Zeydan, E., Mangues-Bafalluy, J., Baranda, J., Martínez, R. & Vettori, L. (2022). A Multi-criteria Decision Making Approach for Scaling and Placement of Virtual Network Functions. *Journal of Network and Systems Management, 30*(2), 32.

DOI: 10.1007/s10922-022-09645-9

**Noname manuscript No.**
(will be inserted by the editor)

# A Multi-Criteria Decision Making Approach for Scaling and Placement of Virtual Network Functions

**Engin Zeydan*** · **Josep Mangues-Bafalluy** · **Jorge Baranda** · **Ricardo Martínez** · **Luca Vettori**

Received: date / Accepted: date

**Abstract** This paper investigates the joint scaling and placement problem of network services made up of Virtual Network Functions (VNFs) that can be provided inside a cluster managing multiple Points of Presence (PoPs). Aiming at increasing the VNF service satisfaction rates and minimizing the deployment cost, we use both transport and cloud-aware VNF scaling as well as multi-attribute decision making (MADM) algorithms for VNF placement inside the cluster. The original joint scaling and placement problem is known to be NP-hard and hence the problem is solved by separating scaling and placement problems and solving them individually. The experiments are done using a dataset containing the information of a deployed digital-twin network service. These experiments show that considering transport and cloud parameters during scaling and placement algorithms perform more efficiently than the only cloud based or transport based scaling followed by placement algorithms. One of the MADM algorithms, Total Order Preference By Similarity to the Ideal Solution (TOPSIS), has shown to yield the lowest deployment cost and highest VNF request satisfaction rates compared to only transport or cloud scaling and other investigated

* Corresponding author.

Engin Zeydan, Josep Mangues-Bafalluy, Jorge Baranda, Ricardo Martínez, Luca Vettori
Centre Tecnològic de Telecomunicacions de Catalunya (CTTC),
Castelldefels, Barcelona, Spain, 08860.
E-mail: {engin.zeydan, josep.mangues, jbaranda, rmartinez, lvettori}@cttc.cat

MADM algorithms. Our simulation results indicate that considering both transport and cloud parameters in various availability scenarios of cloud and transport resources has significant potential to provide increased request satisfaction rates when VNF scaling and placement using the TOPSIS scheme is performed.

**Keywords** scaling · orchestration · MADM · VNF · digital-twin · cloud

## 1 Introduction

Business needs are changing dynamically and becoming more demanding for today's network services. People are using applications provided by virtualized network services and infrastructure over a wide variety of use cases ranging from low latency to high throughput applications. Emerging mobile applications such as virtual reality or autonomous driving demand diversified quality-of-service (QoS). For instance, a virtual reality application can require Gbps data rate whereas an Internet of Things (IoT) device demand can be on the order of kbps data rates. Therefore, optimizing the QoS demands is essential when deploying network services. At the same time given the scale of applications, it is difficult to predict and manage the traffic to the applications and services provided by service or network infrastructure providers. The traffic or load pattern can be in different demand patterns such as bursty, on/off, seasonal or fast growth. Hence, the deployed service and critical infrastructure should be flexible and resilient enough in the face of varying load patterns and against failures which occur frequently in distributed systems. This can handle high volumes of traffic with high capacity utilization and the application instances should be able to match the required demands using the proper amount of resources. For this reason, a scalable solution for an application or service can be beneficial as it can enable the satisfaction of high loads on machines during peak hours and daytime and would reduce underutilized servers during off-peak hours during the night time.

The automation of network management can be easily enabled together with the introduction of Network Function Virtualization (NFV)/Software-Defined Networking (SDN) paradigms inside the underlying network infrastructure. At the same time, containers are making the packaging and orchestration of applications within

its infrastructure (via Kubernetes, LXC or Mesos) easier. For example, automated container deployment, scaling, and management platforms, like Kubernetes can manage up to 5000 servers in a cluster and 150,000 pods (containers)[1]. For this reason, most of the existing solutions or platforms on Management and Network Orchestration (MANO) rely on the network flexibility provided by these technologies. On the other hand, MANO platforms should also rely on intelligent network management and orchestration algorithms to better automate network operations. For example for scaling, network services requiring scaling need to be auto-scaled intelligently so that the Service Level Agreements (SLAs) between the service providers and users can be satisfied in dynamic environments. In case auto-scaling is not an available option for network services, a lot of time and manual effort need to be given to predict the exact traffic demand, provision servers, manage the infrastructure, monitor and control the deployment cost. Even so, there can be corner cases where network services can be either under-provisioned or over-provisioned. In under-provisioning case, the necessary computing and network resources can reject some of the incoming traffic loads and therefore can detriment the overall network service performance. In resource over-provisioning case, idle resources can be available resulting in higher cost for service providers. However, together with the introduction of auto-scaling feature for network services, several advantages can be obtained. Some of them are related to managing sudden bursts in traffic (e.g. during festivals, Christmas) gracefully, enhancing fault tolerance, availability and cost efficiency of the deployment, reducing human error and finally instant deployment of new services.

For all these reasons, as the infrastructure grows in volume, complexity, traffic, etc., MANO systems require auto-scaling capabilities to avoid or minimize SLA violation by the offered services and in the worst-case scenario avoid their downtime. At the same time, the scaling must be done in every dimension, geographically, in number of nodes, in requests per second, in data size, in number of tenants, etc. For example, auto-scaling mechanisms can be proactively triggered to mitigate the effects of high demand events to the offered services provided by the infrastructure as

[1] https://kubernetes.io/blog/2017/03/scalability-updates-in-kubernetes-1-6/, Accessed: May-2021

a whole. In traditional auto-scaling operations, first the relevant information needs to be collected (central processing unit (CPU) or memory utilization, load average, queue length, etc) and compared with respect to target utilization values. Later, whether a need for scaling is determined based on algorithm design. Note that in some cases, target values may result from a prediction of what a service is expecting in the future. There are various papers that discuss how to predict the future attributes of services on different scales, e.g. via Machine Learning (ML) in [1] to predict future NFV requests or forecasting the future evolution of metrics in an NFV infrastructure as done in [2].

### 1.1 Related Work

In the literature, there are contributions considering the scaling on transport or cloud domains and container/Virtual Machines (VMs)[2] placement problems as two distinct topics or combining them together.

**In scaling,** many of the approaches concentrate on cloud environments and are still an active area of research. In open source community, the Kubernetes container orchestration system[3] offers mainly three types of scaling:

1. *Cluster autoscaling* — Adjust the number of worker nodes in the cluster automatically via the Cluster Autoscaler to optimize your nodes' resources as given in Fig. 1[4]. Cluster autoscaling mechanism ensures to scale over cluster which can be geographically distributed, i.e. in different Point of Presences (PoPs). Pods (smallest deployable units in Kubernetes) are placed according to the available CPU and memory metrics of each node so that new instances can run without issues. Therefore, Kubernetes can add more servers to the current cluster in case cluster autoscaling needs more servers depending on the availability of pending pods and node utilization metric. More specifically, cluster autoscaling adjusts

[2] In the rest of paper, we interchangeably use both containers and VMs when referring to scaling.

[3] https://kubernetes.io/, Accessed: May-2021

[4] https://thenewstack.io/kubernetes-deployments-work/, Accessed: May-2021

the Kubernetes cluster size (i.e. the number of nodes) in case (i) there is pod failures in the cluster due to insufficient resources, (ii) some pods in the cluster are underutilized for an extended period of time and their pods can be placed on other existing nodes.

2. *Horizontal pod autoscaling* — Adjust the number of pods in your deployment via the Horizontal Pod Autoscaler based on the pods' CPU and memory utilization. In case the current physical host cannot accommodate the updated CPU and memory requirements, pods will be evicted and a new one will be scheduled. In this mechanism, the number of replicas deployed increases or decreases depending on specific observed and target metrics as also outlined in the rest of the paper. Horizontal scaling can occur either in cluster or in individual microservice level. In this paper, we assume it occurs in individual microservice level. Additionally, in cases more replicas are going to be deployed, load balancers are required to balance the traffic among the replicas so that the observed metrics (e.g. CPU) can be reduced into desired or targeted values. Horizontal scaling also refers to scale in/out operation.
3. *Vertical pod autoscaling* — Adjust the CPU and memory of your pods to meet the application's real usage. Vertical scaling also refers to scale up/down operation.

Horizontal scaling is usually more desirable than vertical scaling due to higher flexibility. However, building a software to scale horizontally is more complex than vertical scaling. On the other hand, vertical scaling cost increases exponentially after a certain threshold [3]. Even horizontal scaling can have some limits in CPU and memory. In that case, it would be preferable either to upgrade the processing power through vertical scaling or switch to cluster scaling. Microservices based applications can be scaled in multiple ways since they provide more granular controls over the application performance than monolithic scaling which are less likely to scale horizontally, i.e. inside the same VM/container. The authors in [4] propose an Integer Linear Programming (ILP) approach and a greedy heuristic to address this NP-Hard Virtual Network Function (VNF) scaling problem. They considered both horizontal (scale out/in) and vertical (scale up/down) scaling and show that together with suc-

cessful scalings and VNF migrations, more users into the system can be accepted. The paper in [5] investigates the problem of selecting the most appropriate performance metrics to enable more accurate scaling decisions. An enhanced version of the Kubernetes horizontal pod scaling auto-scaling algorithm is proposed to control the application response time and to keep it according to service level objectives. The authors in [6] investigate solutions to control the horizontal and vertical scalability of container-based applications using reinforcement learning RL algorithms. The results indicate the benefits and flexibility of RL based solutions, especially the model-based approach, where the best adaptation policy based on user-defined deployment targets can successfully be learnt.

However, the auto-scaling solutions to scaling problems are either based on rule or threshold based approaches (e.g. Ceilometer, the monitoring component of OpenStack[5]) considering only simple compute metrics (i.e. CPU, Random Access Memory (RAM) or storage utilization) while mostly ignoring network measurements or do not consider placement decisions after scaling. At the same time, Kubernetes is also working on extending its solutions to accommodate other customized metrics via Application Programming Interface (API)[6]. Due to automatic increase or decrease of the number of computing resources assigned to the application based on the service demands at any given time, auto-scaling has been an essential feature provided by many industrial cloud companies such as Amazon, Google, etc. Serverless and event-driven cloud platforms such as Cloud Functions[7] and Cloud Run[8] provided by Google Inc. enable the creation of highly scalable applications. The main advantage of using serverless cloud platforms is that infrastructure details are left to cloud providers and application developers only focus on their own application code and let them auto-scale after deployment. On the other hand, a failure in auto-scaling can also be costly (e.g. loss of sales for retail) to cloud providers [7]. The authors in [8]

[5] OpenStack's Ceilometer https://docs.openstack.org/ceilometer/latest/, Accessed: May-2021

[6] https://github.com/kubernetes/community/blob/master/contributors/design-proposals/instrumentation/custom-metrics-api.md, Accessed: May-2021

[7] https://cloud.google.com/functions, Accessed: May-2021

[8] https://cloud.google.com/run, Accessed: May-2021

improves Kubernetes default auto-scaling paradigm by considering different metrics such as memory and number of undelivered messages in the queue. The paper in [9] experiments horizontal pod auto-scaling of Kubernetes through diverse aspects such as latency, request processing, cluster size, metric collection, etc. to obtain its operational behaviours. A hierarchical architecture for controlling the elasticity of microservice based applications with a Kubernetes extension is studied in [10]. The authors in [11] propose a Kubernetes scaling engine that makes the auto-scaling decisions using various machine learning forecast methods.

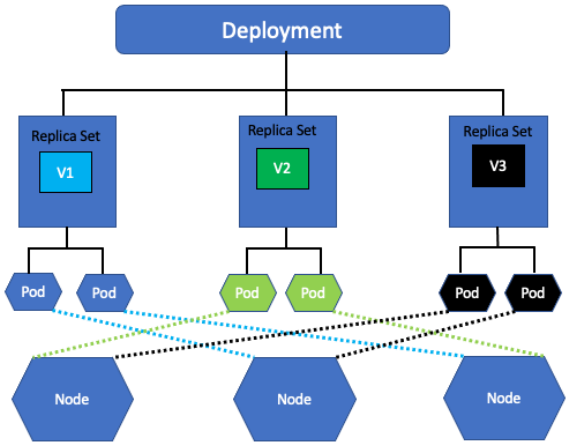



Fig. 1: Kubernetes deployment for cluster auto-scaling

**In placement,** many methods with different objectives have been proposed from the container/VNF placement point of view [12]. Some typical examples are the minimization of operational cost and service latency [13, 12], optimizing the host utilization and the communication cost while considering load balancing [14], jointly optimizing cloud data center energy usage and resource utilization in [15], balancing the access, switching and communication delay via access network selection and service placement in [16]. Hosting all VNFs on the same host can minimize the cost [13] while latency minimization can be achieved by replicating VNF instances through load balancing [12]. The authors in [17] have addressed network aware placement of virtual computing and data components in data centers. Many of the open-source

community platforms such as OpenStack or Kubernetes rely on container/service instance placement in a node within a cluster based on some simplistic metrics such as CPU or memory of each node. Kubernetes considers this as scheduling process that is managed by scheduler by which pods are matched to available resources. Transport-awareness, i.e. network-aware scheduler is shown to decrease the job completion times of Hadoop clusters in comparison to default scheduler of Hadoop in [18]. The authors in [19] have defined VNF placement problem as a potential game to maximize the number of allocated VNF requests with minimum overall deployment cost. Finally, a summary of VM placement methods that are classified as either dynamic or static, including their objectives, and their metrics is given in [20,21]. A VM placement algorithm considering their priorities as well as guaranteed bandwidth requirements to reduce network congestion is studied in [22]. The authors in [23] present a multi-objective VM placement scheme that utilizes an Artificial Bee Colony optimization algorithm for power and network-aware assignment of VMs onto physical machines to minimize the network traffic between interacting VMs as well as power dissipation of the data center switches and physical machines.

**In joint scaling and placement,** the authors in [24] are investigating the dynamic placement of VNFs after scaling decisions both at edge nodes and cloud data centers using ML classifier models and obtain low end-to-end latency as well as reduce latency violations. The authors in [25] address the VNF placement problem with the goal of minimizing service chain deployment as well as end-to-end latency minimization considering the location requirements. The authors in [26] have considered auto-scaling services while considering network QoS metrics as well. The authors in [27] worked on extending the OpenStack scheduler to consider a network-aware placement of VMs considering ingress and egress bandwidth constraints to nodes and utilize it via a prototype. However, the performance gains of the approach are based on "flavour" (the instance type) settings rather than actual measurements and the benefits are not quantified in large-scale numerical evaluations.

**Multiple attribute decision making:** Multiple Attribute Decision Making (MADM)/ Multiple Criteria Decision Making (MCDM) algorithms have been extensively used in the network scenarios to achieve the best trade-off between the alternative choices.

The authors in [28] provide an investigation of the MCDM-based service selection methods for web service selection, composition, cloud service selection, and Cloud Service Provider (CSP) selection. The authors in [29] have applied MADM algorithms that are also used in this paper for the selection of best network and Device-to-Device (D2D) communications to exploit the user proximity in crowded environments. The authors in [30] have done performance evaluation of various MADM-based methods for best network selection in terms of anytime and anywhere connection. A flexible hybrid MADM heterogeneous network selection algorithm consisting of fuzzy analytic hierarchy process (FAHP), standard deviation, and Grey relational analysis (GRA) is proposed in [31] to reduce the number of vertical handovers and ping-pong effects.

To summarize, traditional cloud auto-scaling solutions do not consider the network requirements of service when scaling and placing the service instances. However, it is important to consider both network and cloud parameters when deploying composite services and applications with many inter-connecting micro-services where placement of one service in certain locations may affect the others. In fact, acquiring and integrating different data sources enables better decision-making and service experience in many use cases. In comparison to above works in our scaling decision process, we differentiate transport scaling and cloud scaling and also consider the application of MADM algorithms for appropriate placement of the scaled VMs. In case cloud and transport resources are jointly optimized, better decisions have been made from higher perspectives during network service management. Reduced network service deployment cost and higher VNF satisfaction rates are some of the benefits that have been obtained in this paper in comparison to decision making over limited observations either in cloud or transport domain. To the best of our knowledge, this is the first transport and cloud parameter aware scaling and VNF placement method that can be rolled out agilely to improve the overall performance of the network infrastructure.

Table 1

Symbols used throughout the paper

| Symbol | Meaning |
|---|---|
| $K$, $\mathcal{K}$ | number of nodes , Node set |
| $S$, $\mathcal{S}$ | total number of requests, the set of requests in the available infrastructure |
| $S_k$, $\mathcal{S}_k$ | number of request into node-k, set of requests placed into node-k |
| $F_k^i$, $\mathcal{F}_k^i$ | number of NEs used to reach to node-k for a given request-i , the set of NEs used to reach to node-k for a given request-i |
| $\mathcal{F}$ | the set of all NEs |
| $B_{k,j}^{i,f}$, $\mathcal{B}_{k,j}^{i,f}$ | number of utilized resources at NE-$f$ after VNF allocation, the set of resources utilized after VNF allocation |
| R, $\mathcal{R}$ | number of different types of resources, the type of the resource |
| $b_j$ | j-th resource type from $\mathcal{B}_{k,j}^{i,f}$. |
| $\mathbf{C}$, $C_j$ | An R-dimensional vector capacities for R resources , the capacity of $b_j$ |
| $\mathbf{d}_i$, $d_i(j)$ | Individual demand vector of request-$i$, Demand of request $i$ for resource $j$. |
| $c_i(j)$ | the resource amount allocated to request-$i$ for resource $j$. |
| $P$ | Target number of VMs. |
| $\mathbb{E}$, $E_i(t_i)$ | the set of selected nodes for all the requests, the selected access technology for the $i$-th request at time instant $t_i$ |
| $\mathcal{M}$, $M$ | the multiple attribute set, number of the multiple attributes |
| $m_i$ | i-th attribute of $\mathcal{M}$ |
| $w_i$ | The weight of each attribute $m_i$ |
| $\mathbf{A}_s$ | $K \times M$ decision matrix for a given request $s \in \mathcal{S}$ |

## 1.2 Contributions

The main goal of this paper is to propose a framework on taking appropriate and intelligent scaling and placement decisions and embedding it into network management and orchestration processes targeting a digital twin service. The proposed method is mainly divided into four parts: The first part is the auto-scaling in cloud using the

demands of each VNF and cloud level observations. The second part is the scaling in transport where transport level requirements can be satisfied either by adding additional links or path level optimizations (e.g. re-rerouting or upgrading the links' quality). The third part is the application of the most suitable MADM algorithm for each scaled container to place them into appropriate nodes in the cluster considering the trade-off between the service demands and infrastructure level observations. The last part is the final roll-out of the intended service that satisfy the requirements.

In the rest of the paper, first we formally formulate the joint scaling and placement problem then discuss the difficulties in solving it, and present two separate heuristic approaches to solve the proposed optimization problem in scaling and placement domain separately. We also conduct extensive simulations to compare the performance of the proposed heuristics against simple scaling options which include either scaling in same node only (i.e. horizontal scaling) or cluster-scaling based on different placement algorithms in various scenarios depending on the availability of Enough Cloud Resources (ECR) and Enough Transport Resources (ETR). The considered metrics are overall deployment cost and VNF satisfaction rates. Simulation results show that a well-structured scaling and placement structure generally decreases the overall deployment cost. More precisely, the followings are our main contributions:

- We propose a new architecture and framework which brings together network management and orchestration planes with Artificial Intelligence (AI)/ML plane.
- Using a digital twin dataset, we jointly utilize transport and cloud resources (network and cloud aware schedulers for scaling and placement) that can assist in taking effective and intelligent scaling and placement decisions given the available resources and the demand requirements to satisfy the VNF service requirements in a dynamic manner. During placement decision making process, proposed methodology aims to reduce the deployment cost while considering different factors in both transport domain (such as latency, throughput) and in cloud domain (CPU utilization).

– Our simulation results indicate that considering both transport and cloud parameters in various resource limited environments has significant potential to provide increased VNF request satisfaction rates. In particular, when VNF scaling and placement using the TOPSIS scheme (one of the investigated MADM techniques) is performed, higher improvements and enhancements (low deployment cost and high VNF satisfaction ratios) to the digital twin service can be obtained.

The rest of the paper is summarized as follows: Section 2 is presenting the proposed general network architecture for scaling and placement operations in a Service Orchestrator (SO) as well as the scaling operation workflow. Section 3 presents the system model and the problem formulation. Section 4 describes the digital twin use case, the algorithm for proposed cloud and transport aware auto-scaling and summarizes MADM algorithms for the considered node selection problem. Section 5 is presenting the simulation environment, the dataset characteristics, and the evaluation results and general discussions. Finally Section 6 gives the conclusions. Additionally, Table 1 provides all symbols and their corresponding definitions used throughout the paper.

## 2 General Architecture and Workflow

One of the challenges of building scalable applications and systems is to decide on how to exchange information between system components while keeping the flexibility of modifying the interfaces without major impact on the on the overall existing design. Fig. 2 shows the general diagram of the proposed 5Growth *5G-enabled Growth in Vertical Industries*[9] architecture operating scaling and placement operations in the SO module. Artificial Intelligence/Machine Learning platform (AIMLP) provides a common API to provide services for 5growth stack namely Vertical Slicer (VS), SO, Resource Layer (RL) and Vertical Oriented monitoring system (VoMS) as detailed in 5Growth project, hence no need for individual interfaces to each of them in case

[9] https://5growth.eu/, Accessed: September-2021

AI/ML-aided operations are needed. Fig. 2 also marks the data producers and consumers at each module of the 5Growth stack. VoMS acts as the producer to AIMLP as well as consumer to 5Growth infrastructure where all network devices produce data into. AIMLP and 5Growth stack act as the consumers. According to this 5growth architecture, which is based on the European Telecommunications Standards Institute (ETSI) NFV architecture and specifications, the AIMLP performs model training, while other building blocks, e.g. VS, SO orRL are responsible for the ML inference tasks [32]. For AI/ML activities, 5Growth stack components rely on i) the available VoMS, which is the block of the 5Growth architecture in charge of collecting monitoring data from deployed network services and Network Function Virtualization Infrastructure (NFVI); and ii) the integrated data engineering pipeline in charge of data ingestion, data processing, analysis, visualization, and data management.

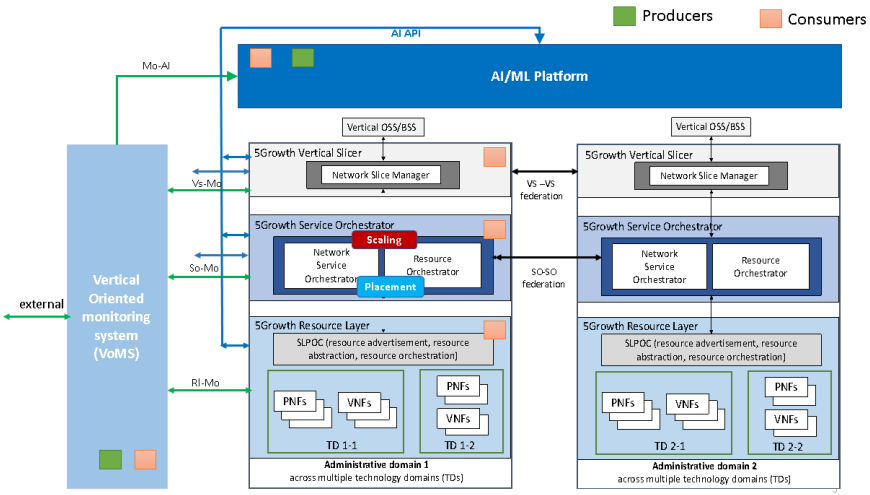



Fig. 2: 5Growth architecture with scaling and placement operations in Service Orchestrator.

In case ML model based solution is preferred, during AI/ML-based VNF scaling and placement operations, together with the existence of a new information element (IE) extending the ETSI NFV-IFA 014 Network Service Descriptor (NSD) template,

the 5Growth SO which is in charge of the lifecycle management of network services, is evolved to coordinate the process. This new IE is used to express the need of interaction with the AIMLP to configure AI/ML-based decisions for a given MANO problem (in this case "scaling") and specifies the metrics out of the ones already defined for this kind of network service in the NSD field "monitoredInfo" required by this AI/ML problem to perform its decisions. Based on some contextual information and the required monitoring information, the 5Growth SO launches an inference job which will decide the best instantiation level in the current network conditions, triggering the scaling operation if the decided instantiation level does not coincide with the current instantiation level. A complete description of the architectural evolution of the 5Growth SO and its operational workflow to support the scaling-based operation is in [32]. As an example usage of the architecture of Fig. 2, the general flow of scaling and placement is as follows:

1. **Connect to and ingest data from VoMS,** The scaling process starts with collecting the necessary measurements from the underlying system. The Virtual Infrastructure Manager (VIM) at the 5Growth RL sends data to 5Growth VoMS regarding radio access network (RAN)/Edge infrastructure that consists of the domain specific network metrics and measurements. Some of the examples of such data are CPU or I/O load, memory consumption, application specific metrics (number of served users, caching memory, thread queue depth, etc). Those standard metrics are provided by NFVI or embedded platform or from standard templates. Interfaces used to collect relevant data are So-Mo (NorthBound Interface (NBI)) for SO and Rl-Mo (NBI) for 5Growth RL. Several tools and options are available for data collection and ingestion purposes. For data connection Kafka connect, HTTP APIs and for data ingestion purposes Apache Kafka, Apache Pulsar, Spark Streaming or Flink Data Streams can be utilized. Some examples of underlying infrastructure related observations from 5Growth SO point of view are:
   - **CPU resources:** Allocated CPU Capacity, Available CPU Capacity, Reserved CPU Capacity, Total CPU Capacity,

- **Memory resources:** Allocated Memory Capacity, Available Memory Capacity, Reserved Memory Capacity, Total Memory Capacity,
- **Storage resources:** Allocated Storage Capacity, Available Storage Capacity, Reserved Storage Capacity, Total Storage Capacity,

2. **Provide data to AIMLP via general APIs:** The data that needs to be aggregated from multiple data sources are consumed by the AIMLP based on the topic name subscription. Obtaining the training dataset to be used to build ML models and necessary hyper-parameters to fine-tune the selected algorithm are done in this step inside AIMLP.
3. **Train ML models:** In case ML model based solution is preferred, the historically collected data with labels are used for training to obtain a ML model. For example, labelled data can be used to obtain a random-forest-classifier model which can later be used by real-time collected data for inference purposes [32,33] as outlined in next steps.
4. **Retrieve Trained Model through REST-API interfaces** So-AI interface for SO and Rl-AI interface for RL are used to download the trained model by communication with AIMLP via REST-APIs. An archive file (e.g. a pre-built jar file) that includes the binary versions of the algorithm and the relevant information to acquire related measurement data, e.g. a topic name, can be gathered from a common message queue that is present between the AIMLP and the SO.
5. **Auto-scaling Algorithms:** In case algorithm based approach is selected for scaling and placement, AIMLP calculates the scaling decisions based on the outcome of the algorithm and also executes the placement algorithms to select the best location of the scaled VNF. For example as described in sections below, a cloud scaling algorithm can be used to determine the required number of VMs for scaling and a MADM based algorithm can be used to select the optimal nodes based on the demands and the availability of resources in the cluster.
6. **Retrieve real-time monitoring data from the VoMS for continuous evaluations:** The scaling and placement algorithm subscribes to relevant internal messages received from the message queue component of VoMS and applies scaling algorithms that may trigger autonomous scaling or suggest scaling ac-

tions as well execute the placement algorithms to select the best available VM locations. In this step, periodically a streaming job running on 5Growth SO ingests the real-time monitoring data requested from the data ingestion module. In case ML model based solution is selected, it performs real-time inference over the trained model using this data. In case auto-scaling and placement algorithm based solution is selected, it executes the logic of algorithm. Later, it notifies the result (i.e., the best scaling decision given the current context or number of instances to remove or add, etc. as well as the best locations) to the 5Growth SO.

7. **Scaling and placement** decision: The 5Growth SO checks the notification, and if triggering the scaling and placement operation is ordered to service lifecycle management system, performs it through the southbound interface of the 5Gr-SO. The scaling and placement order can be either simple "scale service" type of operation where the fixed scaling step is connected to VNF Descriptor (VNFD) template or more complex one where the ML/AI algorithm decides how many instance of a particular VNF, service chain component or service need to be added or removed at selected locations. The scaling and placement decisions inside the clusters are provisioned if existing VMs fail to schedule on any existing nodes due to insufficient available resources. For this reason, adding a new node with the same specifications to the current one can redistribute the load in case the cluster has not reached maximum node count. Note that during scaling decisions, the algorithms should consider the time required for scaling as well as the rate of change in the capacity. Moreover, the resources and instances can be heterogeneous which may yield different maximum and instantaneous capacity.
8. **Final roll-out:** The scalable Virtual Network Function Container (VNFC) at selected locations can either run as part of a VNF, executing on NFVI, or a Physical Network Function (PNF) on some infrastructure. The VNF/PNF archive includes the algorithms and meta-data needed to support scaling and placement of the VNFC.

## 3 System Model and Problem Formulation

### 3.1 System Model

We consider a joint VNF scaling and placement problem. Our scenario addresses the need for dynamic allocation of resources and demands for the requested network service requirements on the network infrastructure. VNF scaling aims to scale up/down or in/out depending on the requests and available resources in cloud servers meeting the compute requirements of the VNF. VNF placement algorithm aims to place the scaled VNFs with desired requirements of service both in transport (e.g. bandwidth, latency) and cloud (e.g. CPU, RAM) domains.

Formally, we consider a set of multiple available cloud nodes given by $\mathcal{K} = \{1, 2, \ldots, K\}$ with $K$ nodes. Let the set of requests in the available infrastructure is given by $\mathcal{S} = \{1, 2, \ldots, S\}$ with $S$ requests. Let $\mathcal{S}_k \subset \mathcal{S}$ be the set of requests placed into node-$k$ with $S_k$ requests. Also let the set of equipment used to reach to node-$k$ for a given VNF request-$i$ be given by $\mathcal{F}_k^i = \{1, 2, \ldots, F_k^i\}$ with $F_k^i$ equipment where equipment can be edge nodes, cells, storage elements, routers, switches, gateways, core network elements, etc. Each VNF request can be allocated using different number of equipment. Moreover, let $\mathcal{F} = \cup_{k \in \mathcal{K}} \cup_{i \in \mathcal{S}_k} \mathcal{F}_k^i$ be the set of all equipment used to reach node-$k$ including node-$k \in \mathcal{S}_k$. Let $\mathcal{B}_{k,j}^{i,f}$ be the set of resources utilized after VNF allocation at equipment-$f \in \mathcal{F}_k^i$ with $B_{k,j}^{i,f}$ resources where $j \in \mathcal{R} = \{1, 2, \ldots, R\}$ denotes the type of the resource and $R$ is number of different types of resources (e.g. CPU, memory and storage available at data centers of node $k$, communication bandwidth available at cells and transport networks, link capacities available at transmission network equipment). To meet the requirements of transport and cloud network requests, each equipment-$f \in \mathcal{F}_k^i$ can be selected from a subset of the resources in $\mathcal{B}_{k,j}^{i,f}$.

Let us assume that the capacities of $R$ resources are represented by an $R$-dimensional vector $\mathbf{C}$ where each element $C_j$ represents the capacity of resource $b_j \in \mathcal{B}_{k,j}^{i,f}$ where subscripts $k, i$ and $f$ are suppressed for simplicity and node $k \in \mathcal{K}$. Similarly, the observed metric values are represented by an $R$-dimensional vector $\mathbf{O}$

where each element $O_j$ represents the observed value of resource $b_j \in \mathcal{B}_{k,j}^{i,f}$ and node $k \in \mathcal{K}$. We denote by $\mathbf{d}_i$ the individual demand vector of VNF request-$i \in \mathcal{S}_k$, and $d_i(j)$ as the demand of request $i \in \mathcal{S}$ for resource type $j \in \mathcal{R}$ after VNF placement. Similarly, we use $c_i(j)$ to denote the resource amount allocated to VNF request-$i \in \mathcal{S}_k$ for resource type $j$ and $\mathbf{c}$ as the resource allocation vector for all request $i \in \mathcal{S}$ for resource type $j \in \mathcal{R}$. Under the light of these assumptions, we have the following set of constraints for the VNF scaling problem for $0 \leq j < R$ and $0 \leq i < S$,

$$c_i(j) = \{0, d_i(j)\}, \;\; C_j^{min} \leq \sum_{i=0}^{S-1} c_i(j) \leq C_j^{max}, \;\; \forall i \in \mathcal{S}, \forall j \in \mathcal{R}. \tag{1}$$

where $C_j^{max}$ and $C_j^{min}$ are the maximum and minimum capacities for each resource type $j$.

### 3.2 Problem Formulation

**Objective:** Our final step is to determine the objective function. The high-level goal of the scaling operation is maximize the number of allocated VNF requests of all $S$ VNF requests with minimum overall deployment cost, while considering various resource type (bandwidth,latency and CPU) costs by judiciously selecting the appropriate PoPs for each scaled VNFs. Therefore, we are interested in the minimization of the overall average deployment cost as given by (2). For a given resource type $j$, we can quantify the discrepancy for VNF request-$i \in \mathcal{S}_k$ as $d_i(j)$ and $c_i(j)$ are not the same that can be treated as the error term that needs to be minimized. To be able to normalize the soft error terms coming from different resources (potentially with different domains and support), this term is also divided by $d_i(j)$. In summary, this gives us the following optimization problem:

**Problem 1 (Original joint transport and cloud-aware node scaling)**

$$\underset{\mathbf{c}}{\text{minimize}} \sum_{i=1}^{S} \sum_{j=1}^{R} \left\lceil \beta_i(j) \times \frac{d_i(j) - c_i(j)}{d_i(j)} \right\rceil, \tag{2}$$

subject to

$$c_i(j) = \{0, d_i(j)\}, \quad 0 \leq j < R, \quad 0 \leq i < S, \tag{3}$$

$$C_j^{min} \leq \sum_{i=0}^{S-1} c_i(j) \leq C_j^{max}, 0 \leq j < R, \tag{4}$$

$$\beta_i(j) \in \{0, 1\}, i = 1, 2, \ldots, S, j = 1, 2, \ldots, R \tag{5}$$

in which the ceiling function is used to map soft information to hard decisions "0" for satisfied and "1" for unsatisfied states and $\beta_i(j)$ is zero for transport network resources when all nodes are inside the same region (PoP), i.e. no transport domain parameters are used for $j \in \mathcal{R}$ and $i \in \mathcal{S}$ else it is 1. Note that if the scaling decision is not precise, it can increase the SLA violations, which increases the deployment cost and also reduces the VNF request satisfaction rate. Then, the demand $d_i(j)$ is called *satisfied* if it is equal to $c_i(j)$. Then VNF average request *satisfaction ratio* can be defined for the set of all requests as:

$$\eta(\mathcal{S}) = \frac{1}{S} \sum_{i \in \mathcal{S}} \prod_{j \in \mathcal{R}} \mathbb{1} \{(d_i(j) = c_i(j))\} \tag{6}$$

where $\mathbb{1}\{...\}$ is the indicator function which takes 1 if the statement holds and 0 otherwise.

Note that in the above formulation, we assume that the demands for each resource are known apriori. However, it is also worth mentioning that sometimes the demand for one resource is known apriori whereas a model can be used to predict the demand for another resource. For example, it is sometimes easy to predict that the incoming traffic would be higher during a specific event, but it is hard to foresee the consequences on CPU usage and memory. As it is previously described in the 5Growth platform, all requests are assumed to be known precisely the resources that they need. However, it is also possible to accommodate requests that only demand some specific resources and the others are inferred from past experiences using the models in and interaction with the AI/ML platform.

### 3.3 Computational Complexity

This section analyzes the computational complexity of the considered problem. It is shown in [34] that even the simple case when requests consist of only one VNF, each request is compatible with every cloud data center and bandwidth is disregarded, the problem is strongly NP-hard. One solution to the considered problem would be to enumerate all possible joint scaling and placement options, which is challenging because the number of possible scaling and placement options exhibits exponential growth. In the case of placement, to compute (2), a centralized agent need to evaluate the total network cost for $K^S$ possible request vector combinations where $K$ is the total number of nodes available after scaling in cloud to place each scaled VM and $S$ is the number of requests. For example, for a simplistic network size with $K = 2$ available nodes when $S = 100$ requests arrive, the search space is $2^{100}$ strategy profiles. Consequently, finding the centralized VM placement is cumbersome in large-scale networks.

The other solution is to figure out the cost of each scaling and placement options. This part cannot be directly decomposed into smaller problems. Given its hardness, to alleviate the complexity problem while maintaining good performance results and cope with the complexity of the huge state space accompanied with the optimization problem, in the following sections we divide the problem into corresponding sub-problems (first solving scaling problem and later the placement problem) and devise an algorithm to solve them heuristically or approximately in an iterative manner that considers both transport and cloud parameters.

## 4 Algorithm Development for a Digital Twin use case

### 4.1 Digital Twin Use Case

Various services have different transport and cloud requirements depending on whether they are customized applications that are compute-intensive, latency-sensitive or both. For example, Augmented Reality (AR)/ Virtual Reality (VR) and gaming applications demand high throughput rather than reliability, whereas e-health and

vehicles may need reliability and stability rather than high throughput connectivity. VPN-as-a-Service are both latency and bandwidth sensitive. Intrusion detection systems are compute-intensive whereas distributed DNS services are latency-sensitive [26]. Some types of VNF (e.g. firewall or video codec [35]) can also demand different bandwidth for inflow and outflow traffics, respectively. In this paper, we consider a digital twin use case which has cloud (CPU) and transport (bandwidth, latency) domain requirements. Digital twin is essentially a virtual representation of something which exists in the real world as physical assets, processes, people, places, systems and devices connected in real-time thanks to continuous data streaming. It is created and maintained to answer questions about the physical counterpart of a physical asset, i.e. its physical twin in a near real-time setting. For Industry 4.0 applications, such replicas can provide a new layer of engineering insight to improve the product performance and advance the existing ones to the next generation version.

The experimented single robot digital twin stack consists of two components (edge server and robot manipulator) and three VNFs as given in Fig. 3. Edge server (consisting of one VNF with digital twin app and interface layers and the third VNF that contains the motion planning and control layers) and robot manipulator (consisting of one VNF that contains the robot drivers.) Robot driver VNF receives navigation commands from the motion planning and control. Robot driver VNF directly interacts with robot hardware and ensures receiving navigation commands from motion planning and control at the edge of network and stream sensor data (mainly robot joint states) to edge server. Motion planning and control layer at the edge network has two sub-layers. Control layer receives navigation commands, runs a control loop following a given frequency towards the robot drivers and receives robot operational states. Motion planning layer ensures finding inverse kinematics and building a path for the robot that consists of a series of navigation commands sent from control layer. The commands are also validated before the robot is able to execute them. Finally, the digital twin application implements 3D models to represent the virtual model of the physical robot while offering remote control mechanisms.

More details about the digital twin application of the utilized experiment can be found in [36] and European 5GPPP project H2020 5G-DIVE website[10]-2021.

We assume that a digital twin motion planning control application in which adding more end-robots requires scaling in cloud and also in transport networks to serve better the motion control of those additional robots. This is because that layer is introducing constraints regarding the latency and resources (mainly CPU and RAM). Therefore, we assume the scaling problem of the most demanding latency requirements instance of motion planning layer of the digital twin service described above. This single VNF is interacting with multiple robots. The placement of scaled VMs depends on where to place by considering multiple factors, such as network bandwidth, latency, privacy and security levels, etc. Therefore, the best placement policy should create a balance between performance and cost. If we have a service in which scaling may mean adding more computing only whilst maintaining the transport resources, some of the discussions below would not hold. By default we assume that when scaling out, the new VM will be deployed in the same PoP as the original one (if there are resources). Fig. 3 also shows the illustration of scaling out and up operations in cloud and transport based scaling over multiple potential nodes. There are four PoPs and each one is labelled based on the condition of cloud and transport domain parameters as Most Suitable, Less Suitable, Even Less Suitable and Not Suitable.

### 4.2 Cloud-aware scaling:

One straightforward example of cloud-aware scaling is similar to an open-source implementation in Kubernetes. The Kubernetes Horizontal Pod Autoscaler (KHPA) controller can operate on the ratio between desired metric value and current metric value[11]:

$$dR = \left\lceil cR \times \frac{cMV}{dMV} \right\rceil$$

[10] https://5g-dive.eu/, Accessed: September

[11] https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/

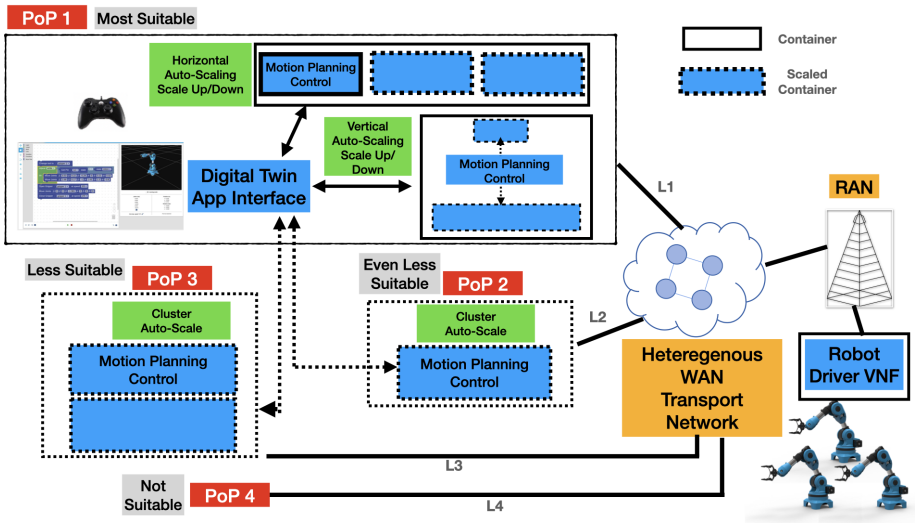



Fig. 3: Scaling up and out operations of the motion planning control VNF in cloud and transport domain over multiple potential nodes.

where $dR$ is desired number of replicas, $cR$ is the current number of replicas, $cMV$ is the current value of the observed metric (e.g. CPU, RAM), $dMV$ is the desired value of the observed metric. For example, if $cMV = 200m$ (millicore of CPU) which corresponds to 20% of CPU utilization, and $dMV = 100m$, the number of replicas will be doubled, since $200.0/100.0 == 2.0$. If instead $cMV = 50m$, we'll halve the number of replicas, since $50.0/100.0 == 0.5$. The scaling is skipped if the ratio is sufficiently close to 1.0. When a $tAV$ (target average value) or $tAU$ (target average utilization) is specified, the $cMV$ is computed by taking the average of the given metric across all pods in the horizontal pod autoscaler's scale target in a given $\tau$ rolling windowing duration.

Algorithm 1 shows the pseudo code of CA-HVA (Cloud-Aware Horizontal VM Autoscaling) algorithm utilized in this paper for comparison purposes. The desired CPU utilization $d_i(j)$ for $j = cpu$, and the set of active VMs $AVM$, deployed in the previous control period ($\tau$ seconds before are taken as input to KHPA and $C_j^{min}$ and $C_j^{max}$ are defined as the minimum and maximum capacity of the server

where the scaling occurs (e.g. the number of VMs to instantiate) for resource type $j = \{cpu, ram\}$ [5]. Target number of VMs $P_{cpu}$ is defined as,

$$P_{cpu} = \left\lceil \frac{\sum_{i \in AVM} \mathbf{c}_i(cpu)}{\mathbf{d}_i(cpu)} \right\rceil \tag{7}$$

**Example:** Assume that we have the following desired requirements provided by the digital twin service: $\mathbf{d}_i = (\mathbf{d}_i(cpu), \mathbf{d}_i(ram)) = (50\%, 1GB)$ for a given request-$i$. Assume that $\mathbf{d}_i(cpu) = 50\%$ (desired CPU utilization) for a given resource $j = cpu$, three application replicas are running, and the per-VM CPU utilization is $\mathbf{c}_1(cpu) = 70\%$, $\mathbf{c}_2(cpu) = 80\%$ and $\mathbf{c}_3(cpu) = 30\%$, respectively. At the next control period, the CA-HVA algorithm determines that a new VM should be deployed $P = \left\lceil \frac{\mathbf{c}_1(cpu)+\mathbf{c}_2(cpu)+\mathbf{c}_3(cpu)}{\mathbf{d}_i(cpu)} \right\rceil = \lceil \frac{70+80+30}{50} \rceil = 4$. The load will be distributed among the $P_{cpu} = 4$ VMs and the estimated per-VM utilization becomes $c_i(j) = \sum_{i=1}^{P_{cpu}} (\mathbf{c}_i(cpu)/P_{cpu}) = 45\%$.

Similarly for the RAM analysis, assume that $\mathbf{d}_i(ram) = 3GB$ (desired RAM utilization), three application replicas are running, and the per-VM RAM utilization is $\mathbf{c}_1(ram) = 7GB$, $\mathbf{c}_2(ram) = 8GB$ and $\mathbf{c}_3(ram) = 3GB$, respectively. At the next control period, the CA-HVA algorithm determines that a new VM should be deployed $P_{ram} = \left\lceil \frac{\mathbf{c}_1(ram)+\mathbf{c}_2(ram)+\mathbf{c}_3(ram)}{\mathbf{d}_i(ram)} \right\rceil = \lceil \frac{7+8+3}{3} \rceil = 6$. The load will be distributed among the $P_{ram} = 6$ VMs and the estimated per-VM RAM utilization becomes $c_i(j) = \sum_{i=1}^{P} (\mathbf{c}_i(ram)/P_{ram}) = 3$.

Finally, to calculate the final number of additional links, we take the maximum of above results, i.e. $P = max(P_{cpu}, P_{ram}) = max(4, 6) = 6$ VMs are required to satisfy the requirements of digital twin service.

### 4.3 Proposed cloud and transport-aware autoscaling:

We are interested in maximizing the service availability and favor new physical hosts with the most available resources. After the number of VMs are selected, we need to check desired transport level parameters observed in SO, i.e. parameters such as available bandwidth, total bandwidth and network QoS (link cost, link delay), etc. Note that in case scaled VM instances need to be deployed in different PoPs as a

**Algorithm 1:** Cloud-Aware Horizontal VM AutoScaling (CA-HVA) Algorithm

**Input:** $d_i(j)$, $c_i(j)$, $AVM$, $C_j^{min}$, $C_j^{max}$, $\forall i \in \mathcal{S}$, $\forall j \in \mathcal{R}$
**Output:** $P$ `// Number of scaled VMs`
**while** TRUE **do**
  **Set:** $\mathcal{U}_j = \emptyset$, $j \in \mathcal{R}$
  **for** $i \in AVM$ **do**
    Get CPU and RAM Utilization $\mathbf{c}_i(cpu)$, $\mathbf{c}_i(ram)$ `// Average values of previous` $\tau$ `sec.`
    $\mathcal{U}_j = \mathcal{U}_j \cup \{\mathbf{c}_i(j), j \in \mathcal{R}$
  **end for**
  **Set:** $P_j = \left\lceil \frac{sum(\mathcal{U}_j)}{\mathbf{d}_i(j)} \right\rceil$, $P = max(P_j)$, $j \in \mathcal{R}$
  **if** $P \geq C_j^{max}$ **then**
    $P \leftarrow C_j^{max}$
  **else if** $P \leq C_j^{min}$ **then**
    $P \leftarrow C_j^{min}$
  **end if**
  wait($\tau$)
**end while**

result of the decision process, transport related metrics such as latency, bandwidth will also be impacted at each PoP.

Let $\mathbb{E} = \{E_1(t_1), \ldots, E_{N_O}(t_{N_O})\}$ denote the set of selected nodes for all the requests based on the values of the multiple attribute set $\mathcal{M}$, where $E_i(t_i) \in \mathcal{E}$ denotes the selected access technology for the $i$-th request at time instant $t_i$. The multiple attribute set is denoted by $\mathcal{M} = \{m_1, m_2, ..., m_M\}$ which consists of elements such as throughout, average latency of the target node for the given application, CPU utilization etc. values where $D$ refers to size of the multiple attribute set $\mathcal{M}$. We define the weight of each attribute $m_i$ as $w_i$.

**MADM Algorithms for Node Selection:** The node selection is subject to computation resource (CPU, RAM) and transport (bandwidth and latency), hence the location constraints. As the first part of the optimization step in the framework, the best placement of containers to selected nodes after scaling in cloud is decided based on MADM algorithms. MADM methods are widely used for making preference deci-

sions when there are several options and conflicting aspects (attributes) under given priorities (weights). In the literature, several algorithms are shown to be useful for ideal placement of a container problem. In the following sections we describe Total Order Preference By Similarity to the Ideal Solution (TOPSIS) [37,38], simple additive weighting (SAW), multiplicative exponent weighting (MEW) [39], GRA [40] and ELimination and Choice Expressing REality (ELECTRE) [41,42] algorithms. A good simulator showcasing the outcome of some of MADM algorithms can be found in this website[12] and more details about those MADM algorithms are given in Appendix A.

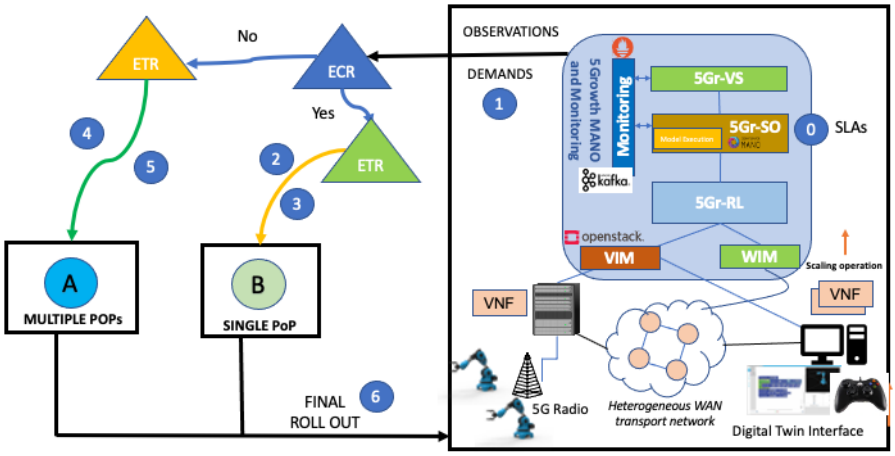
A general flow diagram of the proposed scheme is also given in Fig. 4. A summary of the proposed extended version of the algorithm named as CTA-CVA (Cloud and Transport Aware Cluster VM Autoscaler) is provided as follows following 7 steps:

0. **Check SLA conditions:** The service providers or Mobile Network Operators (MNOs) check whether the SLA conditions are met for the requested demands;
1. **Collect:** The observations and demands from MNOs and service providers are collected in this step;
2. **If ECR and ETR:** There are ECR in original PoP to deploy new VM/or to scale up ECR as given in YES step-2 of Fig 4(a) as well as ETR towards original PoP to make the old traffic not be distorted by the new one and to make the new one reach the new VM without SLA violations as given YES in step-2 of Fig 4(a).
   - **Scale out:** In this case, we simply scale out in the same PoP and "scale" the transport by adding the new logical link towards the rest of the VNFs of this same NS placed in other PoPs. We also add load balancers in cloud.
   - **Scale up:** If scaling up transport and cloud is done together, we also need to add transport resources (e.g. to serve more robots that need digital twins.)
     (a) **Scaling in Cloud:** At the given location, trigger scaling operation using CA-HVA Algorithm 1,

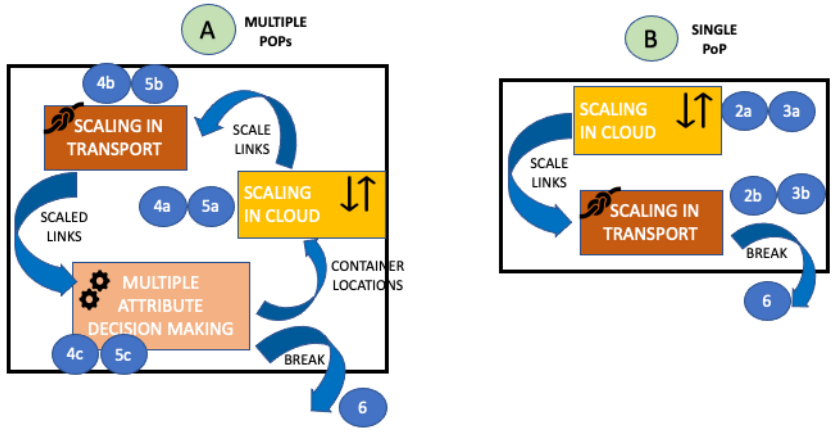
[12] https://decision-radar.com/

(b) **Scaling in Transport:** Scaling out/in the number of possible paths between demanded and all physical nodes based on bandwidth, latency, etc demands and the available utilization.

3. **IF ECR and NOT ETR:**
   - **Scale out:** The new VM must be deployed in another PoP and a new Logical Link (LL) towards that PoP is selected and connected to the rest of PoPs with VNFs of this service. If the original VNF and the new one must be connected and not ETR are available for this communication, the scaling operation is not possible.
   - **Scale up:** In the scale up example, this would imply a migration of the VMs (and scaling up) to a new location with ECR and ETR, removal of old LL to old PoP and setting up the new LL to the new PoP.
     (a) **Scaling in Cloud:** Similar to step-2.
     (b) **Scaling in Transport:** Similar to step-2.
4. **IF NOT ECR and ETR:**
   - **Scale out:** Deployment of VM in new PoP with ECR and selection of LL to connect it to rest of VNFs of this service located in other PoPs
   - **Scale up:** Migration of the VM (and scaling up) to new PoP with ECR and ETR and removal of old LL towards old PoP and set up of new one.
     (a) **Scaling in Cloud:** Similar to step-2.
     (b) **Scaling in Transport:** Similar to step-2.
     (c) **Place w/MADM:** If the remaining resource in node can satisfy the new required capacity demands after scaling in cloud, go to step 5, else for the given service observations and demand, run MADM algorithms using pseudo-code given Algorithm 2 and find a set of neighboring physical candidate nodes. If node capacity and type constraints are not met, move to step-2 else go to step-5 to finalize the rollout in infrastructure.;
5. **IF NOT ECR and NOT ETR:** Exactly the same as case ECR and not ETR given in step-3.
6. **Final roll out of the VMs:** to optimal locations inside the infrastructure.




(a)




(b)

Fig. 4: (a) General diagram of flow of the proposed scheme. (b) Multiple PoP and Single PoP scaling.

**Algorithm 2:** Utilized methodology to run each MADM Algorithms for given nodes, requests and resource types.

**Input:** $\mathbb{E}$, $d_i(j)$, $c_i(j)$, $\forall i \in \mathcal{S}$, $\forall j \in \mathcal{R}$ ; `// Set of nodes to place VMs, demands and capacities for all requests and resource types`
**Output:** $e^*(s)$, $\forall s \in \mathcal{S}$ ; `// Set of selected nodes of VMs best satisfying the request` $\mathtt{d}_i$
**for** $s \in 1...S$ **do**
  **for** $i \in 1...K$ **do**
    **for** $j \in 1...R$ **do**
      **if** $j \in \{cpu, throughput\}$ **then**
        **Calculate:** $a_{ij} = C_j - O_j$ ; `// available resources for throughput and cpu`
        **Obtain:** $a_{ij} = a_{ij} - d_i(j)$
      **else**
        **Calculate:** $a_{ij} = C_j$ if $(O_j \geq C_j)$ else $O_j$ ; `// available resources for latency`
        **Obtain:** $a_{ij} = \frac{1}{|d_i(j) - a_{ij}|}$ if $(d_i(j) > a_{ij})$ else $d_i(j) - a_{ij}$
      **end if**
    **end for**
  **end for**
  Obtain $\mathbf{A}_s = [a_{ij}]_{K \times M}$ ; `// Build the availability matrix`
  Run MADM algorithms in Section 4.3 for $\mathbf{A}_s$
  Find $e^*(s)$ ; `// Get the optimum locations for given request and available resources`
**end for**

Note that the above proposed algorithms can either be implemented in a separate module (e.g. similar to a building block AI/ML outside the 5growth architecture of Fig. 2) or can be directly embedded as a functionality inside the service orchestrator (SO) as a new building block as also described in the given architectural figure. For example, AI/ML-aided scaling algorithms are already trained inside AI/ML building block and served inside SO in previous works of [43,44,32]. In our proposed architecture, scaling and placement decisions are taken by algorithms implemented at the SO level.

## 5 Simulation Environment, Dataset Characteristic and Evaluation Results

We used Python 3.6 to simulate all the studied schemes under evaluation. For implementation of the MADM algorithms given above, namely SAW, MEW, TOPSIS and ELECTRE we use PyPI[13]-2021 and pyDevisions[14]-2021 libraries. For evaluations, for PoP-1 we used links with maximum link capacity of 30 GB/s and 1.5 GB/s in case of ETR and Not ETR cases respectively. We used those selected values to create resource constraint environments in both cloud and transport domains considering the VNF requests and the available resources in the simulations. For latency cases, we used links with a maximum link latency of 1 ms and 100 ms in case of ETR and Not ETR cases respectively. In cloud, in case of ECR and Not ECR, we assumed 3 and only 1 active CPU cores respectively. For PoP-2 and PoP-3, we selected two and twenty times of the PoP-1's maximum link capacities respectively. PoP-2 and PoP-3 are also assumed to have 10 ms and 1 ms maximum link latency values respectively. One-hop connection assumed between the VNF requester and the server, hence no new routing opportunities are considered. In cloud, PoP-2 and PoP-3 are assumed to use 2 and 3 active CPU cores respectively. For comparisons, we use CA-HVA algorithm given in Algorithm 1 for cloud-based scaling and the methodology given in Algorithm 2 to select the best nodes after cloud scaling, transport scaling or cloud and transport scaling depending on the considered scenario.

### 5.1 Digital Twin Dataset

The goal of the digital twin dataset generation process is to obtain a valid dataset for VNF scaling and placement of robot manipulator based digital twin service. Results are obtained within the collaboration of *eDge Intelligence for Vertical Experimentation* (5G-DIVE) and 5Growth projects.

**Experimental methodology and dataset creation:** To find the threshold of the motion planning and control VNF and knowing that the control frequency of the

[13] https://pypi.org/project/mcdm/, Accessed: September

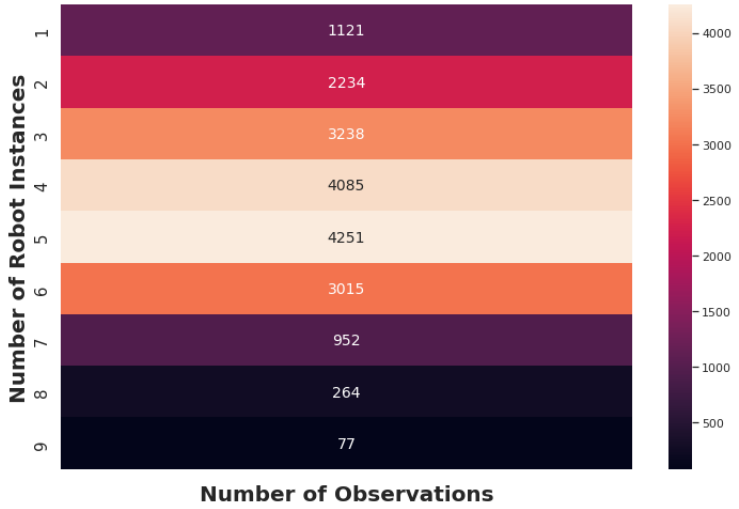
[14] https://github.com/Valdecy/pyDecisions, Accessed: September

robot manipulator is 20 ms, latency threshold for the digital twin service is set to 20 ms. During the first step of the experiments, the number of digital twins that will disrupt the latency, i.e. exceeding the 20 ms threshold in the Motion planning and Control layers VNF is investigated. After the threshold that disrupts the latency is found, the experiment is executed to obtain the the data set. To create the dataset, a new robot instance is instantiated on every 3600 s. The dataset features consist of: "timestamp, node, latency, action, CPU, RAM, transmitted bits, received bits, number of instances". Each entry is obtained based on the pre-selected time windows. The action field consists of:

- **Move Down** is an interface layer interaction via higher control loop where interface layer is a high-level abstraction of the motion planning and control functionality (move down in this case) and can be regarded as a gateway between the remote operator and physical robot to facilitate their interactions via a set of APIs,
- **Pick** is a control layer interaction via smaller control loop, i.e. receiving pick command and running a control loop following a given frequency towards the robot drivers,
- **Move Up** is an interface layer interaction via higher control loop where move up motion planning functionality is abstracted via interface layer.
- **Place** is a control layer interaction via smaller control loop where place command is sent towards robot drivers.

In the generated dataset, the latency of both the control and interface VNFs are available. They can be distinguished by the action command. "Move Up" and "Move Down" actions refer to the interface VNF (having a latency around 400 ms) and "pick" and "place" actions refer to the control VNF (with latency around 40 ms). Fig. 5 shows the heatmap for the number of observations for each instantiated robot instances. Note that due to real-world experiment observations, the amount of data collected for each robot instance may vary. From Fig. 5, the maximum amount of observations are obtained when the number of robot instances was five and the minimum amount of observations are when the number of robot instances was nine.

Without loss of generality, in the evaluation results we have used latency and transmitted bits as transport level parameters and CPU level as cloud level parameter during cloud & transport based auto-scaling and placement decision making process.




(a)

Fig. 5: Heatmap for number of observations for each instantiated robot instances observed during digital twin experiments.

## 5.2 Evaluation Results

**Requests to PoPs:** Fig. 6 shows the boxplot as well as the bar plot of median values with 95% confidence interval (CI) of observed transmitted bits per second, latency and CPU versus increasing number of robots in the considered digital twin dataset. Note that, as a general trend, when the number of robot instances increases, CPU utilization, latency and transmitted bits per second increase. However as observed in Fig. 6e, the latency values are slightly lower when we have more than seven numbers of robot instances. This is in fact due to the low number of real-world observations especially for robot instances of eight and nine (as observed from Fig. 5). When we observe the median values' CI ranges, we can notice that CI ranges are in fact larger

for the number of robot instances larger than seven. During our experiments, we have selected the requests from the median values of each robot instance. Therefore, there are a total of nine distinct randomly selected VNF requests over the considered observation period.

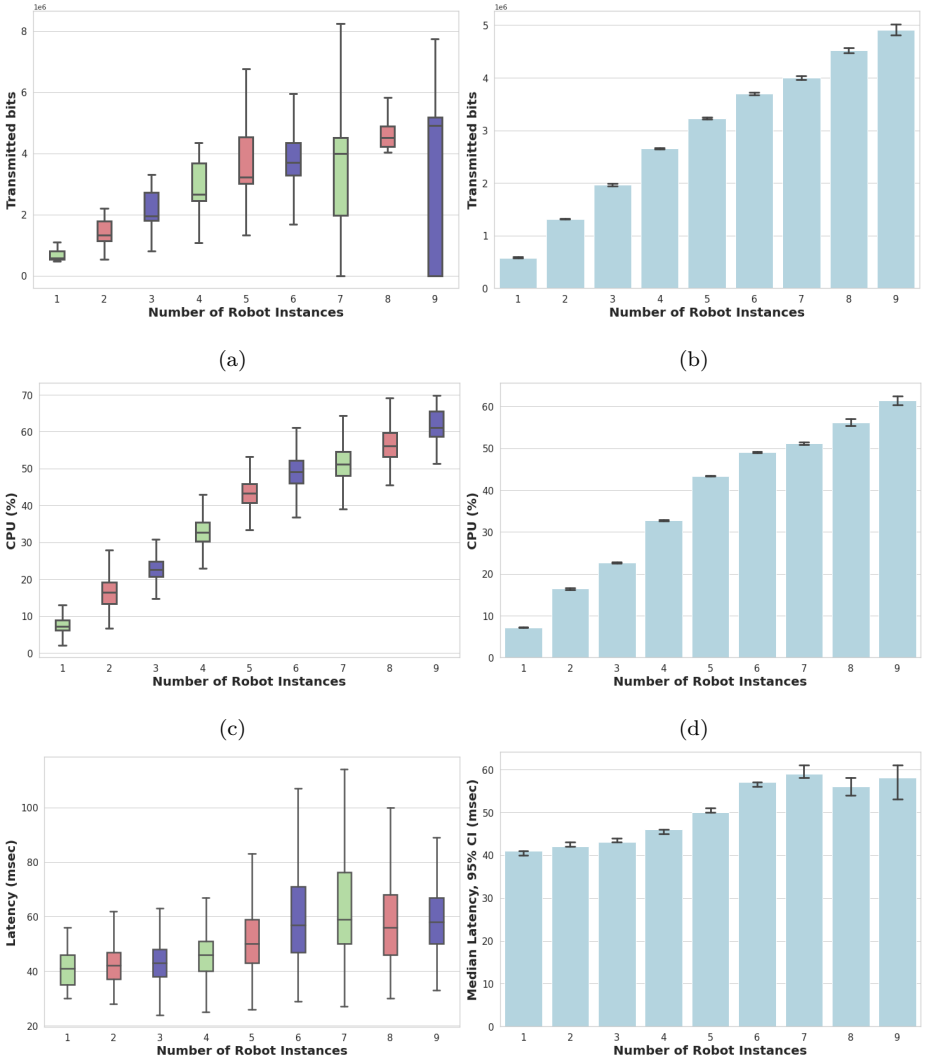
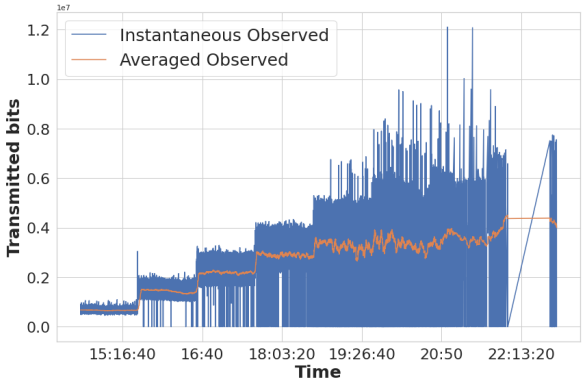



Fig. 6: Boxplot and barplot with 95% CI for increasing number of robot instances versus (a) Transmitted bits. (b) CPU (%) (c) Latency (ms).

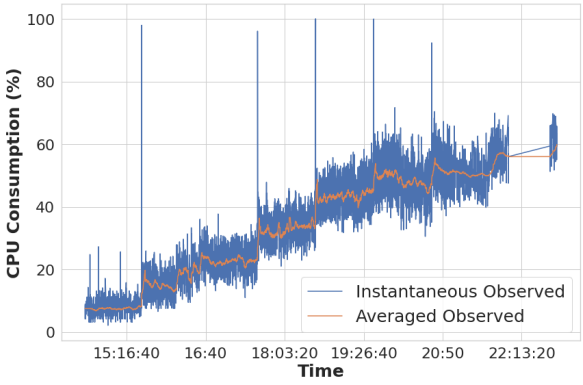
**Observations at each PoP:** In our placement algorithm, the decisions to scale and place are taken every $\tau$ seconds and the values obtained are the rolling averages based on the observed metrics including CPU and memory status of the container, latency, transmitted bps, etc. Fig. 7a, Fig. 7b and Fig. 7c show the transmitted bits, CPU consumption and latency values, respectively. The values are either instantaneous or window rolling averages over $\tau = 120$ seconds as each robot instances are added over time. In our analysis, the observations for each server are selected randomly from the rolling average of each of these transmitted bits, CPU and latency values and a total of 14793 observations are used.

Fig. 7 fluctuations are instantaneous observed values. Note that the digital twin application emulates the physical/end side, while utilizing resources at the virtual/computing side. Therefore, as the number of robot instances at the end side increases (due to the necessity of a higher number of robots), the amount of communication traffic between the control layer at edge and the robot driver increases as well. However, this increase is not strictly linear but fluctuates. One of the reasons for the spikes in Fig. 7 (more clearly visible in CPU consumption) is due to booting effects, because each time new robot instances are emulated, new processes are initiated by the system. Note that in our evaluations, we used averaged observations. Therefore behaviours such as CPU peaks every time a new instance is created as in Fig. 7b do not have a major effect on the observed metrics such as unsatisfied VNF requests or deployment cost.

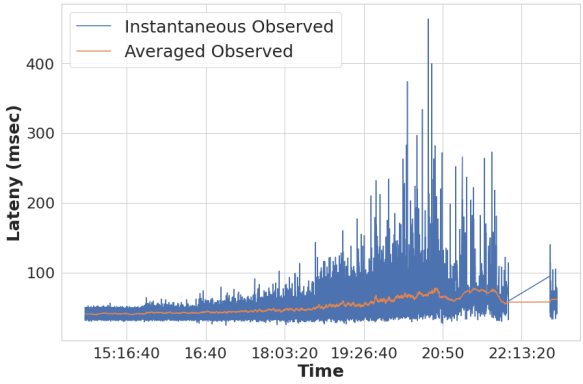
Fig. 8 shows the unsatisfied VNFs request percentages for no scaling, only cloud scaling, only transport scaling and transport & cloud scaling scenarios for (ECR and Not ETR), (Not ECR and ETR) and (Not ETR and Not ECR) cases when only one PoP is utilized, i.e. the case for horizontal or single PoP scaling scenario. Note that the cost values are calculated based on (6). In the case of (not ECR and ETR), we can observe that no scaling and only transport scaling cases gives the same 7.14% unsatisfied request VNF rates out of all considered VNFs requests. The reason is that in this case, there is ETR but doing only transport scaling does not reduce the unsatisfaction rates. However, performing only cloud scaling can reduce the percentage of unsatisfied requests to 3.45%. In cloud & transport scaling, we can




(a)




(b)




(c)

Fig. 7: Plots for observed values and their moving (window rolling) averaged mean values with $\tau = 120$ seconds versus time for (a) Transmitted (bits). (b) CPU (Consumption %) (c) Latency (ms).

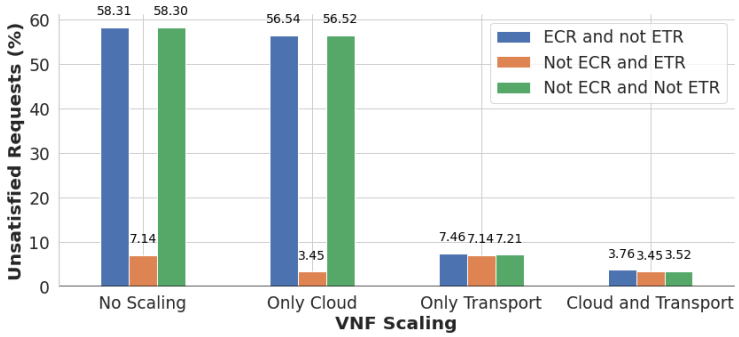



Fig. 8: Unsatisfied VNF requests percentages for no scaling, only cloud scaling, only transport scaling and transport and cloud scaling scenarios for (ECR and Not ETR), (Not ETR and ECR) and (Not ETR and Not ECR) cases under single PoP scenario.

observe the same 3.45% unsatisfied rate as with only cloud scaling. The reason is again due to ETR case where additional scaling in transport does not help to reduce the unsatisfaction ratios.

In the scenario of (ECR and Not ETR) of Fig. 8, as expected no scaling option gives the highest unsatisfaction ratio of 58.31%. Scaling in cloud only reduces unsatisfaction ratio to 56.54% and scaling only in transport further reduces it into 7.46%. The reason for high VNF unsatisfaction ratios observed in only cloud scaling in (ECR and not ETR) case is the following. Even though all the cloud based requests are satisfied thanks to cloud scaling, some of the transport based requests will still be unsatisfied due to not ETR. Therefore, this can result in higher final unsatisfaction ratios since final unsatisfaction ratio depends on the satisfaction ratios of both cloud and transport domains. On the other hand, scaling in both transport & cloud further reduces the unsatisfaction ratio to 3.76%. In the scenario of (Not ECR and Not ETR), similarly no scaling has the highest unsatisfaction ratio (with 58.3%) followed by cloud only (with 56.52%), transport only (with 7.21%) and cloud & transport scaling (with 3.52%). Note also that the case of (ECR and ETR) is not shown since all VNF requests can be satisfied in this scenario. In summary, the results in Fig. 8 indicate that joint cloud & transport scaling can reduce the unsatisfied VNF request considerably.

Fig. 9 shows the comparisons of obtained average deployment cost values of different MADM schemes with (ECR and Not ETR), (Not ECR and ETR) and (Not ECR and Not ETR) cases where all out of 179848 scaled requests are satisfied (both in transport and cloud domains after executing) cluster scaling. Note that the cost values are calculated based on (2) and transport & cloud level scalings are done before VNF placements. The deployment cost values in Fig. 9 are normalized with respect to horizontal scaling. In horizontal scaling scheme, no placement of VNFs are done, hence all new scaled containers remain inside the same server. Therefore, certain VNF unsatisfaction rates are visible as observed from Fig. 8 previously.

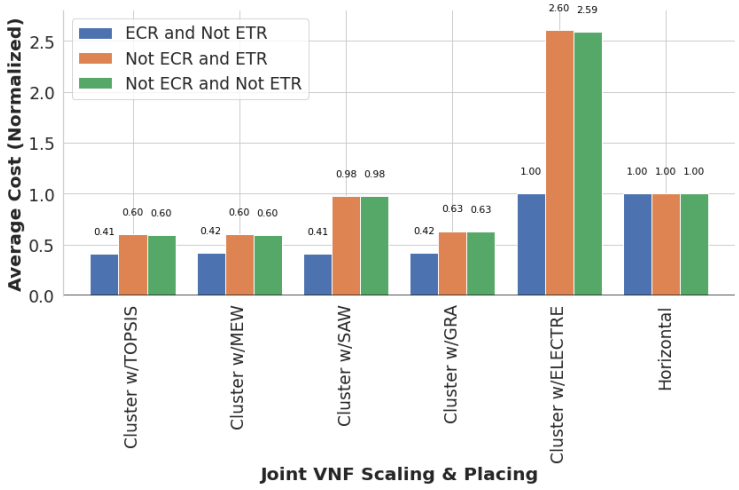



Fig. 9: Average cost of VNF placements using different MADM schemes after cluster auto-scaling normalized with respect to horizontal scaling in (ECR and Not ETR), (Not ETR and ECR) and (Not ETR and Not ECR) cases.

Note that in Fig. 9, TOPSIS algorithm yields the lowest cost of 0.41, 0.60 and 0.60 in (ECR and Not ETR), (Not ECR and ETR) and (Not ECR and Not ETR) cases respectively. This is followed by MEW and GRA algorithms where MEW performance is approximately equal to the performance of TOPSIS in (Not ECR and Not ETR) and (Not ECR and ETR) cases with slightly higher cost in (ECR and Not ETR) case. On the other hand, SAW and ELECTRE give higher average deployment

cost values. More specifically, SAW yields $0,41$, 0.98 and 0.98 average deployment cost values in (ECR and Not ETR), (Not ECR and ETR) and (Not ECR and Not ETR) cases respectively. At the same time, using ELECTRE has given the highest average cost values of 1.04, 2.60 and 2.90 in (ECR and Not ETR), (Not ECR and ETR) and (Not ECR and Not ETR) cases respectively. These values in ELECTRE algorithm are equal or worse than the horizontal scaling average deployment cost values. However, horizontal scaling has 3.52% VNF unsatisfaction ratio as given in Fig. 8. whereas all MADM algorithms that employ VNF cluster scaling and placement have satisfied all the VNF requests (i.e. has 0% VNF unsatisfaction ratio).

In summary, the above analysis results signify that an appropriate selection of the MADM algorithm is quite critical as not optimized algorithms (e.g. ELECTRE) can yield worse performance than simple horizontal scaling with no VNF placing schema. However, the results indicate the advantage of cluster scaling in terms of VNF satisfaction ratios (0% VNF unsatifaction ratio) compared to horizontal scaling (3.52% VNF unsatisfaction ratio) even in the case of higher deployment cost in all the considered cases (i.e. (ECR and Not ETR), (Not ECR and ETR) and (Not ECR and Not ETR)).

### 5.3 Discussion on Evaluation Results

Note that cluster auto-scaling has some advantages in terms of lower deployment cost and VNF request unsatisfaction ratios. Additional advantages include higher resilience capability to random hardware failures and outage due to existence of multiple servers, automatic upgrades in case one server goes down, hence resulting in low downtime. On the other hand, there are also some disadvantages of cluster auto-scaling in comparison to horizontal scaling. Cluster auto-scaling has the potential to create higher data inconsistency due to distributed and cross-server communication, has higher cost, more required code and can have synchronization challenges. Therefore, all these considerations should be taken into account when designing the scaling choice of operations according to observed number of VNF requests.

The results in Section 5.2 also indicated that the performance of more complex algorithms such as TOPSIS and GRA can be relatively similar to simple algorithms (such as MEW) that have lower complexities. However, the performance of simple algorithms are quite sensitive to variations in the dataset. For this reason, the best strategy to follow would be first to start investigating the performance of more simple algorithms such as MEW and SAW and then adjust the parameters according to the characteristics of the dataset. However, in case the dataset characteristic is variant, it is better to select more complex and robust algorithms (such as TOPSIS) that have the potential to give the lowest deployment cost and zero VNF un-satisfaction rate in all the considered scenarios.

From a computational complexity perspective, TOPSIS algorithm complexity is in the part of attribute normalization and weighting resulting in $\mathcal{O}(n^2)$ [45]. On the other hand, the computational overhead of MEW is shown to be about 7%, 0.8%, and 3% more than SAW, TOPSIS, and GRA, respectively in certain simulation scenarios in [30]. In summary, considering the computation overhead and performance comparisons, TOPSIS can be considered as one of the most suitable choice of MADM algorithm in the considered scaling and placement problem.

## 6 Conclusions

In this paper, we studied the VNF auto-scaling and placement problem inside clusters by considering transport and cloud related parameters during both scaling and placement decision making process. Our aim was to maximize the number of allocated VNF requests with minimum overall deployment cost, while considering bandwidth, latency and CPU costs by judiciously selecting the appropriate PoPs in a given cluster for each scaled version of VNFs. The original joint scaling and placement problem is known to be NP-hard and hence the problem is solved by separating scaling and placement problems and solving them individually. To evaluate the effectiveness of the proposed approach, we first emphasized the influence of scaling in both transport and cloud by comparing the VNF satisfaction rates. Then, to make experiments with different numbers of VNF requests, we used the digital twin dataset where addition

of robot instances and their corresponding cloud and transport parameters are utilized. The analysis results indicated that joint transport and cloud scaling along with TOPSIS algorithm yield the lowest deployment cost and VNF request unsatisfaction ratios compared to only transport or cloud scaling and other MADM algorithms.

## A Appendix

### A.1 TOPSIS

TOPSIS algorithm consists of easy implementation steps. After a decision matrix $\mathbf{A}_s = [a_{ij}]_{K\times M}$ is created in the first step for a given request-$s \in \mathcal{S}$, a normalized decision matrix $\mathbf{N}_s = [n_{ij}]_{K\times M}$ is formed in the second step (dropping the subscript $s$ here on) as:

$$n_{ij} = \frac{a_{ij}}{\sqrt{\sum_{k=1}^{K} a_{kj}^2}}. \tag{8}$$

In the third step, a weighted normalized decision matrix $\mathbf{V} = [v_{ij}]_{K\times M}$ is created by multiplying each column of the matrix $\mathbf{N}$ by a corresponding weight $w_i$ as:

$$\mathbf{v}_i = w_i \cdot \mathbf{n}_i \tag{9}$$

where

$$\mathbf{n}_i = [n_{1i}, ..., n_{pi}]^{\mathrm{T}}, \quad \mathbf{v}_i = [v_{1i}, ..., v_{pi}]^{\mathrm{T}}, \quad \text{for } i = 1, 2, .., M$$

In the fourth step, the positive $\mathrm{V}^+$ and negative $\mathrm{V}^-$ solution points are formed as:

$$\mathrm{V}^+ = \left\{v_1^+, v_2^+, ..., v_m^+\right\} = \left\{\max_i v_{ij} | j \in \{1, 2, ..., M\}\right\} \tag{10}$$

$$\mathrm{V}^- = \left\{v_1^-, v_2^-, ..., v_m^-\right\} = \left\{\min_i v_{ij} | j \in \{1, 2, ..., M\}\right\} \tag{11}$$

In the fifth step, the Euclidean distance $L_i^+$ of each multiple decision point from the positive point $\mathrm{V}^+$ and the Euclidean distance $L_i^-$ of each multiple decision point from the negative point $\mathrm{V}^+$ are calculated as:

$$L_i^+ = \sqrt{\sum_{j=1}^{p} \left(v_{ij} - v_j^+\right)^2}, \ i = 1, 2, .., K \tag{12}$$

$$L_i^- = \sqrt{\sum_{j=1}^{p} \left(v_{ij} - v_j^-\right)^2}, \ i = 1, 2, .., K. \tag{13}$$

In the next step, the relative similarity of the alternatives from the positive and negative point is calculated as:

$$T_i = \frac{L_i^-}{L_i^- + L_i^+},\ i = 1, 2, ..., K. \tag{14}$$

Then, the final solution $e^*$ (the best node selection for the placement of each containers that are scaled) is selected as:

$$e^* = e_{i^*} \text{ where } i^* = \arg\max_i T_i,\ i = 1, 2, ..., K. \tag{15}$$

## A.2 SAW

SAW algorithm is simple and is the most popular scoring method. In SAW, the score of each candidate node $i$ is obtained by adding the contributions from each attribute $a_{i,j}$ multiplied by the weight factors $w_j$. Then, the selected node is

$$e^* = e_{i^*} \text{ where } i^* = \arg\max_i \sum_j w_j a_{i,j}, \tag{16}$$

where $a_{i,j} = a_{i,j}/a_j^+$ for benefit parameters and $a_{i,j} = a_j^-/a_{i,j}$ for cost parameters and $a_j^+ = \max_i a_{i,j}$ and $a_j^- = \min_i a_{i,j}$.

## A.3 MEW

MEW or Weighted Product Model (WPM) is another scoring method where the node scores are determined based on weighted product of the attributes. The selected node is

$$e^* = e_{i^*} \text{ where } i^* = \arg\max_i \prod_j a_{i,j}^{w_j}. \tag{17}$$

## A.4 GRA

GRA algorithm is basically based on building grey relationships between elements of two series in order to compare each member quantitatively. One of the series consists of best-quality entities and the other series contains comparative series. If the difference between two series of the comparative series is low, then it is more preferable. A Grey relational coefficient (GRC) is defined to be used for describing the relationships between two series and is calculated based on the level of similarity and variability. GRA algorithm is generally implemented in six steps:

1. Apply classification of the series based on three conditions: larger-the-better, smaller-the-better and nominal-the-best. In this paper, larger-the-better is selected as the comparison condition. It is assumed that $D$ series $(\mathbf{A}^1, \mathbf{A}^2, \ldots, \mathbf{A}^D)$ are compared where each series $\mathbf{A}^i = [a_{i,1}, a_{i,2}, \ldots, a_{i,M}]$ has $M$ entities.

2. The upper, lower and moderate bounds of series elements are defined. The upper bound and lower bound are defined as $u_j = max\{a_{1,j}, a_{2,j}, \ldots, a_{D,j}\}$ and $l_j = min\{a_{1,j}, a_{2,j}, \ldots, a_{D,j}\}$ respectively where $j = 1, 2, \ldots, M$.
3. Normalize the individual entities: Before calculating GRC, the decision matrix $\mathbf{A}$ needs to be normalized. For normalization, following equation is used for larger-the-better condition.

$$a_{ij} = \frac{a_{ij} - l_j}{u_j - l_j}, \tag{18}$$

where $i = 1, 2, \ldots, D$.
4. Define the ideal series: A reference series $\mathbf{A}^0 = [a^*_{0,1}, a^*_{0,2}, \ldots, a^*_{0,M}] = [u_1, u_2, \ldots, u_M]$ is formed which corresponds to ideal solution.
5. Calculate the GRCs: The GRC can be calculated from

$$\Gamma_{0,i} = \frac{1}{M} \sum_{j=1}^{M} w_j \frac{\Delta_{min} + \Delta_{max}}{\Delta_i + \Delta_{max}}, \quad i = 1, 2, \ldots, D \tag{19}$$

where $\Delta_i = |a^*_{0,j} - a_{i,j}|$, $\Delta_{max} = \max_{i,j}(\Delta_i)$ and $\Delta_{min} = \max_{i,j}(\Delta_i)$. $\max_{i,j}()$ and $\max_{i,j}()$ are the functions of the maximum and minimum value of a set of numbers with varying i and j respectively.
6. Select the alternative that has the highest GRC

$$e^* = e_{i^*} \text{ where } i^* = \arg\max_i \Gamma_{0,i}. \tag{20}$$

## A.5 ELECTRE

ELECTRE method is based on outranking relation theory and is another method that analyzes data over the decision metric. When making a decision, the concordance and discordance indexes are used to measure the amount of dissatisfaction during the decision making process. After obtaining normalization decision matrix $\mathbf{N} = [n_{ij}]_{K \times M}$ as in (8) and weighted normalized decision matrix $\mathbf{V}$ as in (9), the concordance and discordance sets are applied. The set of criteria is divided into two different subsets. Denote $\mathcal{K} = \{k_1, k_2, k_3, \ldots, k_K\}$ a finite set of alternatives. In the following formulation, the data is divided into two different sets of concordance and discordance. If the alternative $k_1$ is preferred over alternative $k_2$ for all the criteria, then the concordance set is composed as follows:

$$C(k_1, k_2) = \{j | v_{k_1 j} > v_{k_2 j}\}, \quad (k_1, k_2 = 1, \ldots, M \text{ and } k_1 \neq k_2), \tag{21}$$

where $C(k_1, k_2)$ denotes the collection of attributes in which $k_1$ is better than, or equal, to $k_2$. Then the concordance index of $(k_1, k_2)$ is defined as:

$$C_{k_1,k_2} = \sum_{j^*} w_j, \tag{22}$$

where $j^*$ are the attributes contained in the concordance set $C(k_1, k_2)$. Similarly the discordance set is defined as:

$$D(k_1, k_2) = \{j | v_{k_1 j} < v_{k_2 j}\}, \quad (k_1, k_2 = 1, \ldots, M \text{ and } k_1 \neq k_2), \tag{23}$$

The discordance index $D_{k_1,k_2}$ represents the degree of disagreement in $k_1 - > k_2$ in the following way:

$$D_{k_1,k_2} = \frac{\sum_{j^+)} |v_{k_1 j^+} - v_{k_2 j^+}|}{\sum_{j)} |v_{k_1 j} - v_{k_2 j}|}, \tag{24}$$

where $j^+$ are the attributes contained in the discordance set $D(k_1, k_2)$ and $v_{ij}$ is the weighted normalized evaluation of the alternative i on criterion j. This method implies that $k_1$ outranks $k_2$ when $C_{k_1,k_2} \geq \hat{C}$ and, $D_{k_1,k_2} \leq \hat{D}$ where $\hat{C}$ is the averages of $C_{k_1,k_2}$ and $\hat{D}$ is the averages of $D_{k_1,k_2}$.

## Conflict of Interest Statement

On behalf of all authors, the corresponding author states that there is no conflict of interest.

## B Acknowledgements

This work was partially funded by EC H2020 5GPPP 5Growth project (Grant 856709), Spanish MINECO grant TEC2017-88373-R (5G-REFINE), Generalitat de Catalunya grant 2017 SGR 1195 and the national program on equipment and scientific and technical infrastructure, EQC2018-005257-P under the European Regional Development Fund (FEDER). We would also like to thank Milan Groshev, Carlos Guimarães for providing dataset for scaling of robot manipulator based digital twin service.

## References

1. Alessio Scalingi, Flavio Esposito, Waqar Muhammad, and Antonio Pescapé. Scalable provisioning of virtual network functions via supervised learning. In *2019 IEEE Conference on Network Softwarization (NetSoft)*, pages 423–431. IEEE, 2019.
2. Tommaso Cucinotta, Giacomo Lanciano, Antonio Ritacco, Fabio Brau, Filippo Galli, Vincenzo Iannino, Marco Vannucci, Antonino Artale, Joao Barata, and Enrica Sposato. Forecasting operation metrics for virtualized network functions. In *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, pages 596–605. IEEE, 2021.

3. Richard Fox and Wei Hao. *Internet infrastructure: networking, web services, and cloud computing*. CRC Press, 2017.
4. Omar Houidi, Oussama Soualah, Wajdi Louati, Marouen Mechtri, Djamal Zeghlache, and Farouk Kamoun. An efficient algorithm for virtual network function scaling. In *GLOBECOM 2017-2017 IEEE Global Communications Conference*, pages 1–7. IEEE, 2017.
5. Emiliano Casalicchio. A study on performance measures for auto-scaling cpu-intensive containerized applications. *Cluster Computing*, 22(3):995–1006, 2019.
6. Fabiana Rossi, Matteo Nardelli, and Valeria Cardellini. Horizontal and vertical scaling of container-based applications using reinforcement learning. In *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, pages 329–338. IEEE, 2019.
7. Eugene Kim. Amazon prime day crash: internal docs reveal scramble to fix, July 2018.
8. Abeer Abdel Khaleq and Ilkyeun Ra. Agnostic approach for microservices autoscaling in cloud applications. In *2019 International Conference on Computational Science and Computational Intelligence (CSCI)*, pages 1411–1415. IEEE, 2019.
9. Thanh-Tung Nguyen, Yu-Jin Yeom, Taehong Kim, Dae-Heon Park, and Sehan Kim. Horizontal pod autoscaling in kubernetes for elastic container orchestration. *Sensors*, 20(16):4621, 2020.
10. Fabiana Rossi, Valeria Cardellini, and Francesco Lo Presti. Hierarchical scaling of microservices in kubernetes. In *2020 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS)*, pages 28–37. IEEE, 2020.
11. László Toka, Gergely Dobreff, Balázs Fodor, and Balázs Sonkoly. Machine learning-based scaling management for kubernetes edge clusters. *IEEE Transactions on Network and Service Management*, 18(1):958–972, 2021.
12. Francisco Carpio, Samia Dhahri, and Admela Jukan. Vnf placement with replication for loac balancing in nfv networks. In *2017 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2017.
13. Defang Li, Peilin Hong, Kaiping Xue, and Jianing Pei. Virtual network function placement and resource optimization in nfv and edge computing enabled networks. *Computer Networks*, 152:12–24, 2019.
14. Samaneh Sadegh, Kamran Zamanifar, Piotr Kasprzak, and Ramin Yahyapour. A two-phase virtual machine placement policy for data-intensive applications in cloud. *Journal of Network and Computer Applications*, 180:103025, 2021.
15. Sadoon Azizi, Mohammad Shojafar, Jemal Abawajy, and Rajkumar Buyya. Grvmp: A greedy randomized algorithm for virtual machine placement in cloud data centers. *IEEE Systems Journal*, 2020.
16. Bin Gao, Zhi Zhou, Fangming Liu, and Fei Xu. Winning at the starting line: Joint network selection and service placement for mobile edge computing. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pages 1459–1467. IEEE, 2019.

17. Md Hasanul Ferdaus, Manzur Murshed, Rodrigo N Calheiros, and Rajkumar Buyya. An algorithm for network and data-aware placement of multi-tier applications in cloud data centers. *Journal of Network and Computer Applications*, 98:65–83, 2017.
18. Peng Qin, Bin Dai, Benxiong Huang, and Guan Xu. Bandwidth-aware scheduling with sdn in hadoop: A new trend for big data. *IEEE Systems Journal*, 11(4):2337–2344, 2015.
19. Xiangqiang Gao, Rongke Liu, and Aryan Kaushik. Virtual network function placement in satellite edge computing with a potential game approach. *arXiv preprint arXiv:2012.00941*, 2020.
20. Abdelquoddouss Laghrissi and Tarik Taleb. A survey on the placement of virtual resources and virtual network functions. *IEEE Communications Surveys & Tutorials*, 21(2):1409–1434, 2018.
21. Manoel C Silva Filho, Claudio C Monteiro, Pedro RM Inácio, and Mário M Freire. Approaches for optimizing virtual machine placement and migration in cloud environments: A survey. *Journal of Parallel and Distributed Computing*, 111:222–250, 2018.
22. Jungmin Son and Rajkumar Buyya. Priority-aware vm allocation and network bandwidth provisioning in software-defined networking (sdn)-enabled clouds. *IEEE Transactions on Sustainable Computing*, 4(1):17–28, 2018.
23. Sayyid Shahab Nabavi, Sukhpal Singh Gill, Minxian Xu, Mohammad Masdari, and Peter Garraghan. Tractor: Traffic-aware and power-efficient virtual machine placement in edge-cloud data centers using artificial bee colony optimization. *International Journal of Communication Systems*, page e4747, 2021.
24. Tejas Subramanya and Roberto Riggio. Machine learning-driven scaling and placement of virtual network functions at the network edges. In *2019 IEEE Conference on Network Softwarization (NetSoft)*, pages 414–422. IEEE, 2019.
25. Aris Leivadeas, George Kesidis, Mohamed Ibnkahla, and Ioannis Lambadaris. Vnf placement optimization at the edge and cloud. *Future Internet*, 11(3):69, 2019.
26. Thomas Lin and Alberto Leon-Garcia. Towards a client-centric qos auto-scaling system. In *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium*, pages 1–9. IEEE, 2020.
27. Michael Scharf, Manuel Stein, Thomas Voith, and Volker Hilt. Network-aware instance scheduling in openstack. In *2015 24th International Conference on Computer Communication and Networks (ICCCN)*, pages 1–6. IEEE, 2015.
28. Mehdi Hosseinzadeh, Hawkar Kamaran Hama, Marwan Yassin Ghafour, Mohammad Masdari, Omed Hassan Ahmed, and Hemn Khezri. Service selection using multi-criteria decision making: a comprehensive overview. *Journal of Network and Systems Management*, 28(4):1639–1693, 2020.
29. A Serdar Tan and Engin Zeydan. Performance maximization of network assisted mobile data offloading with opportunistic device-to-device communications. *Computer Networks*, 141:31–43, 2018.

30. Jian Zhou and Can-yan Zhu. Compensatory analysis and optimization for madm for heterogeneous wireless network selection. *Journal of Electrical and Computer Engineering*, 2016, 2016.
31. He-Wei Yu and Biao Zhang. A hybrid madm algorithm based on attribute weight and utility value for heterogeneous network selection. *Journal of network and systems management*, 27(3):756–783, 2019.
32. J Baranda, J Mangues-Bafalluy, Engin Zeydan, L Vettori, Ricardo Martínez, Xi Li, Andres Garcia-Saavedra, CF Chiasserini, C Casetti, K Tomakh, et al. On the integration of ai/ml-based scaling operations in the 5growth platform. In *2020 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pages 105–109. IEEE, 2020.
33. Xi Li, Andres Garcia-Saavedra, Xavier Costa-Perez, Carlos J Bernardos, Carlos Guimarães, Kiril Antevski, Josep Mangues-Bafalluy, Jorge Baranda, Engin Zeydan, Daniel Corujo, et al. 5growth: An end-to-end service platform for automated deployment and management of vertical services over 5g networks. *IEEE Communications Magazine*, 59(3):84–90, 2021.
34. Paola Cappanera, Federica Paganelli, and Francesca Paradiso. Vnf placement for service chaining in a distributed cloud environment with multiple stakeholders. *Computer Communications*, 133:24–40, 2019.
35. Sevil Mehraghdam, Matthias Keller, and Holger Karl. Specifying and placing chains of virtual network functions. In *2014 IEEE 3rd International Conference on Cloud Networking (CloudNet)*, pages 7–13. IEEE, 2014.
36. Milan Groshev, Carlos Guimarães, Jorge Martín-Pérez, and Antonio de la Oliva. Toward intelligent cyber-physical systems: Digital twin meets artificial intelligence. *IEEE Communications Magazine*, 59(8):14–20, 2021.
37. Farooq Bari and Victor Leung. Multi-attribute network selection by iterative topsis for heterogeneous wireless access. In *2007 4th IEEE consumer communications and networking conference*, pages 808–812. IEEE, 2007.
38. Ching-Lai Hwang, Young-Jou Lai, and Ting-Yun Liu. A new approach for multiple objective decision making. *Computers & operations research*, 20(8):889–899, 1993.
39. Jose D Martinez-Morales, Ulises Pineda-Rico, and Enrique Stevens-Navarro. Performance comparison between madm algorithms for vertical handoff in 4g networks. In *2010 7th International Conference on Electrical Engineering Computing Science and Automatic Control*, pages 309–314. IEEE, 2010.
40. Qingyang Song and Abbas Jamalipour. A network selection mechanism for next generation networks. In *IEEE International Conference on Communications, 2005. ICC 2005. 2005*, volume 2, pages 1418–1422. IEEE, 2005.
41. Babak Daneshvar Rouyendegh and Serpil Erol. Selecting the best project using the fuzzy electre method. *Mathematical Problems in Engineering*, 2012, 2012.

42. K Paul Yoon and Ching-Lai Hwang. *Multiple attribute decision making: an introduction*. Sage publications, 1995.
43. J Baranda, J Mangues-Bafalluy, E Zeydan, C Casetti, CF Chiasserini, M Malinverno, C Puligheddu, M Groshev, C Guimarães, K Tomakh, et al. Aiml-as-a-service for sla management of a digital twin virtual network service. In *IEEE INFOCOM 2021-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 1–2. IEEE, 2021.
44. Chrysa Papagianni, Josep Mangues-Bafalluy, Pedro Bermudez, Sokratis Barmpounakis, Danny De Vleeschauwer, Juan Brenes, Engin Zeydan, Claudio Casetti, Carlos Guimarães, Pablo Murillo, et al. 5growth: Ai-driven 5g for automation in vertical industries. In *2020 European Conference on Networks and Communications (EuCNC)*, pages 17–22. IEEE, 2020.
45. Hamdani and Retantyo Wardoyo. The complexity calculation for group decision making using topsis algorithm. In *AIP conference proceedings*, volume 1755, page 070007. AIP Publishing LLC, 2016.