



Research article

Dynamic network-aware container allocation in Cloud/Fog computing with mobile nodes

Tsvetan Tsokov*, Hristo Kostadinov

Institute of Mathematics and Informatics, Bulgarian Academy of Sciences, Acad. Georgi Bonchev Str., Block 8, Sofia, 1113, Bulgaria



ARTICLE INFO

Keywords:

Resource scheduling
Mixed-integer linear programming (MILP)
Cloud computing
Fog computing
IoT
Mobility

ABSTRACT

The massive adoption of movable devices, created a new use cases such as Internet of Things (IoT), autonomous vehicles, spacecraft computing, etc. Not only end users, but also computing infrastructures can change their location. The applications are composed of many interdependent microservices with specific resource requirements. Latest state of the art in Cloud/Fog infrastructures is considering only the computational 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 Quality of Service (QoS) and network costs. Many researchers are focused only on a theoretical level. This paper 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 MILP optimization model and implementation in a Cloud/Fog platform. Several examples with sample Edge-Native application are obtained. The results show reduction in the total end-to-end network latency compared to the latest state of the art.

1. Introduction

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 [1] 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 Quality of Service (QoS) and Quality of Experience (QoE) [2–5]. Such kinds of applications are for example virtual and augmented reality, autonomous driving vehicles and traffic emissions control in smart cities [6]. 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 [7–10].

* Corresponding author.

E-mail address: tsvetan.tsokov@math.bas.bg (T. Tsokov).

URL: <https://blog.tsokov.eu/> (T. Tsokov).

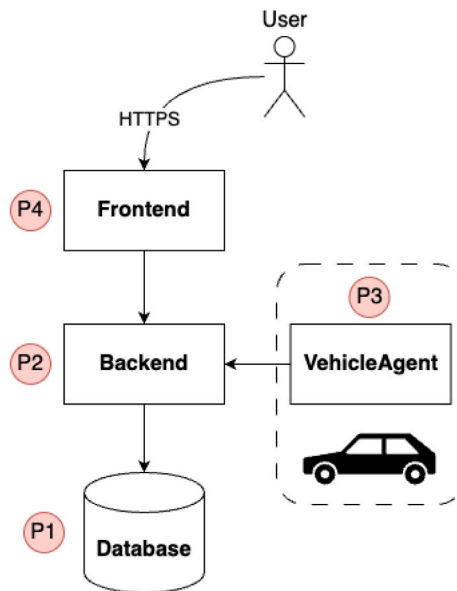


Fig. 1. EcoLogic application [6] microservice architecture.

Major Cloud providers are building their Edge computing platforms, including Multi-access Edge Computing (MEC) [11,12], like Azure Edge Zone, Google Distributed Cloud, AWS Local Zones and Alibaba Link IoT Edge, but they are at early stage due to the difficulties related to less computational resources and geo-distribution of Edge nodes [13]. On the other hand major content providers like Google and Facebook 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, Akamai and Fastly to begin to provide not only caching, but also computational processing on Edge by leveraging their global CDN infrastructure [14]. Macrometa [15] 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 [16]. 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 [17]. It is known that Edge network layer can be composed of nodes with low-power and ARM CPU architecture [18]. From the stated above we conclude that constraint 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 [19,20] principles and Twelve-Factor App methodology. Right now a new novel principles are emerging in practice called Edge-Native [21,22]. 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. For example EcoLogic [6] 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. 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. 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.

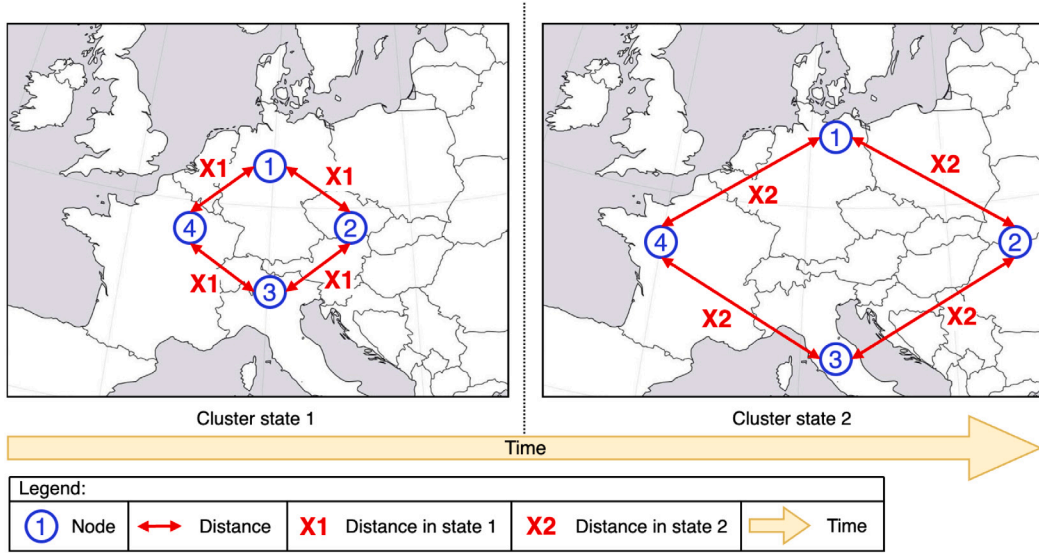


Fig. 2. Movement of nodes, part of geo-distributed (multi-region) edge cluster.

According to the above exposition we identify several major features of Edge/Fog computing platforms such as infrastructure topology awareness, virtualization type, application microservice dependencies support, dynamicity, mobility, network latency awareness, ARM64 CPU architecture support, evaluation type. We will use these features as a pillars for comparison with other related works and in system design and evaluation.

In our research we extend the MILP model proposed by Santos et al. [23] with several contributions:

- Introduction of 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.
- The model execution flow is modified in order to support dynamic optimization of replica placements with moveable nodes. The flow consists of phases and iterations (Fig. 3).

The remainder of the paper is structured as follows. In Section 2 the available scientific related works will be described. The developed Mixed-Integer Linear Programming (MILP) model will be shown in Section 3. In Section 4 the MILP model will be validated with examples based on the EcoLogic sample application [6]. Conclusion remarks and open problems will be given in Section 5.

2. Related work

This section reviews some important results related to the identified in Section 1 major features of Edge/Fog platforms such as 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 [24] and Cloud computing domains [25]. The resource provisioning techniques evolved from localness to remoteness. According to a recent surveys [26–29] we are seeing reverse movement with the research of novel resource allocation solutions in Edge/Fog domain, which favors both localness and remoteness.

The available resource allocation procedures in the literature are classified 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 [26]. Usually ILP models are considered 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 optimality benchmark for comparison with other faster heuristics-based techniques.

In Fog platforms the isolation of resources is crucial, because the devices are constraint. Resource allocations are classified by their virtualization model including Virtual Machines, used by many researchers and containerization [26]. Some investigations

Table 1
Related works.

Feature support		References
Infrastructure topology		[23,31,35–41,44] ^a
Microservice dependencies		[23,31,34–38,40,41,50] ^a
Dynamicity		[31–33,35–38,40,47–52,54] ^a
Mobility		[32,33,47,48,51,52,54] ^a
Network latency		[23,30–33,36–41,47,49–51,54] ^a
ARM64 CPU		[23,32,33,54] ^a
Scheduling objective	R	[34,35,48]
	L	[47,49,50] ^a
	B	[52]
	L&B	[23,36,38–41]
	R&L&B	[37]
	R&LU	[30–33,51,54]
	B&C	[44]
Evaluation type	S	[34,37,38,41,44,47–50,52]
	T	[30,31,39,51] ^a
	S&T	[23,32,33,35,36,40,54]

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

Evaluation Type: S = Simulation, T = Testbed.

^a The following feature is supported by the current work.

accentuate on container placement optimizations by considering different metrics, including multi-tenant resource fair scheduling and waiting time [30] and reducing end-to-end (E2E) tail latency [31–33].

The orchestrated entities are classified by task and microservice in Costa et al. survey [27]. The microservice dependencies are addressed mainly in the area of batch-job scheduling [34–36]. They target on resource utilization [35], categorization of inter-job dependencies [34] and modeling machine learning job flow by considering training stages [36].

Microservice dependencies and infrastructure topology are considered with objectives to optimize network overhead [37], maximize performance of CPU cores [38], minimize network tail latency [39], execute Deep Learning jobs on specific infrastructure resources of GPUs, TPUs, FPGAs and ASICs [40] or to minimize network latency and bandwidth of Big Data jobs and applications [41]. Most investigations using data center clusters emphasize mainly on network bandwidth instead of latency and use simulation-based evaluations.

A Mixed-Integer Linear Programming (MILP) resource provisioning in Fog–Cloud environments with containerization is examined in [42,43]. The formulation considers several LPWAN technologies and objectives, including network latency and bandwidth. It is further practically evaluated in Kubernetes (K8s) cluster with stationary, non-constraint (not ARM based) virtual machine nodes [23]. A network-aware MILP model and system for reduction of overall network expenses in Edge computing is studied in [44].

The dynamicity and mobility support in Fog is tackled by Salaht et al. and Ostrowski surveys [45,46]. They classify the dynamicity of the environment in two aspects: dynamicity of Fog infrastructure and dynamicity of applications. The Fog infrastructure is dynamic, because node's resources can vary over time and they can join and leave the network due to instability of links or malfunctions. 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. The surveys categorize the mobility related to Fog nodes and to 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. The mobility support is important challenge that has to be addressed due to the limited number of current works investigating it.

End-user network latency minimization in MEC by cross-edge service migration schemes is presented in [47]. Optimal service placement sequence for minimization of the average cost in mobile micro clouds is studied by Wang et al. [48]. Dynamic replica placement algorithms for data services using mean latency are presented in [49–51]. Placement algorithm for replicated VMs in Fog Radio Access Network (F-RAN), using end-user-to-edge proximity instead of edge-to-edge proximity, is proposed by Yu et al. [52].

Dynamic replica placement, replica autoscaling, traffic load balancing and Edge device provisioning with objectives to maintain low application end-to-end tail latency (QoS) and load distribution are demonstrated in [32,33,53,54]. Heuristic-based algorithm is used, elasticity is not supported and nodes are blocked until task is executed.

Table 1 shows comparison of the papers discussed above with our proposed model concerning the scheduling objective, infrastructure topology awareness, microservice dependencies, dynamicity, mobility, network latency, ARM64 support and evaluation type.

Later we will see that our investigation 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 constraint devices with ARM64 CPU architecture.

In the next Section 3 a MILP model with the above listed major features will be formulated.

Table 2

Mixed-Integer Linear Programming (MILP) input variables for infrastructure nodes and workloads.

Symbol	Description
b_p	Bandwidth requirement of pod p in Megabits/second (Mb/s), where $b_p \in \mathbb{N}$.
B_n	Total bandwidth capacity of node n in Megabits/second (Mb/s), 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_p	CPU requirement of pod p in CPU units, where $c_p \in \mathbb{Q}^+$, $c_p > 0$.
C_n	Total CPU cores capacity of node n in CPU units, where $C_n \in \mathbb{Q}^+$, $C_n > 0$.
e_p	Memory requirement of pod p in Megabytes (Mbyte), where $e_p \in \mathbb{N}$.
E_n	Total memory capacity of node n in Megabytes (Mbyte), where $E_n \in \mathbb{N}$.
$M_{n,u,z}$	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 .
M_p^w	Membership matrix (binary) $M_p^w \in \{0, 1\}$. $M_p^w = 1$ if pod p is member of workload w .
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 replicas $R = \{r_1, r_2, \dots\}$
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.
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.

3. 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 [55]. 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.

We will propose 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. 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 subsections.

3.1. 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 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 overall MILP model orchestration and its execution are illustrated in Fig. 3.

The model formulates the set of nodes $N = \{n_1, n_2, \dots\}$ in a Kubernetes-based Cloud/Fog cluster. Each node $n \in N$ has resources C_n , E_n and B_n . The network latency and bandwidth parameters between each two nodes are defined with metrics B_{n_u, n_v} and T_{n_u, n_v} . Nodes are separated in infrastructure regions $U = \{u_1, u_2, \dots\}$ and zones $Z = \{z_1, z_2, \dots\}$ forming a specific grouping.

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 pod has one or more instances, called replicas. Its number depends on variables R_p^{req} and

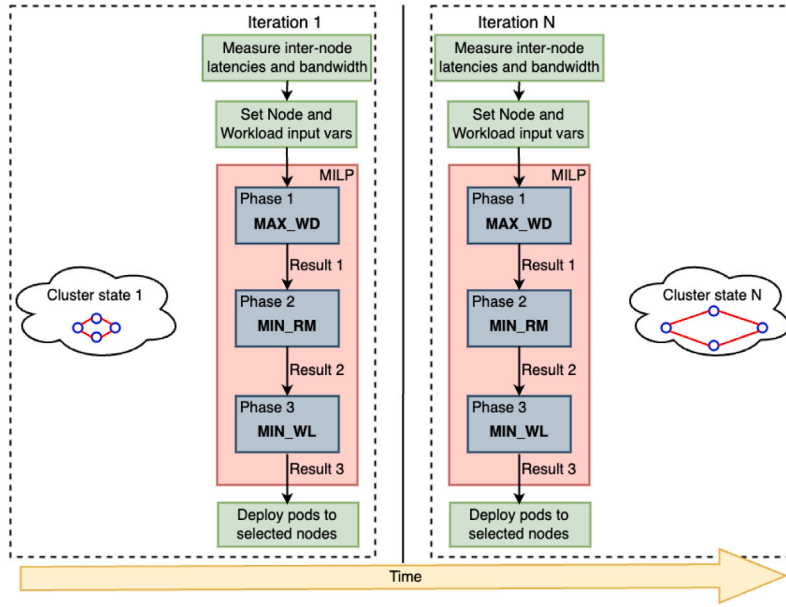


Fig. 3. MILP model orchestration and execution.

R_p^{max} . Each pod p has resource requirements c_p , e_p and b_p necessary for its proper functioning. As an example a CPU requirement c_p that is equal to 0.1 cpu (i.e. 100 millicpu) means that the pod needs 10% of a CPU core to function properly. The requirements predicate matrices Ω_{p_i, p_j}^β and Ω_{p_i, p_j}^τ define whether the corresponding inter-pod latency τ_{p_i, p_j} and bandwidth β_{p_i, p_j} requirements are guaranteed or not. Furthermore replica movements across nodes are limited by δ , affecting the dynamic allocation process. The input variables of the model related to the infrastructure nodes and workloads are described in Table 2. They have to be set in each iteration, before the execution of the objective functions, as shown in Fig. 3. 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 Algorithms 1 and 2 respectively.

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)

```

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. 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. 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 . The decision variables are shown in Table 3.

Algorithm 2: MILP model execution

```

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

```

Table 3

Mixed-Integer Linear Programming (MILP) decision variables.

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 .
$S_{p_i, r_m}^{w, p_i, r_k}(n_u, n_v)$	Inter-replica stream matrix (binary), representing replica and node dependencies in a network stream. $S_{p_i, r_m}^{w, p_i, r_k}(n_u, n_v) = 1$ 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.

3.2. 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. 3 and described in Algorithms 1 and 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.

3.2.1. 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 2) */
Output: out
/* MILP decision variables (Table 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

```

3.2.2. 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.

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

```

Input: in, out
/* MILP input variables (Table 2), decision variables (Table 3) */
Output: out
/* MILP decision variables (Table 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 ← minimizeReplicaMigrations(in, out)
6   return out

```

3.2.3. 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:

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

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 MIN_WL is described further in Algorithm 5.

In the next Section 4 the described above MILP model will be applied with examples and will be validated in practice.

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

```

Input: in, out
/* MILP input variables (Table 2), decision variables (Table 3) */
Output: out
/* MILP decision variables (Table 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
   interReplicaStreamMatrixLatencyConstraint(in) and
4   streamBandwidthFactorConstraint(in) and
   streamLatencyFactorConstraint(in)
5   do
6     out  $\leftarrow$  minimizeWorkloadTotalNetworkLatency(in, out)
7   return out

```

Table 4

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

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. ^a
N	Cluster is composed of 11 nodes.
P	EcoLogic Pods: Database (P1), Backend (P2), VehicleAgent (P3), Frontend (P4). ^a
R_p^{req}	1 to 6 ^b
R_p^{max}	1 to 6 ($R_p^{max} = R_p^{req}$ for $\forall p \in P$). ^b
T_{n_u, n_v}	10 to 60 ms.
U	11 regions.
W	There is a single workload w , representing the EcoLogic sample application. ^a
Z	11 zones.
β_{p_i, p_j}	100 Mbps for $\forall p_i, p_j \in P$.
δ	6 ^c
τ_{p_i, p_j}	30 ms 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 $\forall p_i, p_j \in P$.

^a EcoLogic sample application Pod microservices are described in Table 5 and Fig. 1.^b Number of Pod replicas used in execution time results and examples are described in Table 5.^c Replica movements across nodes (described in Section 3.2.2) are not limited, because $\delta = R_p^{max}$.**Table 5**

EcoLogic application Pods with Replica counts.

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

4. Results and discussion

In this section we will show two examples and an execution time result to demonstrate how the described model handles the identified in Section 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 [6]. Its microservice dependencies are shown in Fig. 1. The total count of used microservice replicas (instances) is eleven. Each microservice replica is deployed on different node.

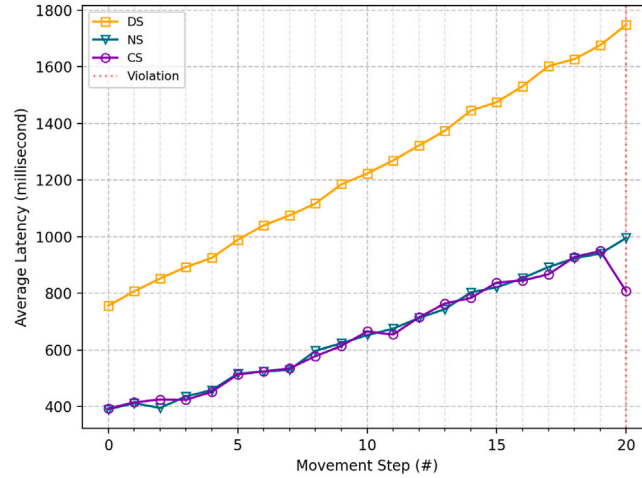


Fig. 4. Example E1 with small movement distance $\Delta x = 10\%$ (80 km) per step. CS optimizes end-to-end latency by 20% in last step 20, compared to existing K8s schedulers.

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. 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. 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 4. 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. 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 ms. The average initial distance between nodes x_1 is 800 km. 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 km). 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 ms 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. 3. 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 Quality of Service (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 Tables 4 and 5. 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 km).

The results of the examples E1 and E2 are depicted in Figs. 4 and 5, 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.

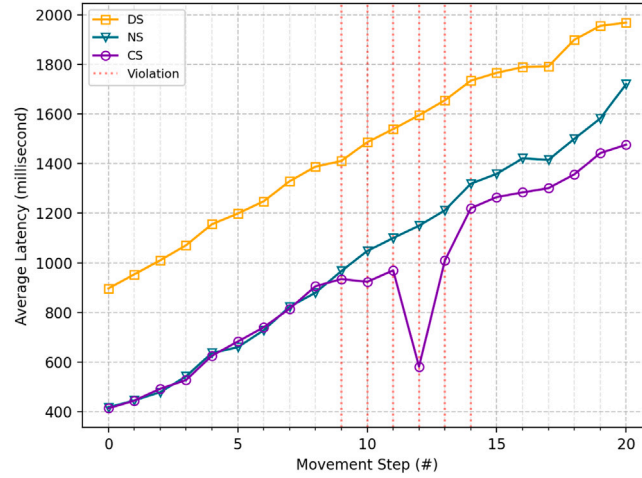


Fig. 5. Example E2 with big movement distance $\Delta x = 20\%$ (160 km) per step. CS optimizes end-to-end latency by up to 48% starting early from step 9, compared to existing K8s schedulers.

In example E1 results (Fig. 4), 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) 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 500 ms (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. 6. 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 min marker depicted in horizontal dashed line. The replicas count of the specific EcoLogic pods that are used are shown in Table 5. 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 s and NS - 0.07 to 0.13 s. In contrast to them, the CS have much bigger execution time of 27 to 117 min. 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 Figs. 4, 5. Usually many real-time and critical solutions in Cloud/Fog platforms should comply to a maximal execution time of 30 min.

4.1. Discussion

The obtained results in Figs. 4 and 5 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. 6 show that the MILP model takes much time to provide the most optimal solution, which is more than the 30 min 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.

5. 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 article presents a MILP formulation for container provisioning in

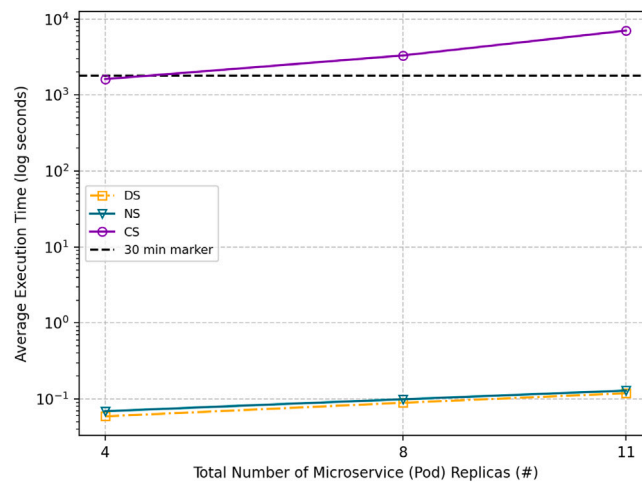


Fig. 6. Execution time of CS with MILP is much bigger (27 to 117 min), compared to existing K8s schedulers (0.06 to 0.13 s).

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 [6] 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/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.

For a future research the following directions are identified:

- 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 ($\tau_{p_i-p_j}$) 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.

CRedit authorship contribution statement

Tsvetan Tsokov: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Resources, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Hristo Kostadinov:** Writing – review & editing, Supervision, Resources, Project administration, Formal analysis, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

Acknowledgments

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). The authors would like to thank the anonymous reviewers for their valuable comments and suggestions that helped us to improve the paper.

References

- [1] G. Beniwal, A. Singhrova, A systematic literature review on iot gateways, *J. King Saud Univ. Comput. Inf. Sci.* 34 (2022) 9541–9563, <http://dx.doi.org/10.1016/j.jksuci.2021.11.007>, URL: <https://www.sciencedirect.com/science/article/pii/S1319157821003219>.
- [2] F. Bonomi, R. Milito, J. Zhu, 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, MCC '12, Association for Computing Machinery, New York, NY, USA, 2012*, pp. 13–16, <http://dx.doi.org/10.1145/2342509.2342513>.
- [3] M. Agiwal, A. Roy, N. Saxena, Next generation 5 g wireless networks: A comprehensive survey, *IEEE Commun. Surv. Tutor.* 18 (2016) 1617–1655, <http://dx.doi.org/10.1109/COMST.2016.2532458>.
- [4] A. Ahmed, H. Arkian, D. Battulga, A.J. Fahs, M. Farhadi, D. Giouroukis, A. Gougeon, F.O. Gutierrez, G. Pierre, P.R.S.J. au2, M.A. Tamiru, L. Wu, Fog computing applications: Taxonomy and requirements, 2019, <http://dx.doi.org/10.48550/arXiv.1907.11621>, arXiv:1907.11621.
- [5] D. Chatzopoulos, C. Bermejo, Z. Huang, P. Hui, Mobile augmented reality survey: From where we are to where we go, *IEEE Access* 5 (2017) 6917–6950, <http://dx.doi.org/10.1109/ACCESS.2017.2698164>.
- [6] T. Tsokov, H. Kostadinov, System for monitoring and control of vehicle's carbon emissions using embedded hardwares and cloud applications, in: H. Hacid, F. Outay, H.-y. Paik, A. Alloum, M. Petrocchi, M.R. Bouadjenek, A. Beheshti, X. Liu, A. Maaradji (Eds.), *Service-Oriented Computing – ICSC 2020 Workshops*, Springer International Publishing, Cham, 2021, pp. 564–577, http://dx.doi.org/10.1007/978-3-030-76352-7_50.
- [7] A. Laghari, A. Jumani, R. Laghari, Review and state of art of fog computing, *Arch. Comput. Methods Eng.* 28 (2021) <http://dx.doi.org/10.1007/s11831-020-09517-y>.
- [8] R. Das, M. Muhammad Inuwa, A review on fog computing: Issues, characteristics, challenges, and potential applications, *Telemat. Inform. Rep.* 10 (2023) 100049, <http://dx.doi.org/10.1016/j.teler.2023.100049>.
- [9] R. Mahmud, R. Kotagiri, R. Buyya, Fog Computing: A Taxonomy, Survey and Future Directions, Springer Singapore, Singapore, 2018, pp. 103–130, http://dx.doi.org/10.1007/978-981-10-5861-5_5.
- [10] S.A. Noghabi, L. Cox, S. Agarwal, G. Ananthanarayanan, The emerging landscape of edge computing, *GetMobile: Mob. Comput. Comm.* 23 (2020) 11–20, <http://dx.doi.org/10.1145/3400713.3400717>.
- [11] A. Ahmed, E. Ahmed, A survey on mobile edge computing, in: *2016 10th International Conference on Intelligent Systems and Control, ISCO, 2016*, pp. 1–8, <http://dx.doi.org/10.1109/ISCO.2016.7727082>.
- [12] H. Jin, M.A. Gregory, S. Li, A review of intelligent computation offloading in multiaccess edge computing, *IEEE Access* 10 (2022) 71481–71495, <http://dx.doi.org/10.1109/ACCESS.2022.3187701>.
- [13] M. Xu, Z. Fu, X. Ma, L. Zhang, Y. Li, F. Qian, S. Wang, K. Li, J. Yang, X. Liu, From cloud to edge: A first look at public edge platforms, 2021, <http://dx.doi.org/10.48550/arXiv.2109.03395>, arXiv:2109.03395.
- [14] Z. Jin, L. Pan, S. Liu, Randomized online edge service renting: Extending cloud-based cdn to edge environments, *Knowl.-Based Syst.* 257 (2022) 109957, <http://dx.doi.org/10.1016/j.knsys.2022.109957>, URL: <https://www.sciencedirect.com/science/article/pii/S0950705122010504>.
- [15] Akamai, Akamai edge workers in macrometa, 2024, URL: <https://www.macrometa.com/docs/akamai/>.
- [16] A. Bhardwaj, C.R. Krishna, Virtualization in cloud computing: Moving from hypervisor to containerization—a survey, *Arab. J. Sci. Eng.* 46 (2021) 8585–8601, <http://dx.doi.org/10.1007/s13369-021-05553-3>.
- [17] C. Dall, S.-W. Li, J.T. Lim, J. Nieh, G. Kolovrentzos, Arm virtualization: Performance and architectural implications, *SIGARCH Comput. Archit. News* 44 (2016) 304–316, <http://dx.doi.org/10.1145/3007787.3001169>.
- [18] K. Sivaprakasam, P. Sriramalakshmi, P. Singh, M. Bhaskar, Chapter 4 - an overview of low power hardware architecture for edge computing devices, in: A.K. Bhoi, V.H.C. de Albuquerque, S.N. Sur, P. Barsocchi (Eds.), *5G IoT and Edge Computing for Smart Healthcare, Intelligent Data-Centric Systems*, Academic Press, 2022, pp. 89–109, <http://dx.doi.org/10.1016/B978-0-323-90548-0.00004-8>, URL: <https://www.sciencedirect.com/science/article/pii/B9780323905480000048>.
- [19] N. Kratzke, P.-C. Quint, Understanding cloud-native applications after 10 years of cloud computing - a systematic mapping study, *J. Syst. Softw.* 126 (2017) 1–16, <http://dx.doi.org/10.1016/j.jss.2017.01.001>, URL: <https://www.sciencedirect.com/science/article/pii/S0164121217300018>.
- [20] J. Alonso, L. Orue-Echevarria, V. Casola, A.I. Torre, M. Huarte, E. Osaba, J.L. Lobo, Understanding the challenges and novel architectural models of multi-cloud native applications – a systematic literature review, *J. Cloud Comput.* 12 (2023) <http://dx.doi.org/10.1186/s13677-022-00367-6>.
- [21] C.N.C. Foundation, Edge-native application principles, 2024, URL: <https://www.cnfc.io/reports/edge-native-application-design-behaviors-whitepaper/>.
- [22] M. Jansen, A. Al-Dulaimy, A.V. Papadopoulos, A.K. Trivedi, A. Iosup, The spec-rg reference architecture for the compute continuum, in: *2023 IEEE/ACM 23rd International Symposium on Cluster, Cloud and Internet Computing, CCGrid, 2022*, pp. 469–484, URL: <https://api.semanticscholar.org/CorpusID:257280275>.
- [23] J. Santos, C. Wang, T. Wauters, F.D. Turck, Diktyo: Network-aware scheduling in container-based clouds, *IEEE Trans. Netw. Serv. Manag.* (2023) <http://dx.doi.org/10.1109/TNSM.2023.3271415>, 1–1.
- [24] A. Beloglazov, J. Abawajy, R. Buyya, Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing, *Future Gener. Comput. Syst.* 28 (2012) 755–768, <http://dx.doi.org/10.1016/j.future.2011.04.017>, URL: <https://www.sciencedirect.com/science/article/pii/S0167739X11000689>. special Section: Energy efficiency in large-scale distributed systems.
- [25] J. Zhang, H. Huang, X. Wang, Resource provision algorithms in cloud computing: A survey, *J. Netw. Comput. Appl.* 64 (2016) 23–42, <http://dx.doi.org/10.1016/j.jnca.2015.12.018>, URL: <https://www.sciencedirect.com/science/article/pii/S104804516000643>.
- [26] J.B. Jr., B. Costa, L.R. Carvalho, M.J.F. Rosa, A. Araujo, Computational resource allocation in fog computing: A comprehensive survey, *ACM Comput. Surv.* (2023) <http://dx.doi.org/10.1145/3586181>, just Accepted.
- [27] B. Costa, J. Bacheiga, L.R. de Carvalho, A.P.F. Araujo, Orchestration in fog computing: A comprehensive survey, *ACM Comput. Surv.* 55 (2022) <http://dx.doi.org/10.1145/3486221>.
- [28] K. Kaur, A. Singh, A. Sharma, A systematic review on resource provisioning in fog computing, *Trans. Emerg. Telecommun. Technol.* 34 (2023) e4731, <http://dx.doi.org/10.1002/ett.4731>, URL: <https://onlinelibrary.wiley.com/doi/abs/10.1002/ett.4731>.
- [29] A. Shakarami, H. Shakarami, M. Ghobaei-Arani, E. Nikougoftar, M. Faraji-Mehmandar, Resource provisioning in edge/fog computing: A comprehensive and systematic review, *J. Syst. Archit.* 122 (2022) 102362, <http://dx.doi.org/10.1016/j.sysarc.2021.102362>, URL: <https://www.sciencedirect.com/science/article/pii/S1383762121002526>.

- [30] A. Beltré, P. Saha, M. Govindaraju, Kubesphere: An approach to multi-tenant fair scheduling for kubernetes clusters, in: 2019 IEEE Cloud Summit, 2019, pp. 14–20, <http://dx.doi.org/10.1109/CloudSummit47114.2019.00009>.
- [31] D. Crankshaw, G.-E. Sela, X. Mo, C. Zumar, I. Stoica, J. Gonzalez, A. Tumanov, Inferline: Latency-aware provisioning and scaling for prediction serving pipelines, in: Proceedings of the 11th ACM Symposium on Cloud Computing, SoCC '20, Association for Computing Machinery, New York, NY, USA, 2020, pp. 477–491, <http://dx.doi.org/10.1145/3419111.3421285>.
- [32] A.J. Fahs, 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, Springer-Verlag, Berlin, Heidelberg, 2020, pp. 508–524, http://dx.doi.org/10.1007/978-3-030-65310-1_37.
- [33] A.J. Fahs, G. Pierre, 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, <http://dx.doi.org/10.1109/MASCOTS50786.2020.9285953>.
- [34] A. Chung, S. Krishnan, K. Karanasos, C. Curino, G.R. Ganger, Unearthing inter-job dependencies for better cluster scheduling, in: Proceedings of the 14th USENIX Conference on Operating Systems Design and Implementation, OSDI'20, USENIX Association, USA, 2020.
- [35] A. Qiao, S.K. Choe, S.J. Subramanya, W. Neiswanger, Q. Ho, H. Zhang, G.R. Ganger, E.P. Xing, Pollux: Co-adaptive cluster scheduling for goodput-optimized deep learning, 2021, <http://dx.doi.org/10.48550/arXiv.2008.12260>, arXiv:2008.12260.
- [36] Y. He, W. Cai, P. Zhou, G. Sun, S. Luo, H. Yu, M. Guizani, Beamer: Stage-aware coflow scheduling to accelerate hyper-parameter tuning in deep learning clusters, IEEE Trans. Netw. Serv. Manag. 19 (2022) 1083–1097, <http://dx.doi.org/10.1109/TNSM.2021.3132361>.
- [37] X. Li, J. Zhou, X. Wei, D. Li, Z. Qian, J. Wu, X. Qin, S. Lu, Topology-aware scheduling framework for microservice applications in cloud, IEEE Trans. Parallel Distrib. Syst. PP (2023) 1–17, <http://dx.doi.org/10.1109/TPDS.2023.3238751>.
- [38] V. Rao, V. Singh, K.S. Goutham, B.U. Kempaiah, R.J. Mampilli, S. Kalambur, D. Sitaram, 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, Springer-Verlag, Berlin, Heidelberg, 2021, pp. 80–100, http://dx.doi.org/10.1007/978-3-030-88224-2_5.
- [39] H. Zhu, K. Kaffes, Z. Chen, Z. Liu, C. Kozyrakis, I. Stoica, X. Jin, RackSched: A Microsecond-Scale scheduler for Rack-Scale computers, in: 14th USENIX Symposium on Operating Systems Design and Implementation, OSDI 20, USENIX Association, 2020, pp. 1225–1240, URL: <https://www.usenix.org/conference/osdi20/presentation/zhu>.
- [40] D. Narayanan, K. Santhanam, F. Kazhamiaka, A. Phanishayee, M. Zaharia, Heterogeneity-aware cluster scheduling policies for deep learning workloads, 2020, <http://dx.doi.org/10.48550/arXiv.2008.09213>, arXiv:2008.09213.
- [41] D. Haja, B. Vass, 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, <http://dx.doi.org/10.1109/CloudNet47604.2019.9064109>.
- [42] J. Santos, T. Wauters, B. Volckaert, F. De Turck, Towards end-to-end resource provisioning in fog computing over low power wide area networks, J. Netw. Comput. Appl. 175 (2021) 102915, <http://dx.doi.org/10.1016/j.jnca.2020.102915>, URL: <https://www.sciencedirect.com/science/article/pii/S108480452030374X>.
- [43] J. Santos, T. Wauters, B. Volckaert, F. De Turck, Resource provisioning for iot application services in smart cities, in: 2017 13th International Conference on Network and Service Management, CNSM, 2017, pp. 1–9, <http://dx.doi.org/10.23919/CNSM.2017.8255974>.
- [44] A. Santoyo-González, C. Cervelló-Pastor, Network-aware placement optimization for edge computing infrastructure under 5g, IEEE Access 8 (2020) 56015–56028, <http://dx.doi.org/10.1109/ACCESS.2020.2982241>.
- [45] F.A. Salaht, F. Desprez, A. Lebre, An overview of service placement problem in fog and edge computing, ACM Comput. Surv. 53 (2020) <http://dx.doi.org/10.1145/3391196>.
- [46] K. Ostrowski, K. Malecki, P. Dziurzański, A.K. Singh, Mobility-aware fog computing in dynamic networks with mobile nodes: A survey, J. Netw. Comput. Appl. 219 (2023) 103724, <http://dx.doi.org/10.1016/j.jnca.2023.103724>, URL: <https://www.sciencedirect.com/science/article/pii/S1084804523001431>.
- [47] T. Ouyang, Z. Zhou, X. Chen, Follow me at the edge: Mobility-aware dynamic service placement for mobile edge computing, IEEE J. Sel. Areas Commun. 36 (2018) 2333–2345, <http://dx.doi.org/10.1109/JSAC.2018.2869954>.
- [48] S. Wang, R. Ugaonkar, T. He, K. Chan, M. Zafer, K.K. Leung, Dynamic service placement for mobile micro-clouds with predicted future costs, IEEE Trans. Parallel Distrib. Syst. 28 (2017) 1002–1016, <http://dx.doi.org/10.1109/TPDS.2016.2604814>.
- [49] A. Aral, T. Ovatman, A decentralized replica placement algorithm for edge computing, IEEE Trans. Netw. Serv. Manag. 15 (2018) 516–529, <http://dx.doi.org/10.1109/TNSM.2017.2788945>.
- [50] Y. Shao, C. Li, H. Tang, A data replica placement strategy for iot workflows in collaborative edge and cloud environments, Comput. Netw. 148 (2019) 46–59, <http://dx.doi.org/10.1016/j.comnet.2018.10.017>, URL: <https://www.sciencedirect.com/science/article/pii/S1389128618311460>.
- [51] C. Li, Y. Wang, H. Tang, Y. Zhang, Y. Xin, Y. Luo, Flexible replica placement for enhancing the availability in edge computing environment, Comput. Commun. 146 (2019) 1–14, <http://dx.doi.org/10.1016/j.comcom.2019.07.013>, URL: <https://www.sciencedirect.com/science/article/pii/S0140366418308296>.
- [52] Y.-J. Yu, T.-C. Chiu, A.-C. Pang, M.-F. Chen, 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, <http://dx.doi.org/10.1109/ICC.2017.7996500>.
- [53] K. Toczé, S. Nadjim-Tehrani, Orch: Distributed orchestration framework using mobile edge devices, in: 2019 IEEE 3rd International Conference on Fog and Edge Computing, IC FEC, 2019, pp. 1–10, <http://dx.doi.org/10.1109/CFEC.2019.8733152>.
- [54] K. Toczé, A.J. Fahs, G. Pierre, S. Nadjim-Tehrani, Violinn: Proximity-aware edge placement with dynamic and elastic resource provisioning, ACM Trans. Internet Things 4 (2023) <http://dx.doi.org/10.1145/3573125>.
- [55] A. Haider, R. Potter, A. Nakao, Challenges in resource allocation in network virtualization, in: 20th ITC Specialist Seminar, Vol. 18, ITC, 2009, URL: <https://api.semanticscholar.org/CorpusID:10681804>.