Network Intelligence for NFV scaling in closed-loop architectures

Paola Soto, Miguel Camelo, Danny De Vleeschauwer, Yorick De Bock, Chia-Yu Chang, Juan F. Botero, and Steven Latré

***Abstract*—Characteristics like self-managing, self-adaptation, and self-organization are the main objectives of intelligent network operation. Artificial Intelligence (AI) and Machine Learning (ML) algorithms will enable future networks to operate entirely autonomously. However, current network architectures are not fully prepared to include and properly handle the promised Network Intelligence (NI). This article looks at scaling, one key Management and Orchestration (MANO) operation that allows the network to adapt to unexpected changes. We show how different scaling methods can fit the Monitor-Analyze-Plan-Execute over a shared Knowledge (MAPE-K), a well-established framework that allows architectural-based adaptation. Using a cloud-based scenario, we compare and highlight architectural differences between two prominent scaling methods, one based on Reinforcement Learning (RL) and the other based on classical control theory, showing that only the data-driven approach is adaptable enough to achieve automation. We conclude the article by pointing toward future research in autonomous, adaptive networks.**

***Index Terms*—Management and Orchestration, Network Automation, NFV scaling, Reinforcement Learning, Self-Adaptive Systems, Zero-Touch Networks.**

I. Introduction

MOBILE networks constantly evolve due to society's desire for connectivity, capacity, and newer services. The fifth (5G) and newer generations of mobile networks rely on softwarization and cloudification trends, enabled by technologies like Software Defined Networking (SDN) and Network Function Virtualization (NFV). Therefore, the network is becoming a general-purpose platform that provides connectivity to many heterogeneous devices and guarantees Quality of Service (QoS). However, Network Service Providers (NSPs) typically manage most network challenges using simple rules, models, and heuristics. Such approaches still face limitations when handling new contexts or reacting to new demands, given their static nature. Therefore, SDN- and NFV-based networks need to evolve to address this open issue and fulfill the vision of zero-touch networks, where the network autonomously adapts to changing conditions [1].

AI, specifically ML, is considered the keystone in this digital transformation [2], after successfully solving critical problems like network service management and resource provisioning [3]. Additionally, a series of collaborative AI/ML algorithms —or Network Intelligent Functions (NIFs) — will build the so-called NI by swiftly detecting/anticipating network changes and then smoothly reacting without human intervention [4]. Unfortunately, existing management frameworks often assume already trained AI/ML models operate independently and do not fully support their lifecycle management (i.e., each model is managed individually). Therefore, it is challenging to ensure End-to-End (E2E) performance alignment, algorithm convergence, and global optimality of the zero-touch network management process when including NI during network operation.

Currently, there are some efforts from European Telecommunications Standards Institute (ETSI) to include AI/ML algorithms and their management in network architectures [1], [5]; however, these efforts are still in their infancy. Following the nature of Zero-touch network and Service Management (ZSM), the EU-funded DAEMON project considers an NI-native architecture that unifies the NI representation, promotes the reuse of NIFs, and is fully aligned with emerging designs in standardization [4]. The key element in DAEMON's architecture is the inclusion of multiple closed control loops along different network micro-domains, breaking the centralized, human-in-the-loop solutions, which are identified to be slow and error-prone.

In the context of 5G and 6G networks, where MANO will be based on NI, this article focuses on scaling, a fundamental MANO operation in NFV that allows NSPs to automatically resize Network Services (NSs) at runtime to meet traffic surges while providing performance guarantees. Furthermore, scaling methods are mainly designed as closed-loop control, so they can be mapped to a general architecture-based framework called MAPE-K, one of the most influential reference control models for autonomous and self-adaptive systems [6], for clearer understanding and detailed comparison. Nevertheless, this framework presents some limitations when contemplating AI/ML-based solutions. The contributions of this article are threefold:

- We establish a methodology to describe several NS scaling methods (e.g., ML-, rule-, and control theory-based) based on an extended MAPE-K framework. To the authors' best knowledge, this is the first article that shows how NIFs for NFV scaling can be generically defined, allowing re-utilization and evolution.
- We design an RL-based algorithm to solve the scaling problem where a multi-objective function is optimized regarding the number of replicas and a target delay. We frame this solution into the NI, going beyond state-of-the-art [7], where the scaling is merely designed following a single optimization criterion.
- Using a cloud-based scenario, we compare two scaling

P. Soto is with University of Antwerp - imec, IDLab, and Universidad de Antioquia. M. Camelo, Y. De Bock, and S. Latré are with University of Antwerp - imec, IDLab. D. De Vleeschauwer and C. Y. Chang are with with Nokia Bell Labs. J.F. Botero is with Universidad de Antioquia.

methods (RL- and control-based). Specifically, we compare them quantitatively and show that, despite several approaches fitting the methodology, only data-driven algorithms produce the desired adaptation and automation in network operations.

This article is organized as follows. In Section II, we describe the scaling problem and list the approaches used to solve it. Then, in Section III, we explain the methodology we follow to represent different NIFs in a unified way and use it to decompose the approaches listed in Section II. In Section IV, we present our RL-based scaler and show the comparison results against a control-theory-based scaler. Finally, we present some open challenges and opportunities and conclude the article in Section V.

II. Background and Related Work

In NFV, Virtual Network Functions (VNFs) are software implementations of network functionalities previously provided as network equipment/devices. Being software components, VNFs can be scaled and moved on demand over general-purpose infrastructure. A composition of multiple VNFs is called an NS, e.g., remote secured access composed of a firewall and a VPN gateway. To meet Service Level Agreements (SLAs), NSPs resize NSs or individual VNFs, i.e., change the number of the replicas of a given NS (horizontal) or change the computing resources assigned to them (vertical), to fulfill operational and business objectives in a multi-objective setup [7]. First, the maximum tolerable delay for a given NS, and second, the infrastructure cost of implementing the necessary software pieces (i.e., VNF replicas).

Simultaneously, scaling is considered a decision-making problem as it determines the number of replicas that achieves both objectives. Decision-making problems are frequently modeled as Markov Decision Processes (MDPs). An MDP is a discrete-time stochastic framework that finds an optimal policy or strategy that maximizes the expected long-term reward, a discounted sum of immediate rewards. The optimal policy is found after many interactions with the environment until the scaler has converged. Several methods have been explored, regardless of how the scaling problem is modeled.

Traditional approaches for scaling are reactive since they manually and statically set the number of replicas, frequently by over-provisioning. Other reactive approaches include threshold-based methods [8], in which the scaler is told what to do if the network Key Performance Indicators (KPIs) are above or below a given threshold, using predefined rules. This threshold represents an optimal operation point, which is derived from expert knowledge. Such approaches only work in small configurations and are difficult to maintain, as either they are based on open-loop architectures or closed-loop but slow (e.g., the expert, as human-in-the-loop, needs to redetermine the threshold).

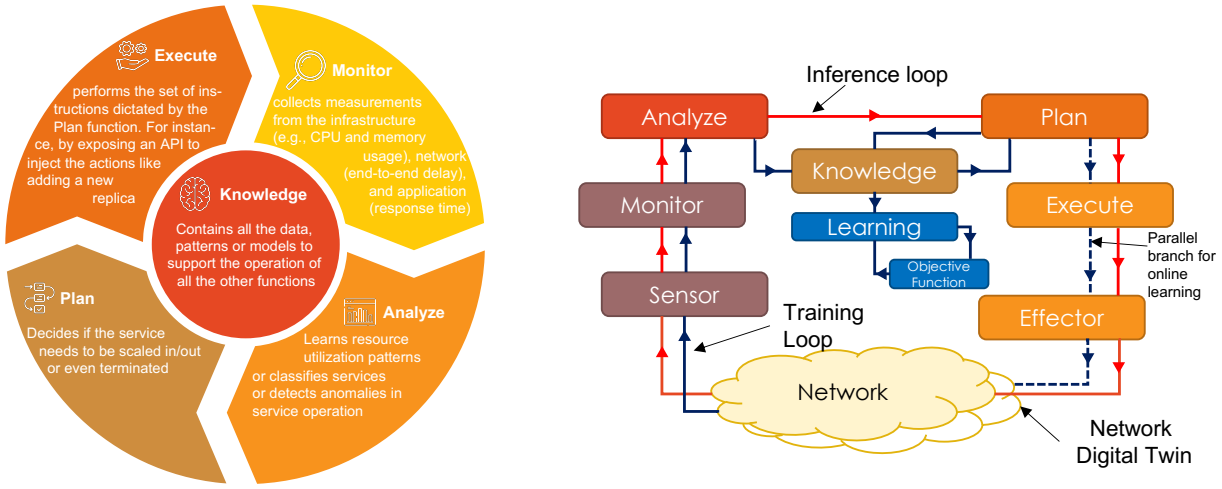
Control theory can also be used for scaling [9]. Here, the controller regulates a control variable, e.g., the number of resources, to keep a (measured) variable controlled, e.g., the KPIs specified in the SLA, as close as possible to a target value within some allowed boundaries. For instance, a Proportional–Integral (PI) controller calculates an error, which represents the deviation between the target and the measured value, and a trend, which captures how the measured value evolves. Since the objective is to keep the measured variable close to the target, the correction signal, i.e., the change that needs to be applied to the control variable, is a weighted sum of the error and the trend. There is also a third term for some controllers, but we do not consider it here. One of the advantages of using PI controllers for scaling is their steady response to small changes in the input variable (i.e., incoming traffic load). Moreover, if the system to be controlled is linear, the controller's parameters can be optimally set using Fourier or Laplace domain analyses. However, for non-linear systems, tuning the controller's parameters can be lengthy and computationally expensive.

Data-driven approaches have also been proposed as scaling techniques. Data-driven approaches employ methods to extract meaningful features from data, which can later be leveraged in decision-making, which emulates human intelligence. ML algorithms belong to this kind of approaches. Predictive scaling is the main area where ML is applied [10]. In predictive scaling, modern Supervised Learning (SL) techniques such as time-series forecasting are applied to find hidden patterns in the traffic load. The discovered patterns are then used to predict the expected load, which could be translated into scaling decisions (e.g., by applying a step activation function) so that MANO platforms can anticipate and set up the resources needed for future demand. More complex SL techniques, such as Convolutional Neural Networks (CNNs), are known for their automatic feature extraction (e.g., patterns) and function approximation (e.g., mapping to scaling decisions). To achieve this, they require enough labeled data to learn to identify such patterns and adapt accordingly. Data labeling is often a lengthy and manual process that, in most cases, also requires expert knowledge.

Given its decision-making nature, RL [11] is also explored as a scaling solution. In RL, an agent explores and analyzes the effects of different actions in an environment. This process can be seen as trial and error, where after many attempts, the agent learns a strategy that allows it to choose better actions over time. The main element in this learning process is the reward function, which tells the agent if the action it took was appropriate. This function guides the agent towards choosing the actions that maximize an expected reward, equivalent to achieving the learning goal. Unlike threshold-based or rule-based solutions, where a scaler is directly instructed on what to do, an RL-based scaler proactively adapts the NS instances according to a learned strategy. This behavior is similar to that of predictive scalers since, in a given state, the RL agent can anticipate future changes. However, RL does not require *a priori* knowledge of the system or expert knowledge to define the optimal operation points, making it more adaptable than static solutions.

III. Closed-loop control for NI-based scaling

MAPE-K is an architectural framework for autonomic and adaptive systems consisting of a closed loop of four phases.



(a) General MAPE-K

(b) Extended MAPE-K

Fig. 1. The (a) general and (b) extended MAPE-K framework for NFV scaling, where the main differences are the learning function (in blue), a training and an inference loop which inject their outcomes to the network or its digital twin, depending on which loop is considered.

During the first phase, the *Monitor* function collects data from the resources through sensors distributed over the system. In the second phase, the *Analyze* function correlates the received information and creates a system model, which could be used to predict future situations. Then, the *Plan* function decides the actions needed to achieve the system's objectives. Finally, the *Execute* function performs the planned actions over the resources. The *Knowledge* is represented by the data shared among the previous functions.

As explained before, scaling is mainly designed following a closed-loop control, similar to that proposed by MAPE-K. Thus, the scaling solutions can be naturally elaborated using the MAPE-K framework, as depicted in Figure 1a, where we show, in a general way, what every MAPE-K function expects of a scaling solution. The MAPE-K can represent the interactions between different components in a unified way, allowing their coexistence in the same network infrastructure.

Nonetheless, MAPE-K presents some limitations when contemplating AI/ML-based solutions, especially in determining if the NI is being trained or used in inference and distinguishing between different ML types (e.g., SL and RL). Therefore, we extend the definition of the MAPE-K, as shown in Figure 1b, to include such specificities coming from the integration of ML and networked systems. We introduce a learning block that optimizes an objective function. Notice that the objective function optimizes the algorithm's learning (e.g., minimizing the cross-entropy) according to the learning problem (e.g., classification), and it is not necessarily related to the optimal solution of the networking problem (e.g., selecting the best modulation scheme to minimize interference). Moreover, depending on which loop is considered, we include different training and inference loops, which inject their outcomes into the network or its digital twin. A digital twin network is a virtual representation of a physical network. Thanks to their access to real-time data, digital twins can accurately model complex systems. This way, multiple decisions can be tested safely without impacting the production network [12].

Regarding this last point, ML algorithms typically have two operation modes: training and testing/inference. Algorithms that have been tested on a digital twin can be readily introduced in production networks. In contrast, the outcomes in training mode cannot be applied directly. Since current network architectures do not naturally integrate these solutions, they are typically trained in non-production networks (e.g., simulators, emulators, and testbeds) and deployed in isolation (e.g., without interacting with other ML algorithms). Consequently, the gap between reality and simulation in the networking domain is big and difficult to close.

In Table I, we decompose the most prominent scaling solutions using this extended version of the MAPE-K. All algorithms monitor similar variables and execute their decisions similarly by exposing an Application Programming Interface (API) to communicate with MANO platforms. The *Monitor* and the *Execute* functions represent the algorithms' inputs and outputs and are common to all scaling methods. However, depending on how the infrastructure exposes the monitoring information and the control parameters, the information these two functions process may vary.

The main differences emerge in the *Analyze* and *Plan* functions, where intelligent and non-intelligent methods differ. For SL-based scalers, the intelligence is in the *Analyze* function, where they identify the common patterns between the incoming traffic load and the scaling decisions. On the contrary, the intelligence of RL-based scalers is in the *Plan* function, where they translate network states into scaling decisions, freely deciding when to increase or decrease the number of NS instances. Notice that a combination of algorithms is also possible by using the same MAPE-K representation. For example, a scaling solution may combine an RL method to estimate the derivative term of a PI controller, which determines the number of replicas needed to meet the SLA [13].

Another difference among the scaling methods is how the *Knowledge* is acquired. In non-intelligent methods, the knowledge is extracted from the network and resides in behavioral rules (e.g., increasing the replicas by two if the threshold is surpassed), typically designed by NSPs, which are external

TABLE I
MAPE-K DECOMPOSITION OF DIFFERENT SCALING METHODS

MAPE-K Component	Scaling Method			
	Threshold-based	**Control-based**	**SL-based**	**RL-based**
Monitor	Resource State: CPU utilization Network State: service latency, E2E delay	Network State: service latency, E2E delay	Traffic Demands: Traffic load Resource State: CPU utilization, number of replicas Network State: service latency, E2E delay	Resource State: CPU utilization, number of replicas Network State: service latency, E2E delay
Analyze	Comparison between the monitored variables and the predefined thresholds	Computation of how the control variable needs to be changed as a weighted sum of an error term and a trend	The monitored variables are passed by a time series forecasting algorithm to learn hidden patterns	The monitored variables are averaged to compose the State.
Plan	Predefined actions according to the thresholds		Apply a mathematical formula to translate future patterns into scaling decisions	An agent takes the best action according to the learned strategy and current network state
Execute	An API to the MANO platform to communicate scaling decisions			
Knowledge	External: Human knowledge to define the threshold	Control terms' values	Model of the expected traffic load	Strategy with the actions to be taken according to the network state
Training Loss State/Actions/Rewards	N/A		Cross-Entropy	States: Avg CPU utilization, Avg latency Rewards: Resource utilization tolerance

agents. In contrast, in NIFs, the knowledge can be automatically incorporated into the network by including the learning function explained above. However, the way the scalers obtain this knowledge is entirely different. While intelligent scalers obtain their knowledge by training, the control-based tune the control parameters by running brute-force heuristics.

IV. USE CASE: NI FOR NFV-SCALING USING CLOSED-CONTROL LOOPS

From the scalers presented in the previous section, only two of them inherently include closed-control loops in their internal working: the RL- and the control-based. We assume that the scalers are deployed in the control plane of edge-cloud domains so they can control NSs replicas in such domains. This section shows the internal design of these two algorithms for NS scaling and compares them.

A. Scalers

1) ***RL-based scaler****:* The scaler interacts with the network in discrete time steps. It observes the network state at every step, composed of the tuple {*number of VNF replicas*, *average Central Processing Unit (CPU) utilization*, *perceived peak latency*}. Then, based on these observations, the agent decides to increase (by one), decrease (by one), or maintain the number of replicas. The executed action changes the network from state. The objective of this RL agent is to fulfill the agreed SLA with the minimum amount of replicas. Therefore, in every time step, the agent pays an immediate cost depending on how good or bad the action it took is. The cost of taking action when the environment moves from one state to another can be defined as a weighted function, including the following contributions.

- If the agent cannot fulfill the SLA, it incurs a performance cost, with an associated w_{perf}, which is paid every time the perceived peak latency exceeds a predefined threshold. The cost is zero otherwise.
- If the agent must deploy a new replica, a resource cost is paid, with an associated w_{res}; this can be seen as a rental cost in cloud environments or the consumed energy of the replica while it is running.

These two contributions are combined into a weighted function, where the respective non-negative weights define an optimization profile. The weights (w_{perf} and w_{res}) multiply an indicator function that varies between 1 and -1 depending on whether or not a condition is met. For instance, if the perceived peak latency is above a threshold, the indicator function is 1 or 0 otherwise; if a new replica is instantiated, the indicator function is 1, or -1 if the replica is removed. Finally, the reward function is defined as the negative cost function.

2) ***Control-based scaler****:* This scaler is inspired by a traditional PI controller. At the end of every time step, the scaler calculates a parameter ϕ by adding two terms: (1) a term proportional to how much the current latency deviates from a target θ and (2) a term proportional to the trend in latency (i.e., the latency difference in the current and previous step). The proportional coefficients for the first and second terms are α and β, respectively. At the beginning of the next step, the scaler changes the number of replicas depending on the calculated ϕ. If ϕ is more than one unit larger than the current number of replicas, a new replica is added. Otherwise, a replica is removed if ϕ is more than one unit smaller than the current number of replicas.

For training the PI scaler, we use the same reward function as above. For each combination of the values for the parameters:

$$\alpha = 0.015625, 0.0625, 0.25, 1, 4, 16$$
$$\beta = 50, 100, 200, 400$$
$$\theta = 10ms, 15ms, 20ms$$

we calculate the (time-varying) value, defined as the accumulated, discounted reward (with discount factor $\gamma = 0.9$), as we run through the training trace, thus obtaining $6 \times 4 \times 3$ value traces. Of those value traces, we take the step-per-step minimum. The parameter combination (θ, α, β) associated with the value trace that deviates the least from that minimum is selected as the best combination.

B. Simulation scenario

To evaluate the performance of the scalers, we simulated a simple use case based on a cloud system. The incoming

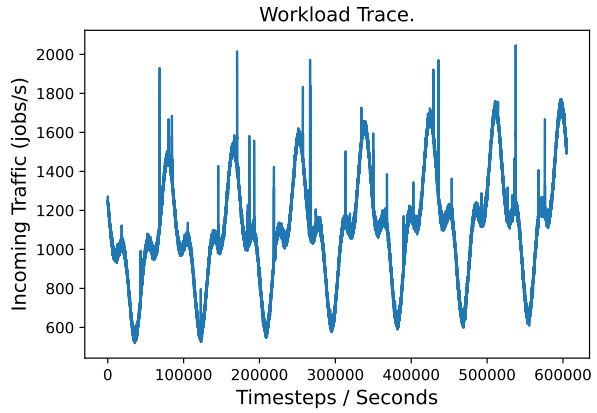


Fig. 2. Complete workload trace

traffic load enters a load balancer which distributes it among the VNF replicas. The number of replicas must be dynamically and efficiently changed to meet a given SLA without incurring in under- or over-provisioning. To be as general as possible, we assume horizontal scaling, where the processing capacity is increased/decreased by instantiating/removing a replica. Horizontal scaling can be exploited by distributed services where the workload is shared among different instances of the same service, leveraging the virtually infinite computing capacity of the cloud. We consider a monitoring block that constantly reports usage metrics to a decision-making agent. The agent automatically determines the number of replicas based on the received information. This use case contains all the functions proposed by MAPE-K, explained in Section III.

The proposed scenario was simulated using Simulation of Discrete Systems of All Scales (Sim-Diasca), a general-purpose, parallel, and distributed discrete-time simulation engine for complex systems written in the Erlang language [14]. Using ZeroMQ, we communicate Sim-Diasca with high-level programming languages like Python. The scalers proposed in the previous section were implemented in Python and interact with the environment (i.e., the simulator in Sim-Diasca) on a time-step basis. In a time step, Sim-Diasca simulates all the defined functionalities and waits for the scaler's decision. Subsequently, the time manager increases the time step by one, and the simulation goes to the next step. We also defined an initial scenario composed of a server with two NSs and a load balancer between them. This initial scenario is deployed every time the simulation is restarted.

We considered as incoming traffic load a datacenter-like pattern, where the traffic is low during off-work hours and high during working hours. However, as seen in Figure 2, we introduce some randomness to this pattern to test the ability of the scalers to react to sudden changes. Moreover, we decided that a time step represents one second, which means that a point in the workload trace represents the incoming traffic during a second. Thus, the trace represents a week's workload for a given service. For processing this workload, a VNF can process up to 300 jobs each time step, representing a CPU's capacity. Similarly, the server can host a maximum of 40 NSs.

Regarding the RL-based scaler, we implemented a Proximal Policy Optimization (PPO) agent using Stable-Baselines3 (SB3) default values. SB3 is a framework that implements popular RL algorithms for benchmarking purposes. PPO is a model-free algorithm that does not require knowledge of the system dynamics. Moreover, PPO is an on-policy algorithm which means that the algorithm improves the strategy that is used to make decisions, resulting in more stable learning during training [15]. PPO is also easier to implement and tune than other RL algorithms. We trained the PPO scaler using one day ($86\,400$-time steps), two days, and three days of the workload trace. We evidenced during some exploratory experiments that the agent needed at least $172\,800$ samples to learn a strategy resulting in good behavior during testing. Consequently, the results reported in this section are of a PPO scaler trained during the first two days of the trace.

Regarding the PI-based scaler, we used the method described in Section IV-A2 to determine the optimal PI parameters during the first two days of the trace. It turns out that for the considered sets of weights, the same PI parameters yield the lowest cost, i.e., (θ, α, β) = $(10\text{ms},\ 0.0625,\ 400)$.

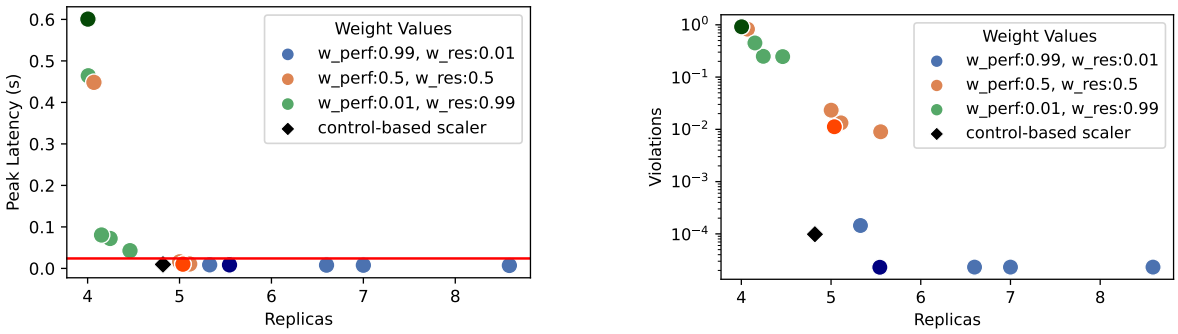
C. Simulation results

To show the agents' ability to react to unseen workload traces, they were tested using the last two days of the trace (i.e., the last 172.8K workload values). We evaluated different cost function weights ((w_{perf}, w_{res}) = $(0.99,\ 0.01)$, $(0.5,\ 0.5)$, and $(0.01,\ 0.99)$) to show how the scalers can adapt their behavior based on what they are focused on optimizing. For instance, by optimizing performance over resources, the scaler will more likely pay attention to fulfilling the SLA while disregarding the number of replicas. Since the RL behavior during training highly depends on the initial weights of the neural network, we trained the PPO scaler several times with different seeds. In each training, the same weight configuration is used. Notice that, despite training and testing the RL in separate processes, RL can still learn during the testing phase, similar to an online learning method. On the contrary, once the PI parameter values are set, its behavior is deterministic. Moreover, for the reward function defined here, the best set of PI parameters is insensitive to the weights.

The results of the evaluation using the testing trace are shown in Figure 3. Figure 3a shows the behavior of the proposed scalers in terms of the peak latency where the red line identifies the SLA. In this case, the SLA specifies a service's maximum tolerated peak latency (24ms). Similarly, Figure 3b shows how many violations are made by the scalers.

As it can be seen, when the PPO is trained with a reward function that optimizes the resources over the performance (i.e., green dots), the average amount of replicas is always lower than five; however, there is no guarantee of the achievement of the SLA. For instance, the darker green dot in Figure 3 creates, on average, 4.003 replicas, obtaining an average peak latency of 600.75ms, which translates into violating the SLA about 91.7% of the time.

On the contrary, when the scaler is trained with a reward function that optimizes the performance over the resources (i.e., blue dots), the violations are reduced to their minimum



(a) Peak Latency vs Replicas trade-off

(b) SLA violations vs Replicas trade-off

Fig. 3. Testing results showing the trade-off between (a) peak latency, and (b) fraction of violations and number of replicas.

at the expense of creating more replicas. The scaler on the darker blue dot creates an average of 5.544 replicas, obtaining an average of 8.5ms of peak latency, minimizing the violation of the SLA to 0.0023% of the time.

It also can be observed how the PPO tries to find a balance between both objectives when the weights are equal (i.e., orange dots). This clearly indicates that the PPO performs plausibly, optimizing the objective it should optimize and trying to balance both objectives when indicated. On average, the solution in dark orange creates 5.04 replicas, having an average of 10.24ms of peak latency and violating the SLA just 1.1% of the time.

Nevertheless, without having an optimal strategy, which indicates the optimal number of replicas achieving the SLA, there is no simple and fast comparison between the designed strategies. In that sense, we use the control-based scaler as a reference point to the RL-based scaler since it provides a solution that shows a good trade-off between both objectives (i.e., minimizing the number of replicas while fulfilling the SLA). The solution shown as a black diamond in Figure 3 creates, on average, 4.82 replicas, having an average of 9.62ms of peak latency and violating the SLA 0.0098% of the time.

D. Discussion

In the previous sections, we have shown how two approaches for scaling can be mapped in a common framework to support self-adaptation and how they behave when evaluated using the same environment. Notice that, despite this fact, they are intrinsically not comparable since they have different requirements for learning.

On the one hand, the control-based approach is deterministic, which implies that its output will likely be the same if the tuned control parameters are the same. Nevertheless, the tuning process is long and computing expensive since it requires brute force to determine the appropriate values for the control parameters.

On the other hand, the RL-based approach is non-deterministic, meaning that different outputs can be obtained for the same parameter configuration due to the random initialization of the neural network's weights used in the state-action approximation. However, as shown, RL algorithms can change or learn the model during runtime. This is especially powerful since the algorithm does not require complete re-training to learn new patterns when the workload changes. Moreover, the reward function guides learning by determining the actions that reach the (multi-)objective function. In each state, the RL-based scaler can find the action that suits that state; thus, we are not instructing the scaler what action to take; instead, the scaler takes that decision autonomously, following a learned strategy.

The previous point also relates to the inability of threshold-based algorithms to adapt since the operation thresholds must be determined using expert knowledge, which cannot be introduced during runtime without tearing down the service. Therefore, the operational thresholds are valid for a given network configuration and traffic load. Once they change, the thresholds must be calculated again. Moreover, the operational thresholds might be challenging to define in more complex cases, like the ones involving multi-objective optimization problems. On the contrary, ML-based approaches automatically learn the operation thresholds from the data they receive, even in multi-objective problems.

However, the learning behavior of an RL-based scaler heavily depends on the reward function definition. As shown, two slightly different reward functions may result in very different behaviors. Thus, the reward function must be carefully designed, and its effects on the stability of the RL algorithm must be studied.

V. Conclusion and Open Challenges

Automating scaling is investigated in this article to decide the number of NS replicas required to achieve operational, business, and economic goals for multiple stakeholders. Therefore, it is considered a decision-making and multi-objective problem. This perspective proposes closed-loop architectures for next-generation networks as they are more likely to adopt data-driven approaches. Using and extending the MAPE-K framework, intelligent (SL- and RL-based) and non-intelligent (threshold- and control-based) scalers can be swiftly integrated as NIFs in future network infrastructures. To conclude, two scalers are compared, and we observe that only the data-driven approach is adaptable enough to achieve automation, as it adapts correspondingly to system dynamics.

To move forward, some open challenges still need to be addressed. For instance, a flexible architecture is desired to support both ML operation modes (i.e., training and testing) in production networks. To achieve this, new kinds of algorithms must be developed that are transferable to other domains, for example, by training in simulators and transferring that knowledge to real-world networks or by improving the quality of the simulators so that the reality-simulation gap is non-existent [12].

Moreover, a network-wide intelligence orchestrator is also desired to efficiently manage intelligent and non-intelligent algorithms. Among other tasks and responsibilities, this intelligence orchestrator must monitor the NI performance and trigger re-training procedures or replacement by legacy and non-intelligent algorithms if the NI is underperforming. Therefore, the way non-intelligent scalers obtain their knowledge should also be considered in future architectures. For instance, the lengthy process of tuning the PI controller can be seen as the training process of an ML algorithm. Finally, we need a better understanding of the stability effects of NI solutions based on RL since their learning behavior heavily depends on the reward function definition.

ACKNOWLEDGMENT

This research was funded by Ctrl App, an FWO research project (Grant Agreement No. G055619N), and by DAEMON, an H2020 EU-funded programme (Grant Agreement No. 101017109).

REFERENCES

[1] ETSI, "Zero-touch network and Service Management (ZSM): Means of Automation," ETSI, Group Report, 2020-05.

[2] D. M. Manias and A. Shami, "The need for advanced intelligence in nfv management and orchestration," *IEEE Network*, vol. 35, no. 1, pp. 365–371, 2020.

[3] R. Boutaba *et al.*, "A comprehensive survey on machine learning for networking: evolution, applications and research opportunities," *Journal of Internet Services and Applications*, vol. 9, no. 1, p. 16, 2018.

[4] M. Camelo *et al.*, "Requirements and Specifications for the Orchestration of Network Intelligence in 6G," in *2022 IEEE Annual Consumer Communications & Networking Conference (CCNC)*. IEEE, 2022, pp. 1–9.

[5] Y. Wang *et al.*, "From design to practice: ETSI ENI reference architecture and instantiation for network management and orchestration using artificial intelligence," *IEEE Communications Standards Magazine*, vol. 4, no. 3, pp. 38–45, 2020.

[6] O. Gheibi *et al.*, "Applying machine learning in self-adaptive systems: A systematic literature review," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 15, no. 3, pp. 1–37, 2021.

[7] O. Adamuz-Hinojosa *et al.*, "Automated network service scaling in nfv: Concepts, mechanisms and scaling workflow," *IEEE Communications Magazine*, vol. 56, no. 7, pp. 162–169, 2018.

[8] S. Dutta *et al.*, "Qoe-aware elasticity support in cloud-native 5g systems," in *2016 IEEE International Conference on Communications (ICC)*. IEEE, 2016, pp. 1–6.

[9] D. De Vleeschauwer *et al.*, "5growth data-driven ai-based scaling," in *2021 Joint European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit)*. IEEE, 2021, pp. 383–388.

[10] T. Subramanya and R. Riggio, "Centralized and federated learning for predictive vnf autoscaling in multi-domain 5g networks and beyond," *IEEE Transactions on Network and Service Management*, vol. 18, no. 1, pp. 63–78, 2021.

[11] H. Sami *et al.*, "Ai-based resource provisioning of ioe services in 6g: A deep reinforcement learning approach," *IEEE Transactions on Network and Service Management*, vol. 18, no. 3, pp. 3527–3540, 2021.

[12] P. Almasan *et al.*, "Network digital twin: Context, enabling technologies and opportunities," *IEEE Communications Magazine*, pp. 1–13, 2022.

[13] Q. Zhu and G. Agrawal, "Resource provisioning with budget constraints for adaptive applications in cloud environments," *IEEE Transactions on Services Computing*, vol. 5, no. 4, pp. 497–511, 2012.

[14] T. Song *et al.*, "Performance evaluation of integrated smart energy solutions through large-scale simulations," in *2011 IEEE International Conference on Smart Grid Communications (SmartGridComm)*. IEEE, 2011, pp. 37–42.

[15] J. Schulman *et al.*, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

BIOGRAPHIES

Paola Soto is a Ph.D. researcher at the University of Antwerp-imec. She received her B.Sc. in Electronics and her M.Sc. in Telecommunications Engineering from the University of Antioquia, Colombia, in 2014 and 2018, respectively. Her current research is focused on developing network management strategies using artificial intelligence and machine learning.

Miguel Camelo received a master's degree in systems and computer engineering (University of Los Andes, Colombia, 2010) and a Ph.D. degree in computer engineering (University of Girona, Spain, 2014). He has authored several papers in international conferences/journals. He is a Senior Researcher at the University of Antwerp-imec, Belgium, where he leads the research on applied artificial intelligence (AI) in networking. His research interests are in the field of applied AI in communication networks.

Danny De Vleeschauwer received his electrical engineering and Ph.D. degrees in applied sciences from Ghent University, Belgium, in 1985 and 1993, respectively. Since 1998 he is working for Alcatel, later Alcatel-Lucent, and now Nokia Bell Labs in Antwerp, Belgium. His research includes signal processing, queuing theory, and their application to networked applications over packet-based networks, as well as using machine learning to solve networking problems. He is the author/co-author of more than 100 papers and 20 patents.

Yorick De Bock obtained his Doctor Degree in Applied Engineering at the University of Antwerp on hard real-time virtualization for multi-core embedded systems. Yorick is a member of the IDLab research group, a joint research initiative between the University of Antwerp and Ghent University, and a core research group of imec. Currently, he is working as a software developer focusing on making prototypes for multiple IoT and AI projects.

Chia-Yu Chang received his Ph.D. from Sorbonne Université, France, and is currently a senior research engineer at Nokia Bell Labs, Belgium. He has more than 12 years of experience in algorithm/protocol research on communication systems and network applications in academic and industrial laboratories, including EURECOM Research Institute, MediaTek, Huawei Swedish Research Center, and Nokia Bell Labs. His research interests include wireless communication, computer networking, low-latency low-loss scalable throughput (L4S), and AI/ML-supported network control.

Juan F. Botero is an Assistant Professor in the Electronics and Telecommunications Engineering Department at the University of Antioquia, Medellin, Colombia. In 2013 he received his Ph.D. in Telematics Engineering from the Technical University of Catalonia, UPC, in Barcelona, Spain. In 2013, he joined GITA (research group on applied telecommunications) at the Electronics and Telecommunications Engineering Department. His main research interests include Network Management, Software Defined Networking, Network Virtualization, Network Functions Virtualization and resource allocation.

Steven Latré received a Ph.D. in computer science engineering from Ghent University, Belgium, in 2011. He has authored over 100 papers in international journals/conferences. His research expertise focuses on machine learning for low-power environments. He is a recipient of the IEEE COMSOC Award for the Best Ph.D. in Network and Service Management (2012), the IEEE NOMS Young Professional Award (2014), the IEEE COMSOC Young Professional Award (2015), and the Laureate of the Belgian Academy (2019).