Bulgarian Academy of Sciences
Institute of Mathematics and Informatics

Tsvetan Krasimirov Tsokov

IoT platforms and protocols

Summary
of 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

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 Internet of Things (IoT) scenarios and incorporate infrastructure clusters with constriant 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 Quality of Service (QoS) and Quality of Experience (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, augmented/virtual reality (AR/VR), 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 Mixed-Integer Linear Programming (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). 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.

Chapter 1

Overview of Edge/Fog distributed systems

The low network latency and mobility requirements are crucial for large range of applications in order to support certain level of QoS and QoE. Such kinds of applications are for example augmented and virtual reality, autonomous driving vehicles and traffic emissions control in smart cities [1]. 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 [2]–[4], providing lower latency and bigger bandwidth.

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 [5]. On the hardware level, ARM CPU architecture is becoming increasingly used for servers, but its virtualization support is more limited than the prevalent x86 architecture [6]. It's known that Edge network layer can be composed of nodes with low-power and ARM CPU architecture [7]. 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 [8] principles and Twelve-Factor App methodology. Right now a new novel principles are emerging in practice called Edge-native [9], [10]. 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. 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 [1] 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 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. [11]–[14]) or heuristic-based algorithms (e.g. [15], [16]) that usually are assessed on simulators like CloudSim [17], EdgeCloudSim [18] and iFogSim [19]. 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**, **application microservice dependencies support**, **dynamicity**, **mobility**, **network latency awareness**, **ARM64 CPU architecture support** and **evaluation type**.

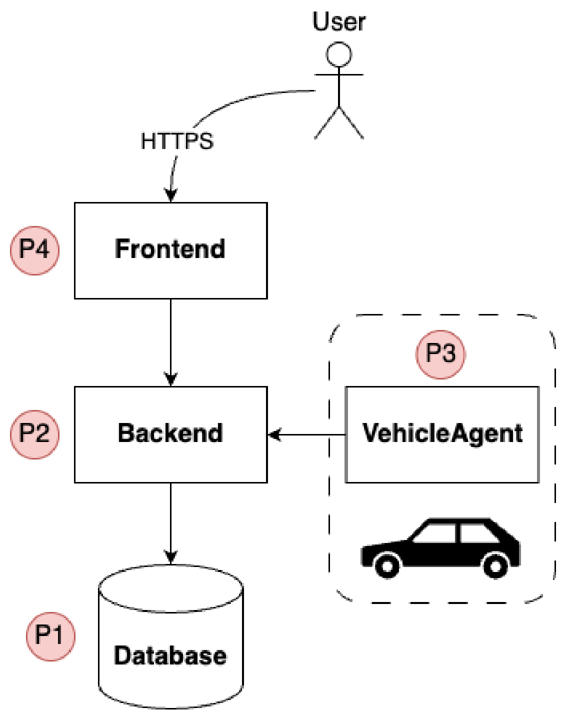

Figure 1.1: EcoLogic application [1] microservice architecture.

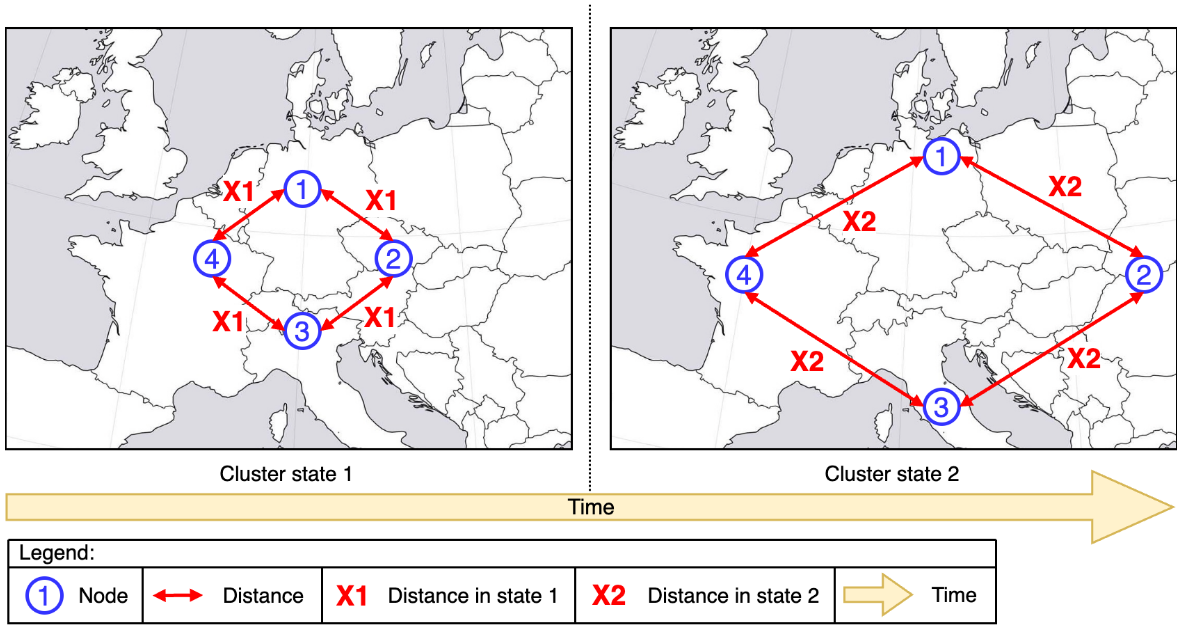

Figure 1.2: Movement of nodes, part of geo-distributed (multi-region) edge cluster.

Chapter 2

EcoLogic IoT application

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.

The dissertation proposes a solution for real-time monitoring and detection of rising levels of carbon emissions from vehicles, called EcoLogic. 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. The data is transferred to a cloud-based applications, where it's stored and analysed. 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. The repositories contain information on how to setup the projects and deploy them on hardware device and server or in the cloud.

2.1 System design

The EcoLogic system is composed of two big parts: edge-native hardware modules and cloud microservice applications. The hardware modules are located in vehicles and the microservice applications are deployed on a cloud platform. Simplified architecture of the system is shown in Fig. 1.1. It becomes a standard for IoT systems to incorporate Cloud and Fog computing platforms in its architectures 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 engine physical parameters such as air pressure, air temperature and fuel mixture. It can also extract parameters from the onboard diagnostic system of the vehicle. The data is sent to applications in the cloud platform.

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.

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 (K-Means) 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 cloud applications provide web user interface, on the base of HTML5, JavaScript and CSS resources. Its endpoint is publicly available and accessible by clients.

2.2 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 cloud platform and services that are configured for the case study are as follows:

- All described cloud applications are deployed into SAP Cloud platform.
- The data is stored in a HANA relational database, provided as a backing service from the cloud platform.
- The Analytics application is represented by a SAP cloud platform predictive service, which provides an algorithm for clustering analysis used for anomaly detection.

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 [20]. It has data for customers with the following parameters: id, name, lifespend, newspend, income and loyalty. The returned results from execution of the clustering analysis with anomaly detection on the test dataset are shown in Fig. 2.1.

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. 2.1 as Outlier, represents a data point, which is located too

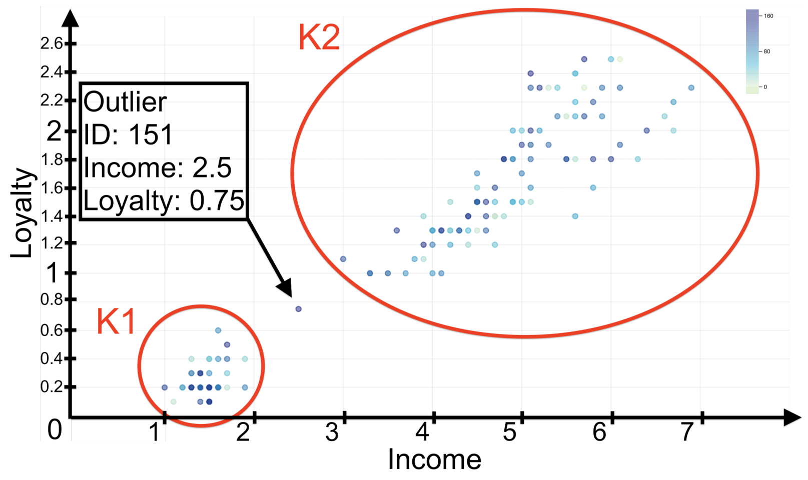

Figure 2.1: Identified clusters in dataset with known anomalies.

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.

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. 2.2. 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. 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 result obtained from the algorithm

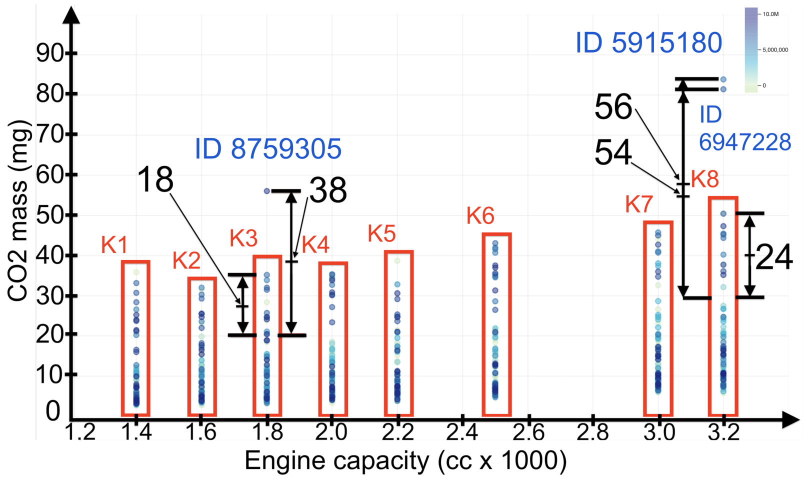

Figure 2.2: Identified clusters in real dataset ingested from vehicle.

is feasible and proves its correctness on the real dataset.

Chapter 3

Comparative analysis of resource allocation in Edge/Fog

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 such computing platforms an important research topic.

The resource provisioning problem in the Edge/Fog computing domain is studied and classified in the Bachiega et al. survey [21]. 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.

The availability and isolation of resources in Fog computing is usually achieved by virtualization model. It is classified in [21] 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 [22] and reducing end-to-end (E2E) tail latency [23]–[25].

Another important topic in Fog is the infrastructure topology awareness. It is considered in Li et al. [26], where Microservice-Oriented Topology-Aware Scheduling Framework (MOTAS) is presented. Another topology aware scheduler, named Gavel, is proposed by Narayanan et al. [27], which 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 [28]. Also a mechanism for dynamic coalescing of hot services into a single container is shown there.

The problem of resource provisioning in Fog-Cloud environments with containerized services is solved by Santos et al. in [29], where MILP formulation is described. The formulation considers SFC concepts, several LPWAN technologies and objectives among which is also network latency.

Dynamic replica placement algorithms for data services are presented in Aral et al. [11], Shao et al. [12] and Li et al. [30]. 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.

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. [31].

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 [24] and Voila [25]. They support stationary and mobile nodes.

A distributed Edge orchestration framework, called ORCH, is presented by Toczé et al. [32]. 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.

An extension of the two related works [25], [32], called VioLinn, is provided by Toczé et al. [33]. An optimization formulation for task placement, service placement and Edge device provisioning in Edge/Fog scenario has been presented.

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 [34]. 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 variaty 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 produce 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 model formulates the set of nodes in a Kubernetes-based Cloud/Fog cluster as a set $N = \{n_1, n_2, \ldots\}$. 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:

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

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

Microservices are called pods in the K8s model. Each workload w is composed of set of pods $P = \{p_1, p_2, \ldots\}$. 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 $R = \{r_1, r_2, \ldots\}$. 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:

- $\Omega^{\beta}_{p_i,p_j}$: Workload bandwidth requirements predicate matrix (binary). We will write $\Omega^{\beta}_{p_i,p_j} = 1$ if network bandwidth requirements from source pod p_i to target pod p_j must be assured.
- $\Omega^{\tau}_{p_i,p_j}$: Workload latency requirements predicate matrix (binary). We will denote $\Omega^{\tau}_{p_i,p_j} = 1$ 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 workload deployment matrix D_w (binary) decision variable, 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^w_p (binary) considers the deployment state of each pod from concrete workload. We will write $D^w_p = 1$ if the whole pod p from workload w is deployed, meaning that all of its replicas are deployed.

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

Inter-replica stream matrix $S^{w,p_i,r_k}_{p_j,r_m}(n_u, n_v)$ (binary) represents the network streams. We will write $S^{w,p_i,r_k}_{p_j,r_m}(n_u, n_v) = 1$ 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.

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. 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).

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:

$$\sum_{w\in W}\sum_{p\in P}\sum_{r\in R_p^{max}} D_{r,n}^{w,p} \times c_p \le C_n,$$

$$\sum_{w\in W}\sum_{p\in P}\sum_{r\in R_p^{max}} D_{r,n}^{w,p} \times e_p \le E_n,$$

$$\sum_{w\in W}\sum_{p\in P}\sum_{r\in R_p^{max}} D_{r,n}^{w,p} \times b_p \le B_n,$$

$$b_{p_i} = \sum_{\substack{j=1\\ j\neq i}}^{|P|} \beta_{p_i,p_j},$$

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.

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:

$$\sum_{w \in W} D_w = \text{result\_from\_(MAX\_WD)}.$$

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:

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

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.

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:

$$\sum_{w \in W} D_w = \text{result\_from\_(MIN\_RM)}.$$

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:

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

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:

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

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.

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 results presented in this chapter are published in [35].

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 [1]. Its microservice dependencies are shown in Fig. 1.1. The examples are using the Database ($P1$), Backend ($P2$), VehicleAgent ($P3$) and Frontend ($P4$) microservices, without the Analytics microservice. The total count of used microservice replicas (instances) is eleven. Each 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. 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 x1 is 800 kilometers. On each step all nodes are moved away from each other with a fixed distance Δx (Δx = x2−x1). 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). 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 5.1. 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.

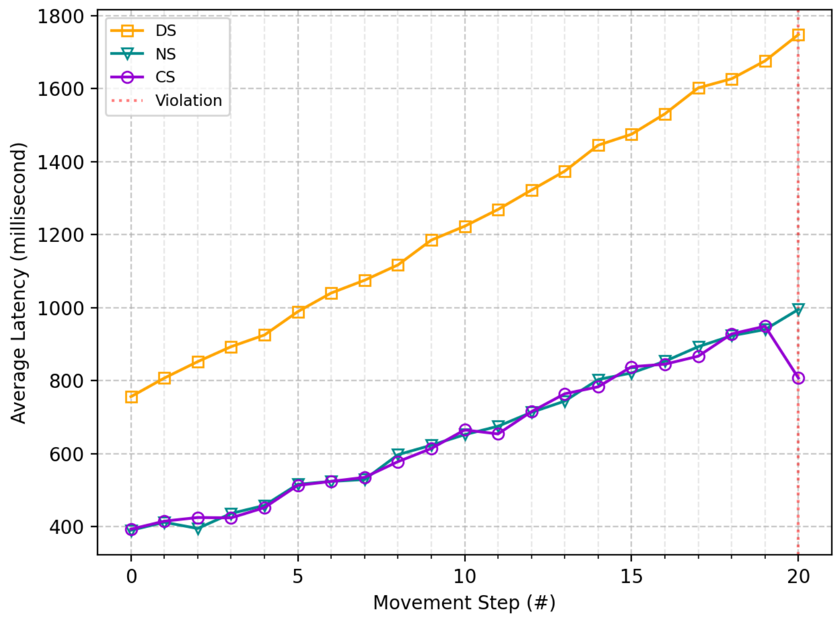

Figure 5.1: Example **E1** with small movement distance $\Delta x = 10\%(80km)$ per step. CS optimizes end-to-end latency by 20% in last step 20, compared to existing K8s schedulers.

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 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

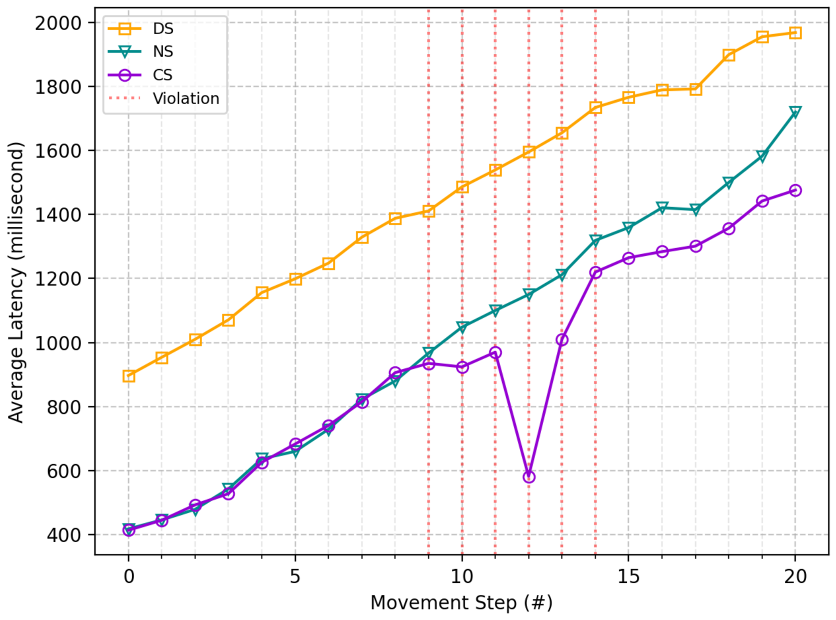

Figure 5.2: Example **E2** with big movement distance $\Delta x = 20\%(160km)$ per step. CS optimizes end-to-end latency by up to 48% starting early from step 9, compared to existing K8s schedulers.

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

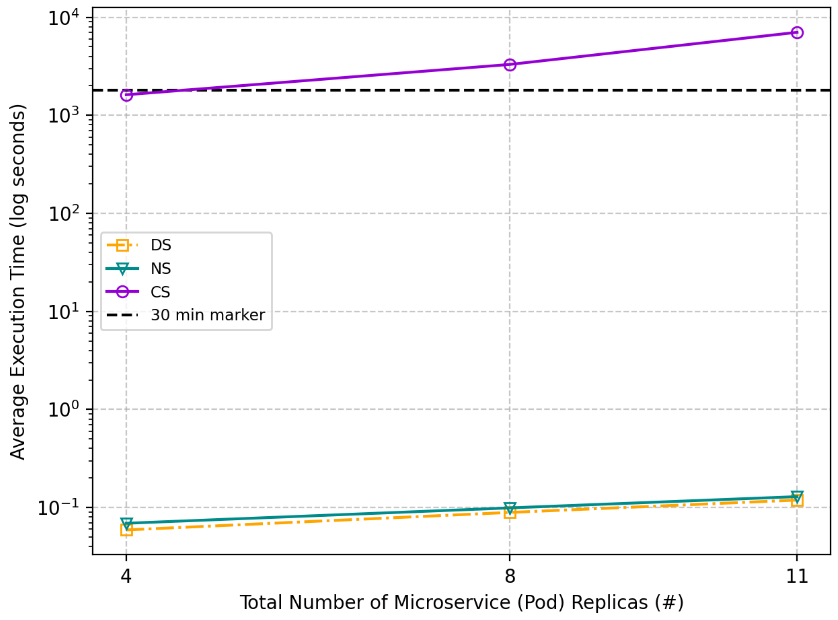

Figure 5.3: Execution time of CS with MILP is much bigger (27 to 117 min), compared to existing K8s schedulers (0.06 to 0.13 sec).

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 [35].

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 [1] 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.

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/Virtual Reality, real-time audio/video streaming, etc. The solution is implemented in the most popular Cloud/Edge/Fog platform in practice, called Kubernetes 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. [36] 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.
- 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 [1], 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.
- 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.

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.

Bibliography

[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: 10.1007/978-3-030-76352-7_50. [Online]. Available: https://doi.org/10.1007/978-3-030-76352-7_50.

[2] 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.

[3] 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.

[4] 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. [Online]. Available: https://doi.org/10.1007/978-981-10-5861-5_5.

[5] 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. [Online]. Available: https://doi.org/10.1007/s13369-021-05553-3.

[6] 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. [Online]. Available: https://doi.org/10.1145/3007787.3001169.

[7] 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.

[8] 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. [Online]. Available: https://doi.org/10.1186/s13677-022-00367-6.

[9] “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).

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] S. Hassan, I. Ahmad, A. Rehman, S. Hussen, 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.

[16] 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.

[17] 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 [cs.DC].

[18] 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.

[19] 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 [cs.DC].

[20] 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).

[21] 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. [Online]. Available: https://doi.org/10.1145/3586181.

[22] 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.

[23] 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. [Online]. Available: https://doi.org/10.1145/3419111.3421285.

[24] 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. [Online]. Available: https://doi.org/10.1007/978-3-030-65310-1_37.

[25] 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.

[26] 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.

[27] 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.

[28] 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. [Online]. Available: https://doi.org/10.1007/978-3-030-88224-2_5.

[29] 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. 102 915, 2021, ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2020.102915. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S108480452030374X.

[30] 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.

[31] 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.

[32] 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.

[33] 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. [Online]. Available: https://doi.org/10.1145/3573125.

[34] 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.

[35] 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.

[36] 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.