestration and execution are shown in Algorithm 1 and Algorithm 2 respectively.

4.1.3 Decision variables

The workload deployment matrix D_w (binary), represents the deployment state of each workload as a whole, whether all of its pods are deployed on nodes and fully operational. We will denote $D_w = 1$ if the whole workload w is deployed.

The pod deployment matrix D_p^w (binary) considers the deployment state of each pod from concrete workload. We will write $D_p^w = 1$ if the whole pod p from workload w is deployed, meaning that all of its replicas are deployed.

Algorithm 1: MILP model orchestration

```
1 Function milpOrchestrate():
2   while milpIterationPeriod() do
3     measureNodeResources()
4     measureInterNodeLatenciesAndBandwidth()
5     inputVars ← setMilpInputVars()
6     decisionVars ← milpExecute(inputVars)
7     scheduledPodsMap ← getScheduledPods(decisionVars)
8     deployPodsOnNodes(scheduledPodsMap)
```

Algorithm 2: MILP model execution

```
Input: in
/* MILP input variables (Table 4.1, Table 4.2) */
Output: result3
/* MILP decision variables (Table 4.3) */
1 Function milpExecute(in):
2   result1 ← MAX_WD(in)
3   result2 ← MIN_RM(in, result1)
   /* Preserve number of deployed workloads from MAX_WD */ /*
4   result3 ← MIN_WL(in, result2)
   /* Preserve number of deployed workloads from MAX_WD, MIN_RM */ /*
5   return result3
```

The replica deployment matrix $D_{r,n}^{w,p}$ (binary) represents whether a concrete replica from pod is placed on concrete node. We will denote $D_{r,n}^{w,p} = 1$ if replica r from pod p , part of workload w , is deployed on node n .

Inter-replica stream matrix $S_{p_j, r_m}^{w, p_i, r_k}(n_u, n_v)$ (binary) represents the network streams. We will write $S_{p_j, r_m}^{w, p_i, r_k}(n_u, n_v) = 1$ if replica r_k of pod p_i is deployed on node n_u and replica r_m of pod p_j is deployed on node n_v for workload w . It asserts that network stream is achieved between specific replicas, deployed on specific nodes.

The workload latency matrix T_w gives the total end-to-end network latency of workload w in milliseconds.

The replica movement per iteration $\Delta_{r,n}^{w,p}$ (binary) represents whether replica r_i is moved from node n to another. We will denote $\Delta_{r,n}^{w,p} = 1$ if replica r has moved from node n to another, according to the previous iteration.

The replica movement matrix $\Delta_r^{w,p}$ (binary) shows whether replica r_i is moved from one node to another. We will write $\Delta_r^{w,p} = 1$ if replica r has moved from node, according to the previous iteration.

All decision variables are shown in Table 4.3.

4.2 Objectives and constraints

The MILP model has the following objectives:

- Maximization of workload deployments (MAX_WD).
- Minimization of microservice replica movements across nodes between iterations (MIN_RM).
- Minimization of workload end-to-end network latency (MIN_WL).

These objective functions are executed one after another in consecutive passes, depicted in Fig. 4.1 and described in Algorithm 1 and Algorithm 2. In the beginning, the maximization of workload deployments (MAX_WD) is executed. The result from the first objective is passed-through to the second objective, which considers replica movements across nodes between iterations (MIN_RM). If necessary it minimizes the replica movements according to constraints and modifies the result. The result from the second pass is given to the third objective, which further enhances the result towards minimization of the end-to-end latency (MIN_WL). Here we differentiate the two mentioned terms: passes vs. iterations. The passes are the separate phases in which the different optimization objective functions are executed in continual manner as part of the MILP model. The iterations are the separate executions of the model as a whole. Nodes may move in space and change their network characteristics between each iteration. In other words, one iteration of the model consists of one execution of the model with its three phases with objectives. These three objective functions can be merged in one, but it will become more complex. This can be tackled as a future work.

All objectives are described in the following subsections.

4.2.1 Maximize workload deployments (MAX_WD)

The MAX_WD objective function is responsible for maximization of workload deployments. It is subject to several constraints.

Pods to workload deployment constraint

$$\sum_{p \in P} D_p^w = \sum_{p \in P} M_p^w,$$

where $D_w = 1$, for $\forall w \in W$, assures that workload w is fully deployed if all pods, which are members of the workload w , are deployed.

As pods consist of microservice replicas, the replicas to pod deployment constraint

$$\sum_{r \in R_p^{max}} \sum_{n \in N} D_{r,n}^{w,p} = M_p^w \times R_p^{req},$$

where $D_p^w = 1$, for $\forall w \in W, \forall p \in P$, states that pod p is fully deployed if the requested number of its replicas (R_p^{req}) are scheduled.

Because the model decides on which nodes to deploy replicas, the following constraints assures that replicas are scheduled just on the nodes with enough available CPU, memory and network bandwidth resources according to the pod requirements:

$$\begin{aligned} \sum_{w \in W} \sum_{p \in P} \sum_{r \in R_p^{max}} D_{r,n}^{w,p} \times c_p &\leq C_n, \\ \sum_{w \in W} \sum_{p \in P} \sum_{r \in R_p^{max}} D_{r,n}^{w,p} \times e_p &\leq E_n, \\ \sum_{w \in W} \sum_{p \in P} \sum_{r \in R_p^{max}} D_{r,n}^{w,p} \times b_p &\leq B_n, \\ b_{p_i} &= \sum_{\substack{j=1 \\ j \neq i}}^{|P|} \beta_{p_i, p_j}, \end{aligned}$$

for $\forall n \in N, \forall p_i \in P$, where pod bandwidth is defined as b_{p_i} .

Then the model is capable to deploy more than one replica from a pod on a same node, which is undesirable, so the replica spread across nodes constraint

$$\sum_{r \in R_p^{max}} D_{r,n}^{w,p} = 0 \text{ or } 1,$$

for $\forall w \in W, \forall p \in P, \forall n \in N$, assures that each replica from one pod is deployed on different node.

The model is also able to place all replicas in a single region and zone, which again is undesirable due to concerns related with the high availability of the workload. So the replica spread across regions and zones constraint

$$\sum_{n \in N} \sum_{r \in R_p^{max}} D_{r,n}^{w,p} = \begin{cases} 1 & \text{if } M_{n,u,z} = 1 \\ 0 & \text{if } M_{n,u,z} = 0, \end{cases}$$

for $\forall w \in W, \forall p \in P, \forall u \in U, \forall z \in Z$, makes the pod replicas to be spread across different regions and zones.

Finally the maximization of workload deployments objective function is defined as:

$$\text{MAX_WD} = \max \sum_{w \in W} D_w.$$

It is aiming to deploy as much as possible number of workloads on the available set of nodes, meeting all constraints. The MAX_WD is shown further in Algorithm 3.

Algorithm 3: Maximize workload deployments (MAX_WD) objective function

Input: in
 /* MILP input variables (Table 4.1, Table 4.2) */
Output: out
 /* MILP decision variables (Table 4.3) */

```

1 Function MAX_WD(in):
2   out ← initDecisionVars(in)
3   while PodsToWorkloadConstraint(in) and
4     ReplicasToPodConstraint(in) and
5     PodResourceConstraint(in) and
6     ReplicaSpreadOnNodesConstraint(in) and
7     ReplicaSpreadOnZonesConstraint(in) do
8     | out ← maximizeWorkloadDeployments(in, out)
9   return out

```

4.2.2 Minimize replica movements (MIN_RM)

The result from the first objective (MAX_WD) consists of the concrete placements of pod replicas on the set of nodes. It includes the deployed workloads decision variable D_w . This result is passed-through to the second objective (MIN_RM), which minimizes the number of replica movements across nodes, according to the δ input variable, which is used for limitation of replica movements. In order to preserve the number of already deployed workloads from the first objective function (MAX_WD), there is an auxiliary constraint defined as:

$$\sum_{w \in W} D_w = \text{result_from_}(\text{MAX_WD}).$$

In this way the second objective (MIN_RM) is enhancing the first result by preserving the number of scheduled replicas from the previous objective (MAX_WD).

Minimization of the replica movements between iterations objective is subject to several constraints. Let us define replica movement per iteration decision variable as:

$$\Delta_{r,n}^{w,p} = \begin{cases} 0 & \text{if } D_{r,n}^{w,p} = 1 \wedge (D_{r,n}^{w,p})^{-1} = 1 \\ & \text{if } D_{r,n}^{w,p} = 0 \wedge (D_{r,n}^{w,p})^{-1} = 0 \\ 1 & \text{Otherwise,} \end{cases}$$

for $\forall w \in W, \forall p \in P, \forall r \in R_p^{max}, \forall n \in N$, where $(D_{r,n}^{w,p})^{-1}$ is the deployment matrix from the previous iteration. Here iteration means the previous execution of the MILP model.

Then replica movement matrix, which contains movements for all replicas, is defined as:

$$\Delta_r^{w,p} = \begin{cases} 0 & \text{if } \sum_{n \in N} \Delta_{r,n}^{w,p} = 0 \\ 1 & \text{if } \sum_{n \in N} \Delta_{r,n}^{w,p} \geq 1, \end{cases}$$

for $\forall w \in W, \forall p \in P, \forall r \in R_p^{max}$.

Furthermore the assurance of total number of possible replica movements from one node to another for each pod p in W is defined as:

$$\sum_{p \in P} \sum_{r \in R_p^{max}} \Delta_r^{w,p} \times M_p^w \leq \delta \times \sum_{p \in P} R_p^{req} \times M_p^w,$$

for $\forall w \in W$. From this constraint follows that:

- $\delta \geq R_p^{max}$, if we want to not limit the movements of replicas across nodes.
- $\delta < R_p^{max}$, if we want to limit the movements of replicas across nodes.

Finally the minimization of replica movements between iterations objective function is defined as:

$$\text{MIN_RM} = \min \sum_{w \in W} \sum_{p \in P} \sum_{r \in R} \Delta_r^{w,p}.$$

The minimization of replica movements is useful if we want to reduce the replica movements between iterations, because in some cases the migration of replicas require additional computational overhead and introduce network latency, which may be unnecessary in very dynamic environment, where cluster nodes are moving very often. The MIN_RM is shown in Algorithm 4.

4.2.3 Minimize workload end-to-end network latency (MIN_WL)

The result from the second objective (MIN_RM) consists of the refined placements of pod replicas across nodes. Again it includes the deployed workloads decision variable D_w . This result is passed-through to the third objective (MIN_WL), which aim is to minimize the workload end-to-end latency. In order to preserve the number of already deployed workloads from the first and second objective functions (MAX_WD, MIN_RM), there is an auxiliary constraint defined as:

$$\sum_{w \in W} D_w = \text{result_from_}(\text{MIN_RM}).$$

Algorithm 4: Minimize replica movements across nodes (MIN_RM) objective function

Input: in, out
 /* MILP input variables (Table 4.1, Table 4.2), decision variables (Table 4.3) */
Output: out
 /* MILP decision variables (Table 4.3) */
 1 **Function** MIN_RM(*in*, *out*):
 2 **while** *preserveAlreadyDeployedReplicasConstraint(out)* **and**
 /* Preserve number of deployed workloads from MAX_WD */
 3 *replicaMovementsByMoveFactorConstraint(in)*
 4 **do**
 5 *out* \leftarrow *minimizeReplicaMigrations(in, out)*
 6 **return** *out*

In this way the third objective (MIN_WL) is further enhancing the result, but also preserving the number of provisioned replicas from previous objectives.

Minimization of the workload end-to-end network latency is subject to several constraints regarding the infrastructure network. First the inter-replica stream matrix represents the specific network streams between all replicas placed on specific nodes in the cluster. It is defined as:

$$S_{p_j, r_m}^{w, p_i, r_k}(n_u, n_v) = \begin{cases} 1 & \text{if } D_{r_k, n_u}^{w, p_i} \wedge D_{r_m, n_v}^{w, p_j} = 1 \\ 0 & \text{Otherwise,} \end{cases}$$

for $\forall w \in W, \forall p_i, p_j \in P, i \neq j, \forall r_k, r_m \in R_p^{max}, \forall n_u, n_v \in N, u \neq v$.

Inter-replica stream with bandwidth preservation constraint is defined as:

$$\begin{aligned} \sum_{w \in W} \sum_{p_i, p_j \in P} \sum_{r_k, r_m \in R_p^{max}} \sum_{n_u, n_v \in N} S_{p_j, r_m}^{w, p_i, r_k}(n_u, n_v) = \\ \sum_{p_i, p_j \in P} \Omega_{p_i, p_j}^\beta \times R_{p_i}^{req} \times R_{p_j}^{req}, \end{aligned}$$

for $\forall w \in W, \forall p_i, p_j \in P, i \neq j, \forall r_k, r_m \in R_p^{max}, \forall n_u, n_v \in N, u \neq v$. It assures that all network streams between pods, which require guaranteed bandwidth, are preserved and there are no traffic losses in the cluster.

Inter-replica stream with latency preservation constraint is defined as:

$$\begin{aligned} \sum_{w \in W} \sum_{p_i, p_j \in P} \sum_{r_k, r_m \in R_p^{max}} \sum_{n_u, n_v \in N} S_{p_j, r_m}^{w, p_i, r_k}(n_u, n_v) = \\ \sum_{p_i, p_j \in P} \Omega_{p_i, p_j}^\tau \times R_{p_i}^{req} \times R_{p_j}^{req}, \end{aligned}$$

for $\forall w \in W, \forall p_i, p_j \in P, i \neq j, \forall r_k, r_m \in R_p^{max}, \forall n_u, n_v \in N, u \neq v$. It assures that all network streams between pods, which require guaranteed maximal latency threshold, are preserved and there are no traffic losses in the cluster.

Let us define a stream bandwidth factor as:

$$s_{p_i, p_j}^\beta = M_{p_i}^w \times M_{p_j}^w \times \Omega_{p_i, p_j}^\beta,$$

for $\forall p_i, p_j \in P, i \neq j$. We will write $s_{p_i, p_j}^\beta = 1$ if source pod p_i depends on target pod p_j and the network bandwidth requirements between them must be guaranteed.

Then the stream bandwidth constraint is defined as:

$$\sum_{w \in W} \sum_{p_i, p_j \in P} \sum_{r_k, r_m \in R_p^{max}} s_{p_i, p_j}^\beta \times \beta_{p_i, p_j} \times S_{p_j, r_m}^{w, p_i, r_k}(n_u, n_v) \leq B_{n_u, n_v},$$

for $\forall n_u, n_v \in N, u \neq v$. It validates that the total available network bandwidth in the cluster is respected.

Let us define stream latency factor as:

$$s_{p_i, p_j}^\tau = M_{p_i}^w \times M_{p_j}^w \times \Omega_{p_i, p_j}^\tau,$$

for $\forall p_i, p_j \in P, i \neq j$. We will write $s_{p_i, p_j}^\tau = 1$ if source pod p_i depends on target pod p_j and the network latency requirements between them must be guaranteed.

Algorithm 5: Minimize workload end-to-end network latency (MIN_WL) objective function

```

Input: in, out
/* MILP input variables (Table 4.1, Table 4.2), decision variables
   (Table 4.3) */
Output: out
/* MILP decision variables (Table 4.3) */
1 Function MIN_WL(in, out):
2   while preserveAlreadyDeployedReplicasConstraint(out) and
   /* Preserve number of deployed workloads from MAX_WD, MIN_RM */
3   interReplicaStreamMatrixBandwidthConstraint(in) and
4   interReplicaStreamMatrixLatencyConstraint(in) and
5   streamBandwidthFactorConstraint(in) and
6   streamLatencyFactorConstraint(in)
7   do
8      $\lfloor$  out  $\leftarrow$  minimizeWorkloadTotalNetworkLatency(in, out)
9    $\rfloor$  return out

```

Then the stream latency constraint is defined as:

$$\sum_{w \in W} \sum_{p_i, p_j \in P} \sum_{r_k, r_m \in R_p^{max}} s_{p_i, p_j}^\tau \times \tau_{p_i, p_j} \times S_{p_j, r_m}^{w, p_i, r_k}(n_u, n_v) \leq T_{n_u, n_v},$$

for $\forall n_u, n_v \in N, u \neq v$. It validates that the total network latency between cluster nodes is respected.

Furthermore the workload latency matrix is defined as:

$$T_w = \sum_{p_i, p_j \in P} \sum_{r_k, r_m \in R_p^{max}} \sum_{n_u, n_v \in N} T_{n_u, n_v} \times S_{p_j, r_m}^{w, p_i, r_k}(n_u, n_v),$$

for $\forall w \in W$. It states the total end-to-end latency of workload w in milliseconds unit.

Finally the minimizing workload end-to-end network latency objective function is defined as:

$$\text{MIN_WL} = \min \sum_{w \in W} T_w.$$

In order to obtain minimal workload end-to-end latency, this objective is placing dependent pods on nodes with smaller inter-latencies, usually located close to each other in the space. The MIN_WL is described further in Algorithm 5.

The results presented in this chapter are published in [86].

In the next Chapter 5 the described above MILP model will be applied with examples and will be validated in practice.

Symbol	Description
b_p	Bandwidth requirement of pod p in Megabits/second (Mbit/sec), where $b_p \in \mathbb{N}$.
c_p	CPU requirement of pod p in CPU units, where $c_p \in \mathbb{Q}^+$, $c_p > 0$.
e_p	Memory requirement of pod p in Megabytes (Mbyte), where $e_p \in \mathbb{N}$.
B_n	Total bandwidth capacity of node n in Megabits/second (Mbit/sec), where $B_n \in \mathbb{N}$.
B_{n_u, n_v}	Bandwidth matrix representing network bandwidth capacity from source node n_u to target node n_v .
C_n	Total CPU cores capacity of node n in CPU units, where $C_n \in \mathbb{Q}^+$, $C_n > 0$.
E_n	Total memory capacity of node n in Megabytes (Mbyte), where $E_n \in \mathbb{N}$.
M_p^w	Membership matrix (binary) $M_p^w \in \{0, 1\}$. $M_p^w = 1$ if pod p is member of workload w .
$M_{n,u,z}$	Node membership matrix (binary), representing whether node n is member of region u and zone z in that region. $M_{n,u,z} = 1$ if node n is member of region u and zone z .
N	Set of nodes $n \in N$, on which pod replica instances are deployed.
P	Set of pods $p \in P$. Each pod p is composed of one or more replica instances.
R_p^{req}	Requested number of replica instances r for each pod p .
R_p^{max}	Maximum limit of number of replica instances for each pod p .
T_{n_u, n_v}	Latency matrix representing network latency from source node n_u to target node n_v in milliseconds (ms).
U	Set of infrastructure regions (strings). $U = \{u_1, u_2, \dots\}$. Each region is composed of zones. A node $n \in N$ can be located in exactly one region and zone.
W	Set of workloads $w \in W$. Each workload w is composed of one or more pods $p \in P$ with or without dependencies.
Z	Set of infrastructure zones (strings). $Z = \{z_1, z_2, \dots\}$. Each zone is located in only one region u . A node $n \in N$ can be located in exactly one region and zone.

Table 4.1: Mixed-Integer Linear Programming (MILP) input variables for infrastructure nodes.

Symbol	Description
β_{p_i,p_j}	Bandwidth matrix for inter-pod communication, representing network bandwidth requirement in the link from source pod p_i to target pod p_j .
δ	Replica movement factor, used for limitation of the total number of possible replica movements for pods across nodes.
τ_{p_i,p_j}	Latency matrix for inter-pod communication, representing network latency requirement in the link from source pod p_i to target pod p_j .
Ω_{p_i,p_j}^β	Workload bandwidth requirements predicate matrix (binary). $\Omega_{p_i,p_j}^\beta = 1$ if network bandwidth requirements from source pod p_i to target pod p_j must be guaranteed.
Ω_{p_i,p_j}^τ	Workload latency requirements predicate matrix (binary). $\Omega_{p_i,p_j}^\tau = 1$ if network latency requirements from source pod p_i to target pod p_j must be guaranteed.

Table 4.2: Mixed-Integer Linear Programming (MILP) input variables for workloads.

Symbol	Description
s_{p_i,p_j}^β	Stream bandwidth factor (binary), representing pod dependencies in a network stream. $s_{p_i,p_j}^\beta = 1$ if source pod p_i depends on target pod p_j and the network bandwidth requirements between them must be guaranteed (greater or equal to predefined bandwidth limit β_{p_i,p_j}).
s_{p_i,p_j}^τ	Stream latency factor (binary), representing pod dependencies in a network stream. $s_{p_i,p_j}^\tau = 1$ if source pod p_i depends on target pod p_j and the network latency requirements between them must be guaranteed (less or equal to predefined latency limit τ_{p_i,p_j}).
D_w	Workload deployment matrix (binary). $D_w = 1$ if workload $w \in W$ is fully deployed.
D_p^w	Pod deployment matrix (binary). $D_p^w = 1$ if pod $p \in P \in W$ is fully deployed.
$D_{r,n}^{w,p}$	Replica deployment matrix (binary). $D_{r,n}^{w,p} = 1$ if replica $r \in R \in P \in W$ is deployed on node n .
$S_{p_j,r_m}^{w,p_i,r_k}(n_u, n_v)$	Inter-replica stream matrix (binary), representing replica and node dependencies in a network stream. $S_{p_j,r_m}^{w,p_i,r_k}(n_u, n_v) = 1$ if replica r_k of pod p_i is deployed on node n_u and replica r_m of pod p_j is deployed on node n_v for workload w and network stream between them is achieved.
T_w	Workload latency matrix, representing total end-to-end network latency of workload w in milliseconds.
$\Delta_r^{w,p}$	Replica movement matrix (binary), representing whether replica r_i is moved from one node to another. $\Delta_r^{w,p} = 1$ if replica r has moved from at least one node to another, according to the previous iteration.
$\Delta_{r,n}^{w,p}$	Replica movement per iteration (binary), representing whether replica r_i is moved from node n to another. $\Delta_{r,n}^{w,p} = 1$ if replica r has moved from node n to another, according to the previous iteration.

Table 4.3: Mixed-Integer Linear Programming (MILP) decision variables

Chapter 5

Results and discussion

Pod	Execution time result 1 replicas #	Execution time result 2 replicas #	Execution time result 3 replicas #	Example 1 and 2 (E1, E2) replicas #
Database (P1)	1	1	1	1
Backend (P2)	1	2	2	2
VehicleAgent (P3)	1	3	6	6
Frontend (P4)	1	2	2	2

Table 5.1: EcoLogic application Pods with Replica counts.

In this chapter two examples and an execution time result will be shown, to demonstrate how the described model handles the identified in Chapter 1 major features of Edge/Fog computing platforms in a realistic mobile environment. The examples are conducted in a testbed, which uses Kubernetes (version 1.24.3) Cloud/Fog platform and container-based virtualization. They are aiming to evaluate realistic workload represented by the EcoLogic sample application [19]. Its microservice dependencies are shown in Fig. 1.1. Its full architecture is shown in Fig. 2.1, where the Database (P1), Backend (P2), VehicleAgent (P3) and Frontend (P4) microservices are described in more details. The Analytics microservice is not used into the examples. The total count of used microservice replicas (instances) is eleven. Each microservice replica is deployed on different node. The testbed is emulating dynamic environment composed of geo-distributed mobile Edge/Fog nodes moving in space. The setup is like the one shown in Fig. 1.2, but eleven nodes are used instead of four. Its aim is to evaluate how the proposed model reduces total end-to-end application network latency in this environment.

Example 1 (E1). We consider eleven geographical cities in which nodes N are located, forming a realistic geo-distributed cluster infrastructure, like the setup shown in Fig. 1.2. The node's resource capacities B_n , C_n , E_n , T_{n_u, n_v} , B_{n_u, n_v} and all remaining input variable values of the model, which are used in the examples, are shown in Table 5.2. All nodes are located in different regions U and zones Z . Their count is eleven, which equal to the total count of the used microservice replicas R of the

EcoLogic workload W . The replicas count R_p^{req} for the concrete EcoLogic pods P , which are used in the examples are shown in Table 5.1. The requested number of replicas is equal to the maximum number of replicas permitted in the model ($R_p^{req} = R_p^{max}$). Each node contains only one deployed microservice replica. The real pairwise network latency measurements between these cities are used for emulation of the geo-distributed infrastructure on the testbed. The latency between nodes T_{n_u, n_v} vary from 10 to 60 milliseconds. The average initial distance between nodes x_1 is 800 kilometers. On each step all nodes are moved away from each other with a fixed distance Δx ($\Delta x = x_2 - x_1$). In **E1** Δx equals to 10% of the average initial distance (80 kilometers). We use 20 movement steps, which is enough long period for analysis of the node movements and changes in network characteristics. All pods request CPU c_p and Memory e_p resources of 100 millicpu and 100 Mb respectively. Specific network latency τ_{p_i, p_j} and bandwidth β_{p_i, p_j} requirements between each two dependent pods are used, with values of 30 milliseconds and 100 Mbps respectively. Replica movements across nodes are not limited ($\delta = R_p^{max} = 6$). The MILP model is executed on each movement step (iteration), as described in Fig. 4.1. On each movement step the same input variable values are conducted, except the inter-node latency (T_{n_u, n_v}), which is changing to match on the movement distance Δx . At the end of each iteration, the MILP model returns a placement scheme ($D_{r,n}^{w,p}$) and pod replicas are deployed on concrete nodes.

Depending on the movement distance, the inter-pod latency requirement τ_{p_i, p_j} has to be enough restrictive in order to cause violations of the network requirements and cause subsequent pod reallocations and optimization. But if its value is too restrictive, it is possible to enter a situation without any feasible nodes for reallocation. In this case the model will leave the placement scheme ($D_{r,n}^{w,p}$) in its latest state. The currently deployed applications (workloads) will be available, but with reduced QoS — their total end-to-end latency will be increased and above the configured requirement. This situation will be resolved if nodes move to more favorable (feasible) locations or new additional nodes join the cluster infrastructure. When the network latency requirement is chosen correctly in this boundary zone, evaluations can be used for analysis of the level of optimization of the MILP model compared to other solutions. So this is taken into account in the next example.

Example 2 (E2). The design and parameters are the same as in **E1**, depicted in Table 5.2 and Table 5.1. The only difference is that the nodes are moved away from each other with twice bigger distance Δx than in **E1**, which equals to 20% of the average initial distance (160 kilometers).

The results of the examples **E1** and **E2** are visualized in Fig. 5.1 and Fig. 5.2, respectively. The figures contain one plot with movement steps shown on its abscissa axis and the average workload end-to-end latency in milliseconds on its ordinate axis. There are three graphs, representing Kubernetes Default Scheduler (DS), Network-aware Scheduler (NS) and Custom Scheduler (CS) with MILP. The violations of the network latency requirements are shown in vertical dotted lines on the corresponding movement steps. Pod placement schemes of the NS and CS schedulers are different when there is a violation, which causes a different average latency results.

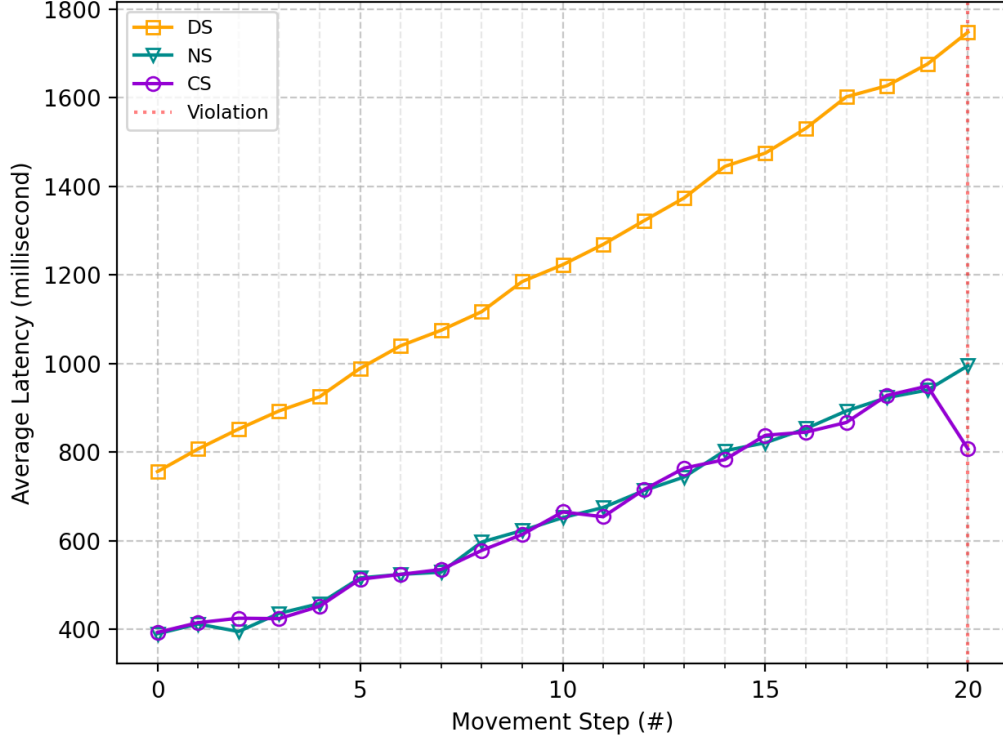


Figure 5.1: Example **E1** with small movement distance $\Delta x = 10\%(80km)$ per step. CS optimizes end-to-end latency by 20% in last step 20, compared to existing K8s schedulers.

In example **E1** results (Fig. 5.1), NS and CS have lower end-to-end network latency compared to DS. NS and CS have practically same latency until the last step due to the same initial placement schemes in the first step. There is a network latency violation happening in last step 20, which results to pod reallocations and lower latency for CS compared to NS, reduced by 20%. The late reoptimization on step 20 means that the network latency requirement τ_{p_i, p_j} is loose.

Example **E2** moves selected nodes with bigger distance. The result (Fig. 5.2) shows that NS and CS have lower end-to-end network latency compared to DS. Violations start early at step 9 and continue to step 14. CS has smaller latency compared to NS, reaching the biggest difference of more than 500ms (approx. 48% improvement) at step 12. This example has more stringent network latency requirement τ_{p_i, p_j} according to its bigger movement distance, compared to example **E1**. Its result has much bigger improvement of the end-to-end latency.

Furthermore, the result of the average execution time of the model compared to the other discussed K8s schedulers is presented in Fig. 5.3. The figure contains one plot with the total number of used pod replicas of the EcoLogic workload on its abscissa axis and the average execution time on its ordinate axis. There are three graphs for

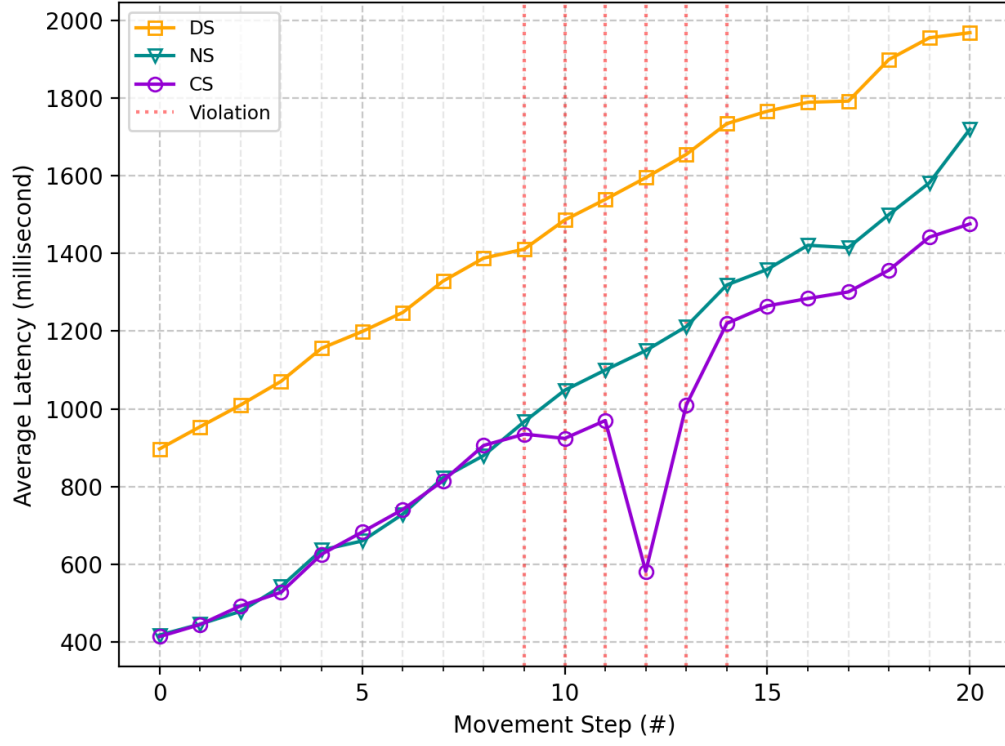


Figure 5.2: Example **E2** with big movement distance $\Delta x = 20\%(160km)$ per step. CS optimizes end-to-end latency by up to 48% starting early from step 9, compared to existing K8s schedulers.

the Kubernetes Default Scheduler (DS), Network-aware Scheduler (NS) and Custom Scheduler (CS) with MILP. There is also a 30 minutes marker depicted in horizontal dashed line. The replicas count of the specific EcoLogic pods that are used are shown in Table 5.1. There are three measurements with total pod replica counts of 4, 8 and 11, respectively. The DS have execution time of 0.06 to 0.12 seconds and NS - 0.07 to 0.13 seconds. In contrast to them, the CS have much bigger execution time of 27 to 117 minutes. This is caused by the MILP model which needs a lot of time to find the most optimal placement solution, proved in the above results Fig. 5.1, Fig. 5.2. Usually many real-time and critical solutions in Cloud/Fog platforms should comply to a maximal execution time of 30 minutes.

5.1 Discussion

The obtained results in Fig. 5.1 and Fig. 5.2 confirm that in dynamic environment with mobile nodes, the model is placing and moving pods on proximal nodes by coping to the movements of the nodes. In this way the workload end-to-end network latency is kept at minimum on runtime and in continual manner. In practice it is possible the replica movements across nodes to cause downtime for the moved replicas, while the not moved ones will continue to work uninterrupted. In this case the replica movement factor δ , can be configured further to reduce the total number of replica movements and their undesirable replica downtimes. Also the period between execution of the model iterations can be configured according to the velocity (frequency) at which the nodes are moving. The period between iterations should be inversely proportional to the movement velocity. The pod network latency requirements (τ_{p_i, p_j}) has to be enough restrictive in order to cause pod placement optimizations, otherwise pods will not be moved.

The execution time results from Fig. 5.3 show that the MILP model takes much time to provide the most optimal solution, which is more than the 30 minutes threshold, used by some real-time critical applications in practice. This means that it can be applied in practical environments by executing it less frequently, for example once/twice daily or less. Nevertheless the MILP model can be used as a benchmark comparing how optimal are other heuristics-based algorithms that could be faster. Additionally it is generic and can be applied in a wide variety of Cloud/Fog use cases. All input variables can be configured by platform engineers in accordance to their infrastructure resource capacities, movement patterns and business application requirements.

The results obtained in this chapter are published in [86].

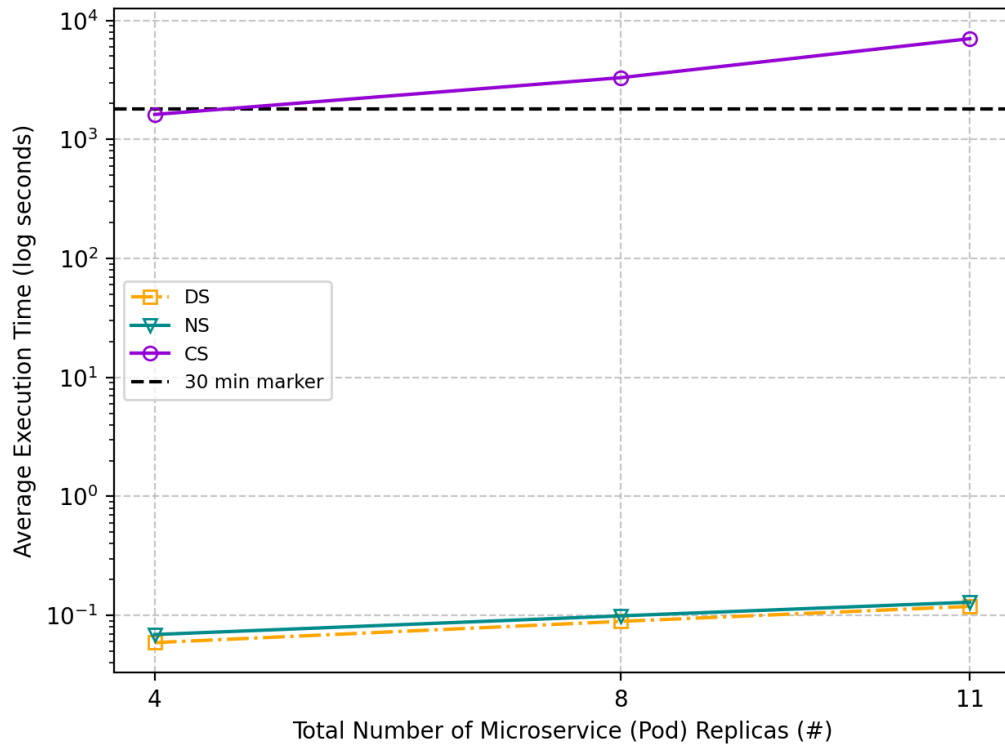


Figure 5.3: Execution time of CS with MILP is much bigger (27 to 117 min), compared to existing K8s schedulers (0.06 to 0.13 sec).

Symbol	Value
b_p	100 Mbps for $\forall p \in P$.
B_n	1 Gbps for $\forall n \in N$.
B_{n_u, n_v}	1 Gbps for $\forall n_u, n_v \in N$.
c_p	100 millicpu for $\forall p \in P$.
C_n	500 millicpu for $\forall n \in N$.
e_p	100 Mb for $\forall p \in P$.
E_n	1 Gb for $\forall n \in N$.
$M_{n,u,z}$	Each node is in different region u and zone z .
M_p^w	All pods are part of a single workload w , representing the EcoLogic sample application. *
N	Cluster is composed of 11 nodes.
P	EcoLogic Pods: Database (P1), Backend (P2), VehicleAgent (P3), Frontend (P4). *
R_p^{req}	1 to 6 **
R_p^{max}	1 to 6 ($R_p^{max} = R_p^{req}$ for $\forall p \in P$). **
T_{n_u, n_v}	10 to 60 milliseconds.
U	11 regions.
W	There is a single workload w , representing the EcoLogic sample application. *
Z	11 zones.
β_{p_i, p_j}	100 Mbps for $\forall p_i, p_j \in P$.
δ	6 ***
τ_{p_i, p_j}	30 milliseconds for $\forall p_i, p_j \in P$.
Ω_{p_i, p_j}^β	1 for $\forall p_i, p_j \in P$.
Ω_{p_i, p_j}^τ	1 for $\forall p_i, p_j \in P$.

Table 5.2: MILP input variables used in examples (E1, E2) and execution time results.

* EcoLogic sample application Pod microservices are described in Table 5.1 and Fig. 1.1.

** Number of Pod replicas used in execution time results and examples are described in Table 5.1.

*** Replica movements across nodes (described in Section 4.2.2) are not limited, because $\delta = R_p^{max}$.

Chapter 6

Conclusion

In recent years the massive adoption of moveable heterogeneous devices with computational capabilities brought the necessity for novel techniques for dynamic resource allocation. This dissertation presented a MILP formulation for container provisioning in Cloud/Fog infrastructures, composed of nodes that move in time and space. It is suitable for dynamic and nondeterministic IoT use cases, which contain moveable entities like vehicles, satellites and aerospace vessels. It uses several objectives attempting to maximize the number of running workloads, reducing its migrations across nodes and improving the end-to-end network latency. Two examples with the practical microservice application EcoLogic [19] were shown. The testbed is emulating a real life movement of computational nodes across geographical regions. The obtained results demonstrate that the solution has a promising potential to bring big improvements in the applications end-to-end network latency in practical Cloud/Edge/Fog infrastructures with moving nodes. Although with big execution time, the MILP model is generic and can serve as a benchmark for research and evaluation of resource allocation algorithms in wide range of dynamic and mobile environments. It can be tailored towards different use cases depending on the infrastructure resource types, workload dependencies and movement patterns.

6.1 Future work

Regarding the Edge/Fog platform presented in Chapter 4 and Chapter 5, the following directions are identified for a future research:

- More experiments with different movement patterns and parameters in order to evaluate further the practical applicability of the system. It is possible to use real life movement patterns from humans, vehicles, satellites, aerospace vessels, etc.
- Configuration of the network latency requirements (τ_{p_i, p_j}) in an automated and

dynamic manner across node's movement steps in order to achieve the best possible end-to-end network latency improvement.

- Enhance the system to be not only reactive – dynamically adapt to movements in nodes, but also proactive – predict movements and adapt accordingly. It is possible to use Artificial Intelligence (AI) for the prediction model.
- The node network infrastructure may be very dynamic and may change before the execution of all MILP objectives is finished in each iteration. This means that it is possible the input variables to represent an outdated state of the infrastructure before MIN_RM and MIN_WL objectives. Enhance the MILP model to work with input variables, which are measured and set before each MILP objective function.
- Usage of a Swarm Computing algorithms for allocation of resources in dynamic mobile Cloud/Fog environment.

Regarding the EcoLogic application, shown in Chapter 2, the following routes for further improvements are identified and will be addressed in the latter stages of the project:

- The engine capacity and carbon dioxide emission parameters are not adequate to appropriately regulate the amount of fuel injected. The solution could be extended to use also other parameters like vehicle weight and performance requirements for better control.
- The measured parameters on cold and hot (with normal working temperature) engine are not deviate between each other. The measurements on a cold engine leads to detection of the outliers. The solution could be optimized to use two separate data sets: data, which is measured on normally working engine and data, which is measured on cold engine.
- Different algorithm for anomaly detection or classification can be used. It can be a supervised machine learning algorithm, which takes the data measured on cold engine as a training dataset, which defines not optimal amount of carbon dioxide emissions.
- Implementation and usage of more lightweight application network protocols such as Constrained Application Protocol (CoAP), Data Distribution Service (DDS) and Advanced Message Queuing Protocol (AMQP).
- Implementation of predictive maintenance algorithm, which makes notifications for potential future failures in vehicles, based on the current and historical data.
- Integration of the project with other third party systems such as emissions trading systems, transportation tax institutions and smart cities. For example, the more eco-friendly drivers could pay smaller taxes in tax systems of countries and city halls. The amount of carbon dioxide emissions could determine the traffic routing by the traffic lights in concrete regions of smart cities.

Acknowledgments

- I would like to thank my supervisor, Assoc. Prof. Hristo Kostadinov, for all his valuable contributions to the dissertation and publications. I will always be grateful for the skills he learned me in math, science, research, engineering and for his continual technical and personal support.
- I would like to thank Dr. Konstantin Delchev, who gave me several Raspberry Pi devices that were involved in the project.
- I would like to thank the Institute of Mathematics and Informatics at Bulgarian Academy of Sciences, for the opportunity to work in their research environment.
- This work was partially supported by the National Science Fund of Bulgaria under Grant KP-06-N32/2-2019.
- This work was supported by the Ministry of Education and Science of Bulgaria (“National Centre for High Performance and Distributed Computing”, grant No. DO1-325/01.12.2023) and Science and Education for Smart Growth Operational Program in Bulgaria (grant No. BG05M2OP001-1.001-0003).
- I am grateful to my parents, brother, and grandmother, for their support and advice when I needed them most.

Appendix A

Public contribution references

The publicly available resources, which are contributed as part of this dissertation are the following:

1. EcoLogic public group of repositories: <https://github.com/ttsokov/vehicle-monitor-controller/> (visited on 06/20/2024).
2. EcoLogic hardware controller repository: <https://github.com/ttsokov/vehicle-monitor-controller-hw-controller/> (visited on 06/20/2024).
3. EcoLogic hardware controller proxy repository: <https://github.com/ttsokov/vehicle-monitor-controller-hw-proxy/> (visited on 06/20/2024).
4. EcoLogic backend repository: <https://github.com/ttsokov/vehicle-monitor-controller-backend/> (visited on 06/20/2024).
5. Public website with shared results and resources: <https://blog.ttsokov.eu/> (visited on 06/20/2024).

Appendix B

Abbreviations

Abbreviation	Description
AMQP	Advanced Message Queuing Protocol
API	Application Programming Interface
CDN	Content Delivery Network
CPU	Central Processing Unit
CoAP	Constrained Application Protocol
CS	Custom Scheduler
DDS	Data Distribution Service
DS	Default Scheduler
HTTPS	Hypertext Transfer Protocol Secure
IoT	Internet of Things
K8s	Kubernetes
LCD	Liquid Crystal Display
LED	Light Emitting Diode
MAX_WD	Maximization of workload deployments
MEC	Multi-access Edge Computing
MILP	Mixed-Integer Linear Programming
MIN_RM	Minimization of microservice replica movements across nodes between iterations
MIN_WL	Minimization of workload end-to-end network latency
MQTT	Message Queue Telemetry Transport Protocol
NS	Network-aware Scheduler
NSA	Non Standalone
O-RAN	Open-Radio Access Network
QoE	Quality of Experience
QoS	Quality of Service
SA	Standalone
SFC	Service Function Chain
VM	Virtual Machine

Appendix C

Terminology

Term	Description
AMQP	Network protocol used in IoT scenarios.
CDN	Content Delivery Network (CDN) is a geo-distributed group of servers, which caches content close to end-users, providing fast and efficient delivery of content (HTML pages, images, videos).
CoAP	Network protocol used in IoT scenarios.
DDS	Network protocol used in IoT scenarios.
Geo-distributed	System that is distributed across multiple geographical locations, providing availability, performance and reliability due to redundancy.
Microservice	Architectural approach where an application is divided into small, independent services that communicate over a network, providing single responsibility, flexibility and scalability.
MILP	Mixed-Integer Linear Programming (MILP) is a type of mathematical program/model, part of discrete mathematics, for solving optimization problems that involve linear equations and inequalities with integer or binary variables.
Mobile	Node or end-user, which is moveable and change its location.
MQTT	Network protocol used in IoT scenarios.
Node	Device, which is part of the infrastructure of a cluster, containing computational resources such as CPU, memory and storage.
O-RAN	Open-Radio Access Network (O-RAN) is a nonproprietary version of the RAN system, allowing interoperability between cellular network equipment provided by different vendors, disaggregating and opening up the RAN deployments.
Scheduler	Software component, which allocates resources (CPU, memory, storage) or containers in a distributed system, typically Cloud/Edge/Fog platform.

Dissertation contributions

The aim of the dissertation is to provide solution for optimal management of resources in modern Cloud/Edge/Fog platforms, enabling execution of complex real-time IoT applications on constrained computational devices, which move in space. It contributes to the better support of scenarios like connected vehicles, spacecraft computing, Augmented Reality (AR), Virtual Reality (VR), real-time audio/video streaming, etc. The solution is implemented in the most popular Cloud/Edge/Fog platform in practice, called Kubernetes [1] and is validated with a complex edge-native IoT application in real practical environment.

The dissertation contains the following contributions:

- The MILP model proposed by Santos et al. [75] is extended with new latency matrix for inter-pod communication variable (τ_{p_i, p_j}), node availability region variable (U) and objective function for minimization of replica movements across mobile nodes (MIN_RM).
- Capacity and demand vectors are replaced by direct variables ($B_n, C_n, E_n, b_p, c_p, e_p$) used into the constraints and objective functions in the mentioned MILP model.
- The execution flow of the above MILP model is modified in order to support dynamic optimization of replica placements with moveable nodes. The flow consists of phases and iterations (Fig. 4.1).
- The MILP optimization model is implemented in real Cloud/Edge/Fog platform (Kubernetes).
- The algorithm and platform are evaluated in a testbed.
- Design and implementation of a real-world edge-native IoT application, called EcoLogic [19], for monitoring and control of carbon emissions from vehicles, suitable to run in smart city environments.
- Validation of the EcoLogic's algorithm for clustering analysis, which detects outlier vehicles that pollute the air excessively.

- Making publicly available open-source repositories with information and source code of the important EcoLogic microservices Appendix [A](#).
- Performing evaluations based on the EcoLogic application. They are making emulation of node movements in the testbed cluster.
- The obtained results from the performed evaluations with the EcoLogic sample application show reduction in the workload's end-to-end latency by up to 48% compared to the latest state of the art.

Approbation of the results

1. T. Tsokov and H. Kostadinov, “System for monitoring and control of vehicle’s carbon emissions using embedded hardwares and cloud applications”, in Service-Oriented Computing - ICSOC 2020 Workshops, H. Hacid, F. Outay, H.-y. Paik, et al., Eds., Cham: Springer International Publishing, 2021, pp. 564-577, isbn: 978-3-030-76352-7. doi: https://doi.org/10.1007/978-3-030-76352-7_50. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-030-76352-7_50; SJR (Q2).
2. T. Tsokov and H. Kostadinov, “Dynamic network-aware container allocation in Cloud/Fog computing with mobile nodes”, Internet of Things, p. 101 211, 2024, issn: 2542-6605. doi: <https://doi.org/10.1016/j.iot.2024.101211>. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S2542660524001525>; Impact factor (Q1).

Presentations at scientific forums

1. T. Tsokov and H. Kostadinov, "Iot System for Monitoring and Control of Vehicle's Carbon Emissions Using Embedded Hardwares and Cloud Applications", 32nd National Seminar on Coding Theory "Acad. Stefan Dodunekov", Chiflik, Troyan District, 10 October 2020.
2. T. Tsokov and H. Kostadinov, "System for Monitoring and Control of Vehicle's Carbon Emissions Using Embedded Hardwares and Cloud Applications", International Workshop on Artificial Intelligence in the IoT Security Services (Service-Oriented Computing - ICSOC 2020 Workshops), Dubai, 14-17 December 2020.
3. T. Tsokov, "IoT fog platform", Problems and Methods Related to Coding Theory, Sofia, 08 February 2022.

Bibliography

- [1] Kubernetes. “Kubernetes.” (2022-04-12), [Online]. Available: <https://kubernetes.io/> (visited on 04/12/2022).
- [2] A. Haider, R. Potter, and A. Nakao, “Challenges in resource allocation in network virtualization,” in *20th ITC specialist seminar*, ITC, vol. 18, Jan. 2009. [Online]. Available: <https://api.semanticscholar.org/CorpusID:10681804>.
- [3] N. Kratzke and P.-C. Quint, “Understanding cloud-native applications after 10 years of cloud computing - a systematic mapping study,” *Journal of Systems and Software*, vol. 126, pp. 1–16, 2017, ISSN: 0164-1212. DOI: <https://doi.org/10.1016/j.jss.2017.01.001>. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0164121217300018>.
- [4] “Edge-native application principles.” (2024-06-21), [Online]. Available: <https://www.cncf.io/reports/edge-native-application-design-behaviors-whitepaper/> (visited on 06/21/2024).
- [5] M. I. Rochman, V. Sathya, D. Fernandez, *et al.*, “A comprehensive analysis of the coverage and performance of 4g and 5g deployments,” *Computer Networks*, vol. 237, p. 110 060, 2023, ISSN: 1389-1286. DOI: <https://doi.org/10.1016/j.comnet.2023.110060>. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1389128623005054>.
- [6] M. S. Elbamby, C. Perfecto, M. Bennis, and K. Doppler, “Toward low-latency and ultra-reliable virtual reality,” *IEEE Network*, vol. 32, no. 2, pp. 78–84, Mar. 2018, ISSN: 1558-156X. DOI: [10.1109/MNET.2018.1700268](https://doi.org/10.1109/MNET.2018.1700268).
- [7] J. B. Jr., B. Costa, L. R. Carvalho, M. J. F. Rosa, and A. Araujo, “Computational resource allocation in fog computing: A comprehensive survey,” *ACM Comput. Surv.*, Mar. 2023, Just Accepted, ISSN: 0360-0300. DOI: [10.1145/3586181](https://doi.org/10.1145/3586181). [Online]. Available: <https://doi.org/10.1145/3586181>.
- [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, Jan. 2022, ISSN: 0360-0300. DOI: [10.1145/3486221](https://doi.org/10.1145/3486221). [Online]. Available: <https://doi.org/10.1145/3486221>.

- [9] K. Kaur, A. Singh, and A. Sharma, "A systematic review on resource provisioning in fog computing," *Transactions on Emerging Telecommunications Technologies*, vol. 34, no. 4, e4731, 2023. DOI: <https://doi.org/10.1002/ett.4731>. eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1002/ett.4731>. [Online]. Available: <https://onlinelibrary.wiley.com/doi/abs/10.1002/ett.4731>.
- [10] A. Shakarami, H. Shakarami, M. Ghobaei-Arani, E. Nikougoftar, and M. Faraji-Mehmandar, "Resource provisioning in edge/fog computing: A comprehensive and systematic review," *Journal of Systems Architecture*, vol. 122, p. 102362, 2022, ISSN: 1383-7621. DOI: <https://doi.org/10.1016/j.sysarc.2021.102362>. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1383762121002526>.
- [11] R. N. Calheiros, R. Ranjan, C. A. F. D. Rose, and R. Buyya, *Cloudsim: A novel framework for modeling and simulation of cloud computing infrastructures and services*, 2009. arXiv: [0903.2525](https://arxiv.org/abs/0903.2525) [cs.DC].
- [12] C. Sonmez, A. Ozgovde, and C. Ersoy, "Edgecloudsim: An environment for performance evaluation of edge computing systems," *2017 Second International Conference on Fog and Mobile Edge Computing (FMEC)*, pp. 39–44, 2017.
- [13] H. Gupta, A. V. Dastjerdi, S. K. Ghosh, and R. Buyya, *Ifogsim: A toolkit for modeling and simulation of resource management techniques in internet of things, edge and fog computing environments*, 2016. arXiv: [1606.02007](https://arxiv.org/abs/1606.02007) [cs.DC].
- [14] G. Beniwal and A. Singhrova, "A systematic literature review on iot gateways," *Journal of King Saud University - Computer and Information Sciences*, vol. 34, no. 10, Part B, pp. 9541–9563, 2022, ISSN: 1319-1578. DOI: <https://doi.org/10.1016/j.jksuci.2021.11.007>. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1319157821003219>.
- [15] 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*, ser. MCC '12, Helsinki, Finland: Association for Computing Machinery, 2012, pp. 13–16, ISBN: 9781450315197. DOI: [10.1145/2342509.2342513](https://doi.org/10.1145/2342509.2342513). [Online]. Available: <https://doi.org/10.1145/2342509.2342513>.
- [16] M. Agiwal, A. Roy, and N. Saxena, "Next generation 5g wireless networks: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 3, pp. 1617–1655, 2016. DOI: [10.1109/COMST.2016.2532458](https://doi.org/10.1109/COMST.2016.2532458).

- [17] A. Ahmed, H. Arkian, D. Battulga, *et al.*, *Fog computing applications: Taxonomy and requirements*, 2019. DOI: <https://doi.org/10.48550/arXiv.1907.11621>. arXiv: 1907.11621 [cs.DC].
- [18] D. Chatzopoulos, C. Bermejo, Z. Huang, and P. Hui, “Mobile augmented reality survey: From where we are to where we go,” *IEEE Access*, vol. 5, pp. 6917–6950, 2017. DOI: [10.1109/ACCESS.2017.2698164](https://doi.org/10.1109/ACCESS.2017.2698164).
- [19] T. Tsokov and H. Kostadinov, “System for monitoring and control of vehicle’s carbon emissions using embedded hardwares and cloud applications,” in *Service-Oriented Computing – ICSOC 2020 Workshops*, H. Hacid, F. Outay, H.-y. Paik, *et al.*, Eds., Cham: Springer International Publishing, 2021, pp. 564–577, ISBN: 978-3-030-76352-7. DOI: [10.1007/978-3-030-76352-7_50](https://doi.org/10.1007/978-3-030-76352-7_50). [Online]. Available: https://doi.org/10.1007/978-3-030-76352-7_50.
- [20] A. Laghari, A. Jumani, and R. Laghari, “Review and state of art of fog computing,” *Archives of Computational Methods in Engineering*, vol. 28, Feb. 2021. DOI: [10.1007/s11831-020-09517-y](https://doi.org/10.1007/s11831-020-09517-y).
- [21] R. Das and M. Muhammad Inuwa, “A review on fog computing: Issues, characteristics, challenges, and potential applications,” *Telematics and Informatics Reports*, vol. 10, p. 100 049, Feb. 2023. DOI: [10.1016/j.teler.2023.100049](https://doi.org/10.1016/j.teler.2023.100049).
- [22] R. Mahmud, R. Kotagiri, and R. Buyya, “Fog computing: A taxonomy, survey and future directions,” in *Internet of Everything: Algorithms, Methodologies, Technologies and Perspectives*, B. Di Martino, K.-C. Li, L. T. Yang, and A. Esposito, Eds. Singapore: Springer Singapore, 2018, pp. 103–130, ISBN: 978-981-10-5861-5. DOI: [10.1007/978-981-10-5861-5_5](https://doi.org/10.1007/978-981-10-5861-5_5). [Online]. Available: https://doi.org/10.1007/978-981-10-5861-5_5.
- [23] S. A. Noghabi, L. Cox, S. Agarwal, and G. Ananthanarayanan, “The emerging landscape of edge computing,” *GetMobile: Mobile Comp. and Comm.*, vol. 23, no. 4, pp. 11–20, May 2020, ISSN: 2375-0529. DOI: [10.1145/3400713.3400717](https://doi.org/10.1145/3400713.3400717). [Online]. Available: <https://doi.org/10.1145/3400713.3400717>.
- [24] A. Ahmed and E. Ahmed, “A survey on mobile edge computing,” in *2016 10th International Conference on Intelligent Systems and Control (ISCO)*, 2016, pp. 1–8. DOI: [10.1109/ISCO.2016.7727082](https://doi.org/10.1109/ISCO.2016.7727082).
- [25] H. Jin, M. A. Gregory, and S. Li, “A review of intelligent computation offloading in multiaccess edge computing,” *IEEE Access*, vol. 10, pp. 71 481–71 495, 2022. DOI: [10.1109/ACCESS.2022.3187701](https://doi.org/10.1109/ACCESS.2022.3187701).
- [26] “Microsoft azure edge zone.” (2024-06-21), [Online]. Available: <https://learn.microsoft.com/en-us/azure/private-multi-access-edge-compute-mec/overview/> (visited on 06/21/2024).
- [27] “Google distributed cloud.” (2024-06-21), [Online]. Available: <https://cloud.google.com/distributed-cloud/> (visited on 06/21/2024).

- [28] “Amazon web services local zones.” (2024-06-21), [Online]. Available: <https://aws.amazon.com/about-aws/global-infrastructure/localzones/> (visited on 06/21/2024).
- [29] “Alibaba link iot edge.” (2024-06-21), [Online]. Available: <https://www.alibabacloud.com/help/en/link-iot-edge/latest/what-is-link-iot-edge/> (visited on 06/21/2024).
- [30] M. Xu, Z. Fu, X. Ma, *et al.*, *From cloud to edge: A first look at public edge platforms*, 2021. DOI: <https://doi.org/10.48550/arXiv.2109.03395>. arXiv: [2109.03395 \[cs.NI\]](https://arxiv.org/abs/2109.03395).
- [31] K.-K. Yap, M. Motiwala, J. Rahe, *et al.*, “Taking the edge off with espresso: Scale, reliability and programmability for global internet peering,” in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM ’17, Los Angeles, CA, USA: Association for Computing Machinery, 2017, pp. 432–445, ISBN: 9781450346535. DOI: [10.1145/3098822.3098854](https://doi.org/10.1145/3098822.3098854). [Online]. Available: <https://doi.org/10.1145/3098822.3098854>.
- [32] B. Schlinker, I. Cunha, Y.-C. Chiu, S. Sundaresan, and E. Katz-Bassett, “Internet performance from facebook’s edge,” in *Proceedings of the Internet Measurement Conference*, ser. IMC ’19, Amsterdam, Netherlands: Association for Computing Machinery, 2019, pp. 179–194, ISBN: 9781450369480. DOI: [10.1145/3355369.3355567](https://doi.org/10.1145/3355369.3355567). [Online]. Available: <https://doi.org/10.1145/3355369.3355567>.
- [33] “Cloudflare workers.” (2024-06-21), [Online]. Available: <https://www.cloudflare.com/products/workers/> (visited on 06/21/2024).
- [34] “Akamai edge workers.” (2024-06-21), [Online]. Available: <https://www.akamai.com/products/serverless-computing-edgeworkers/> (visited on 06/21/2024).
- [35] “Fastly edge compute.” (2024-06-21), [Online]. Available: <https://www.fastly.com/products/edge-compute/> (visited on 06/21/2024).
- [36] Z. Jin, L. Pan, and S. Liu, “Randomized online edge service renting: Extending cloud-based cdn to edge environments,” *Knowledge-Based Systems*, vol. 257, p. 109957, 2022, ISSN: 0950-7051. DOI: <https://doi.org/10.1016/j.knosys.2022.109957>. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0950705122010504>.
- [37] “Akamai edge workers in macrometa.” (2024-06-21), [Online]. Available: <https://www.macrometa.com/docs/akamai/> (visited on 06/21/2024).

- [38] A. Bhardwaj and C. R. Krishna, “Virtualization in cloud computing: Moving from hypervisor to containerization—a survey,” *Arabian Journal for Science and Engineering*, vol. 46, no. 9, pp. 8585–8601, Apr. 2021. DOI: [10.1007/s13369-021-05553-3](https://doi.org/10.1007/s13369-021-05553-3). [Online]. Available: <https://doi.org/10.1007/s13369-021-05553-3>.
- [39] ARM. “Arm cpu architecture.” (2024-06-21), [Online]. Available: <https://www.arm.com/architecture/cpu/> (visited on 06/21/2024).
- [40] C. Dall, S.-W. Li, J. T. Lim, J. Nieh, and G. Koloventzos, “Arm virtualization: Performance and architectural implications,” *SIGARCH Comput. Archit. News*, vol. 44, no. 3, pp. 304–316, Jun. 2016, ISSN: 0163-5964. DOI: [10.1145/3007787.3001169](https://doi.org/10.1145/3007787.3001169). [Online]. Available: <https://doi.org/10.1145/3007787.3001169>.
- [41] K. Sivaprakasam, P. Sriramalakshmi, P. Singh, and M. Bhaskar, “Chapter 4 - an overview of low power hardware architecture for edge computing devices,” in *5G IoT and Edge Computing for Smart Healthcare*, ser. Intelligent Data-Centric Systems, A. K. Bhoi, V. H. C. de Albuquerque, S. N. Sur, and P. Barsocchi, Eds., Academic Press, 2022, pp. 89–109, ISBN: 978-0-323-90548-0. DOI: <https://doi.org/10.1016/B978-0-323-90548-0.00004-8>. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/B9780323905480000048>.
- [42] J. Alonso, L. Orue-Echevarria, V. Casola, *et al.*, “Understanding the challenges and novel architectural models of multi-cloud native applications – a systematic literature review,” *J. Cloud Comput.*, vol. 12, no. 1, Jan. 2023, ISSN: 2192-113X. DOI: [10.1186/s13677-022-00367-6](https://doi.org/10.1186/s13677-022-00367-6). [Online]. Available: <https://doi.org/10.1186/s13677-022-00367-6>.
- [43] “The twelve-factor app.” (2024-06-21), [Online]. Available: <https://12factor.net/> (visited on 06/21/2024).
- [44] M. Jansen, A. Al-Dulaimy, A. V. Papadopoulos, A. K. Trivedi, and A. Iosup, “The spec-rg reference architecture for the compute continuum,” *2023 IEEE/ACM 23rd International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, pp. 469–484, 2022. [Online]. Available: <https://api.semanticscholar.org/CorpusID:257280275>.
- [45] 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, ISSN: 1932-4537. DOI: [10.1109/TNSM.2017.2788945](https://doi.org/10.1109/TNSM.2017.2788945).
- [46] Y. Shao, C. Li, and H. Tang, “A data replica placement strategy for iot workflows in collaborative edge and cloud environments,” *Computer Networks*, vol. 148, pp. 46–59, 2019, ISSN: 1389-1286. DOI: <https://doi.org/10.1016/j.comnet.2018.10.017>. [Online]. Available:

<https://www.sciencedirect.com/science/article/pii/S1389128618311460>.

- [47] V. B. C. Souza, W. Ramírez, X. Masip-Bruin, E. Marín-Tordera, G.-J. Ren, and G. Tashakor, "Handling service allocation in combined fog-cloud scenarios," *2016 IEEE International Conference on Communications (ICC)*, pp. 1–5, 2016.
- [48] K. Zhang, M. Peng, and Y. Sun, "Delay-optimized resource allocation in fog-based vehicular networks," *IEEE Internet of Things Journal*, vol. 8, no. 3, pp. 1347–1357, Feb. 2021, ISSN: 2327-4662. DOI: [10.1109/JIOT.2020.3010861](https://doi.org/10.1109/JIOT.2020.3010861).
- [49] S. Hassan, I. Ahmad, A. Rehman, S. Hussien, and H. Hamam, "Design of resource-aware load allocation for heterogeneous fog computing environments," *Wireless Communications and Mobile Computing*, vol. 2022, Jun. 2022. DOI: [10.1155/2022/3543640](https://doi.org/10.1155/2022/3543640).
- [50] L. Li, Q. Guan, L. Jin, and M. Guo, "Resource allocation and task offloading for heterogeneous real-time tasks with uncertain duration time in a fog queueing system," *IEEE Access*, vol. 7, pp. 9912–9925, 2019, ISSN: 2169-3536. DOI: [10.1109/ACCESS.2019.2891130](https://doi.org/10.1109/ACCESS.2019.2891130).
- [51] F. Xia, L. T. Yang, L. Wang, and A. Vinel, "Internet of things," *International Journal of Communication Systems*, vol. 25, no. 9, pp. 1101–1102, 2012. DOI: <https://doi.org/10.1002/dac.2417>. eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1002/dac.2417>. [Online]. Available: <https://onlinelibrary.wiley.com/doi/abs/10.1002/dac.2417>.
- [52] D. Evans, "The internet of things. how the next evolution of the internet is changing everything," *Cisco IBSG*, 2011. [Online]. Available: https://www.cisco.com/c/dam/en_us/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf.
- [53] M. Carignani, S. Ferrini, M. Petracca, M. Falcitelli, and P. Pagano, "A prototype bridge between automotive and the iot," in *Proceedings of the 2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, ser. WF-IOT '15, USA: IEEE Computer Society, 2015, pp. 12–17, ISBN: 9781509003662. DOI: [10.1109/WF-IoT.2015.7389019](https://doi.org/10.1109/WF-IoT.2015.7389019). [Online]. Available: <https://doi.org/10.1109/WF-IoT.2015.7389019>.
- [54] A. Fehske, G. Fettweis, J. Malmudin, and G. Biczok, "The global footprint of mobile communications: The ecological and economic perspective," *IEEE Communications Magazine*, vol. 49, no. 8, pp. 55–62, Aug. 2011, ISSN: 1558-1896. DOI: [10.1109/MCOM.2011.5978416](https://doi.org/10.1109/MCOM.2011.5978416).
- [55] A. Munir, P. Kansakar, and S. U. Khan, "Ifciot: Integrated fog cloud iot: A novel architectural paradigm for the future internet of things," *IEEE Consumer Electronics Magazine*, vol. 6, no. 3, pp. 74–82, Jul. 2017, ISSN: 2162-2256. DOI: [10.1109/MCE.2017.2684981](https://doi.org/10.1109/MCE.2017.2684981).

- [56] M. S. Henriques and N. K. Vernekar, “Using symmetric and asymmetric cryptography to secure communication between devices in iot,” in *2017 International Conference on IoT and Application (ICIOT)*, May 2017, pp. 1–4. DOI: [10.1109/ICIOTA.2017.8073643](https://doi.org/10.1109/ICIOTA.2017.8073643).
- [57] F. Färber, N. May, W. Lehner, *et al.*, “The sap hana database - an architecture overview,” *Bulletin of the Technical Committee on Data Engineering / IEEE Computer Society*, vol. 35, no. 1, pp. 28–33, 2012.
- [58] P. Mugglestone. “Sap hana academy.” (2014-05-13), [Online]. Available: <https://github.com/saphanaacademy/PAL/tree/master/Source%20Data/PAL> (visited on 06/07/2024).
- [59] A. Beloglazov, J. Abawajy, and R. Buyya, “Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing,” *Future Generation Computer Systems*, vol. 28, no. 5, pp. 755–768, 2012, Special Section: Energy efficiency in large-scale distributed systems, ISSN: 0167-739X. DOI: <https://doi.org/10.1016/j.future.2011.04.017>. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0167739X11000689>.
- [60] J. Zhang, H. Huang, and X. Wang, “Resource provision algorithms in cloud computing: A survey,” *Journal of Network and Computer Applications*, vol. 64, pp. 23–42, 2016, ISSN: 1084-8045. DOI: <https://doi.org/10.1016/j.jnca.2015.12.018>. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1084804516000643>.
- [61] A. Beltre, P. Saha, and M. Govindaraju, “Kubesphere: An approach to multi-tenant fair scheduling for kubernetes clusters,” in *2019 IEEE Cloud Summit*, Aug. 2019, pp. 14–20. DOI: [10.1109/CloudSummit47114.2019.00009](https://doi.org/10.1109/CloudSummit47114.2019.00009).
- [62] D. Crankshaw, G.-E. Sela, X. Mo, *et al.*, “Inferline: Latency-aware provisioning and scaling for prediction serving pipelines,” in *Proceedings of the 11th ACM Symposium on Cloud Computing*, ser. SoCC ’20, Virtual Event, USA: Association for Computing Machinery, 2020, pp. 477–491, ISBN: 9781450381376. DOI: [10.1145/3419111.3421285](https://doi.org/10.1145/3419111.3421285). [Online]. Available: <https://doi.org/10.1145/3419111.3421285>.
- [63] A. J. Fahs and G. Pierre, “Tail-latency-aware fog application replica placement,” in *Service-Oriented Computing: 18th International Conference, ICSOC 2020, Dubai, United Arab Emirates, December 14–17, 2020, Proceedings*, Dubai, United Arab Emirates: Springer-Verlag, 2020, pp. 508–524, ISBN: 978-3-030-65309-5. DOI: [10.1007/978-3-030-65310-1_37](https://doi.org/10.1007/978-3-030-65310-1_37). [Online]. Available: https://doi.org/10.1007/978-3-030-65310-1_37.

- [64] A. J. Fahs, G. Pierre, and E. Elmroth, “Voilà: Tail-latency-aware fog application replicas autoscaler,” in *2020 28th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2020, pp. 1–8. DOI: [10.1109/MASCOTS50786.2020.9285953](https://doi.org/10.1109/MASCOTS50786.2020.9285953).
- [65] A. Chung, S. Krishnan, K. Karanasos, C. Curino, and G. R. Ganger, “Unearthing inter-job dependencies for better cluster scheduling,” in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, USENIX Association, Nov. 2020, pp. 1205–1223, ISBN: 978-1-939133-19-9. [Online]. Available: <https://www.usenix.org/conference/osdi20/presentation/chung>.
- [66] A. Qiao, S. K. Choe, S. J. Subramanya, *et al.*, “Pollux: Co-adaptive cluster scheduling for goodput-optimized deep learning,” in *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*, USENIX Association, Jul. 2021, pp. 1–18, ISBN: 978-1-939133-22-9. [Online]. Available: <https://www.usenix.org/conference/osdi21/presentation/qiao>.
- [67] Y. He, W. Cai, P. Zhou, *et al.*, “Beamer: Stage-aware coflow scheduling to accelerate hyper-parameter tuning in deep learning clusters,” *IEEE Transactions on Network and Service Management*, vol. 19, no. 2, pp. 1083–1097, Jun. 2022, ISSN: 1932-4537. DOI: [10.1109/TNSM.2021.3132361](https://doi.org/10.1109/TNSM.2021.3132361).
- [68] X. Li, J. Zhou, X. Wei, *et al.*, “Topology-aware scheduling framework for microservice applications in cloud,” *IEEE Transactions on Parallel and Distributed Systems*, vol. PP, pp. 1–17, May 2023. DOI: [10.1109/TPDS.2023.3238751](https://doi.org/10.1109/TPDS.2023.3238751).
- [69] D. Narayanan, K. Santhanam, F. Kazhamiaka, A. Phanishayee, and M. Zaharia, “Heterogeneity-aware cluster scheduling policies for deep learning workloads,” in *Proceedings of the 14th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI’20, USA: USENIX Association, 2020, ISBN: 978-1-939133-19-9.
- [70] V. Rao, V. Singh, K. S. Goutham, *et al.*, “Scheduling microservice containers on large core machines through placement and coalescing,” in *Job Scheduling Strategies for Parallel Processing: 24th International Workshop, JSSPP 2021, Virtual Event, May 21, 2021, Revised Selected Papers*, Berlin, Heidelberg: Springer-Verlag, 2021, pp. 80–100, ISBN: 978-3-030-88223-5. DOI: [10.1007/978-3-030-88224-2_5](https://doi.org/10.1007/978-3-030-88224-2_5). [Online]. Available: https://doi.org/10.1007/978-3-030-88224-2_5.
- [71] H. Zhu, K. Kaffes, Z. Chen, *et al.*, “RackSched: A Microsecond-Scale scheduler for Rack-Scale computers,” in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, USENIX Association, Nov. 2020, pp. 1225–1240, ISBN: 978-1-939133-19-9. [Online]. Available: <https://www.usenix.org/conference/osdi20/presentation/zhu>.

- [72] D. Haja, B. Vass, and L. Toka, “Towards making big data applications network-aware in edge-cloud systems,” in *2019 IEEE 8th International Conference on Cloud Networking (CloudNet)*, 2019, pp. 1–6. DOI: [10.1109/CloudNet47604.2019.9064109](https://doi.org/10.1109/CloudNet47604.2019.9064109).
- [73] J. Santos, T. Wauters, B. Volckaert, and F. De Turck, “Towards end-to-end resource provisioning in fog computing over low power wide area networks,” *Journal of Network and Computer Applications*, vol. 175, p. 102915, 2021, ISSN: 1084-8045. DOI: <https://doi.org/10.1016/j.jnca.2020.102915>. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S108480452030374X>.
- [74] Pereira dos Santos, José Pedro and Vanhove, Thomas and Sebrechts, Merlijn and Dupont, Thomas and Kerckhove, Wannes and Braem, Bart and Van Seghbroeck, Gregory and Wauters, Tim and Leroux, Philip and Latré, Steven and Volckaert, Bruno and De Turck, Filip, “City of things : enabling resource provisioning in smart cities,” eng, *IEEE COMMUNICATIONS MAGAZINE*, vol. 56, no. 7, 177–183, 2018, ISSN: 0163-6804. [Online]. Available: <http://dx.doi.org/10.1109/MCOM.2018.1701322>.
- [75] J. Santos, C. Wang, T. Wauters, and F. D. Turck, “Diktyo: Network-aware scheduling in container-based clouds,” *IEEE Transactions on Network and Service Management*, pp. 1–1, 2023. DOI: [10.1109/TNSM.2023.3271415](https://doi.org/10.1109/TNSM.2023.3271415).
- [76] I. Cloud. “Ibm cloud. hybrid. open. resilient. your platform and partner for digital transformation.” (2024-06-21), [Online]. Available: <https://www.ibm.com/cloud/> (visited on 06/21/2024).
- [77] A. Santoyo-González and C. Cervelló-Pastor, “Network-aware placement optimization for edge computing infrastructure under 5g,” *IEEE Access*, vol. 8, pp. 56 015–56 028, 2020, ISSN: 2169-3536. DOI: [10.1109/ACCESS.2020.2982241](https://doi.org/10.1109/ACCESS.2020.2982241).
- [78] 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, Jun. 2020, ISSN: 0360-0300. DOI: [10.1145/3391196](https://doi.org/10.1145/3391196). [Online]. Available: <https://doi.org/10.1145/3391196>.
- [79] C. Li, Y. Wang, H. Tang, Y. Zhang, Y. Xin, and Y. Luo, “Flexible replica placement for enhancing the availability in edge computing environment,” *Computer Communications*, vol. 146, pp. 1–14, 2019, ISSN: 0140-3664. DOI: <https://doi.org/10.1016/j.comcom.2019.07.013>. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0140366418308296>.
- [80] Y.-J. Yu, T.-C. Chiu, A.-C. Pang, M.-F. Chen, and J. Liu, “Virtual machine placement for backhaul traffic minimization in fog radio access networks,” in *2017 IEEE International Conference on Communications (ICC)*, 2017, pp. 1–7. DOI: [10.1109/ICC.2017.7996500](https://doi.org/10.1109/ICC.2017.7996500).

- [81] T. Ouyang, Z. Zhou, and X. Chen, “Follow me at the edge: Mobility-aware dynamic service placement for mobile edge computing,” *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 10, pp. 2333–2345, 2018. DOI: [10.1109/JSAC.2018.2869954](https://doi.org/10.1109/JSAC.2018.2869954).
- [82] S. Wang, R. Urgaonkar, T. He, K. Chan, M. Zafer, and K. K. Leung, “Dynamic service placement for mobile micro-clouds with predicted future costs,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 4, pp. 1002–1016, Apr. 2017, ISSN: 1558-2183. DOI: [10.1109/TPDS.2016.2604814](https://doi.org/10.1109/TPDS.2016.2604814).
- [83] 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)*, 2019, pp. 1–10. DOI: [10.1109/CFEC.2019.8733152](https://doi.org/10.1109/CFEC.2019.8733152).
- [84] K. Toczé, A. J. Fahs, G. Pierre, and S. Nadjm-Tehrani, “Violinn: Proximity-aware edge placement with dynamic and elastic resource provisioning,” *ACM Trans. Internet Things*, vol. 4, no. 1, Feb. 2023, ISSN: 2691-1914. DOI: [10.1145/3573125](https://doi.org/10.1145/3573125). [Online]. Available: <https://doi.org/10.1145/3573125>.
- [85] J. B. Kelsey Hightower Brendan Burns, *Kubernetes: up and running: dive into the future of infrastructure*. O’Reilly Media, 2019.
- [86] T. Tsokov and H. Kostadinov, “Dynamic network-aware container allocation in cloud/fog computing with mobile nodes,” *Internet of Things*, p. 101 211, 2024, ISSN: 2542-6605. DOI: <https://doi.org/10.1016/j.iot.2024.101211>. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S2542660524001525>.