A Novel DRL Framework for Cross-Domain Network Scaling in 6G Networks

Abdelmounaim Bouroudi, Abdelkader Outtagarts, Yassine Hadjadj-Aoul

▶ To cite this version:

Abdelmounaim Bouroudi, Abdelkader Outtagarts, Yassine Hadjadj-Aoul. A Novel DRL Framework for Cross-Domain Network Scaling in 6G Networks. HPSR 2024 - IEEE 25th International Conference on High Performance Switching and Routing, Jul 2024, Pisa, Italy. pp.118-123, 10.1109/HPSR62440.2024.10635964 . hal-04749290

HAL Id: hal-04749290
https://inria.hal.science/hal-04749290v1

Submitted on 23 Oct 2024

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License

A Novel DRL Framework for Cross-Domain Network Scaling in 6G Networks

Abdelmounaim Bouroudi
Bell Labs
Nokia Networks France
Paris, France
abdelmounaim.bouroudi@nokia.com

Abdelkader Outtagarts
Bell Labs
Nokia Networks France
Paris, France
abdelkader.outtagarts@nokia-bell-labs.com

Yassine Hadjadj-Aoul
INRIA
Univ Rennes, CNRS, IRISA
Rennes, France
yassine.hadjadj-aoul@irisa.fr

***Abstract*—The advent of 6G requires efficient management of heterogeneous networks and computational resources to achieve the targeted End-to-End network automation, in which slice orchestration is a key feature. Despite the opportunities offered by the recent advances in network virtualization and distributed cloud infrastructure, these developments introduce complexities in the context of multi-domain networks. This paper presents a distributed horizontal scaling method that leverages deep reinforcement learning (DRL) to enhance network function orchestration (NFO) with intelligent scaling decisions, facilitating seamless cross-domain information exchange. First, we develop a DRL agent designed to handle fluctuating traffic loads and generate scaling actions tailored for the considered network function (NF). This trained DRL agent is then integrated into a multi-domain message exchange scaling framework with traffic prediction capabilities. Moreover, we develop a simulation testbed to manipulate multi-domain topologies, customize network slices, and enable precise per-slice and per-domain scaling decisions. Our DRL-based solution outperforms the Horizontal Pod Autoscaling heuristic used by Kubernetes, improving resource utilization and reducing data rate losses.**

***Index Terms*—Mutli-Domain Orchestration, Distributed Scaling, 6G Network, Deep Reinforcement Learning.**

I. Introduction

6G networks mark a significant evolution in mobile network infrastructure, with a focus on virtualization, cloud and distributed computing [1]. Initiatives such as UNEXT [2] are driving this evolution with an architectural framework that incorporates multi-cloud/multi-domain integration as a fundamental aspect of such an E2E 6G network. This approach aims to unify resources and improve network performance. The progression towards predictive auto-scaling orchestration for cloud-native telecom services underscores the importance of dynamic resource management in enhancing network agility and efficiency [3].

Meeting the challenges posed by these advances requires the adoption of cutting-edge technologies such as distributed deep reinforcement learning (D-DRL) and more generally machine learning. These technologies enable collaboration between operational entities and domains, resulting in improved network performance, reduced latency and improved efficiency.

Multi-Domain Orchestration (MDO) is a crucial research area in 6G networking, facilitating service management across heterogeneous resources owned by various providers [4]. It relies on effective management of various network domains with the objective to coordinate them seamlessly while guaranteeing high scalability and fault tolerance. Different orchestration strategies have been applied throughout the literature, including hierarchical orchestration [5], [6], federation-based orchestration [7], and distributed orchestration [8]. Recent advancements in MDO research leverage Multi-Agent Deep Reinforcement Learning (MADRL) for sophisticated resource allocation and orchestration. Innovations include inter-slice cooperative DRL for enhanced resource efficiency in multi-layer MEC environments [9], and novel MADRL algorithms for 5G V2V communications in the absence of base station coverage, improving packet reception in congested scenarios [10]. Furthermore, distributed DRL approaches for network slicing and cooperative resource allocation among non-cellular networks highlight the move towards decentralized execution with centralized learning [11], [12].

This paper explores a novel approach that leverages DRL agents and communication-driven information exchange within a distributed orchestration framework to address the challenges of NF scaling in 6G. This approach aims to achieve the benefits of distributed orchestration while mitigating its shortcomings through effective communication and intelligent decision-making mechanisms. By facilitating seamless communication and information exchange, this communication-driven approach paves the way for efficient resource utilization and improved network performance in the dynamic environment of the envisioned 6G networks. The main contributions of our paper are summarized bellow:

- We propose a formulation of the multi-domain NF scaling problem, including models for both resource consumption and traffic generation.
- We formulate the scaling problem within a single domain as a Markov Decision Process (MDP), allowing the development of DRL algorithms to NF scaling problems.
- We develop a novel distributed DRL algorithmintegrated into a scaling framework with efficient message exchange and traffic forecasting capabilities.
- We evaluate our proposed solution through simulations, benchmarking its performance against the Horizontal Pod Scaling (HPA) algorithm [13].

The rest of this paper is organized as follows. In section II, we detail our approach to formulating the multi-domain Network Function scaling problem, along with models for resource consumption and traffic generation. In section III, we introduce Deep Reinforcement Learning and detail the MDP formulation for the single-domain DRL scaling agent. Building on this, section IV describes the design and implementation of our distributed scaling framework, including the integration of the DRL algorithm. In section V, we discuss the experimental setup and results, with a comparison to the HPA heuristic. Finally, section VI concludes our paper.

II. Problem Formulation

A. Distributed Network Function Scaling Problem

We consider a network system consisting of a set of $\mathcal{K} := \{1, \ldots K\}$ independent domains, where we deploy a set of $\mathcal{S} := \{1, \ldots, S\}$ end-to-end slice. Each slice s in domain k is composed of a set $\mathcal{N}' := \{1, \ldots, N'_{k,s}\}$ of Network Function (NF), and $\mathcal{L}' := \{1, \ldots, L'_{k,s}\}$ Virtual Link (VL) deployed on $\mathcal{N}_k$ nodes and $\mathcal{L}_k$ links. Each network function $n^{'}_{k,s}$ requests various types of resources denoted by a vector $h_{n'_{k,s}} = [h_{n'_{k,s},0}, \ldots, h_{n'_{k,s},D_{\text{NF}}}]$ where D_{NF} is the number of resource type of network functions. The same thing applies for virtual links, where each $l^{'}_{k,s}$ request a set of resources denoted by a vector $f_{l'_{k,s}} = [f_{l'_{k,s},0}, \ldots, f_{l'_{k,s},D_{\text{VL}}}]$ and D_{VL} is the resource type number for virtual links. Together, the resource definitions of all composing NFs and VLs define the pre-defined resource requirements, denoted as $\Xi_{k,s}$, for slice s in domain k.

In this work we focus on the scaling in and out of NFs based on one resource type which is the CPU consumption. Each network function $n'_{k,s}$ has an actual CPU consumption $\mathit{CPU}_{\text{cur}}$, a max CPU consumption $\mathit{CPU}_{\text{max}}$ and higher thresholds $\alpha_{n'_{k,s}}$ where :

$$\mathit{CPU}_{\text{cur}} < \mathit{CPU}_{\text{max}} \times \alpha_{n'_{k,s}} \tag{1}$$

To guarantee the continuity of the service in the system, each NF n' in slice s and domain k must respect the following constraint :

$$1 \leq \Phi^{n^{'}_{k,s}}_{cur} < \Phi^{n^{'}_{k,s}}_{max} \tag{2}$$

Such that $\Phi^{n^{'}_{k,s}}_{max}$ is the maximum number of instance of n' in slice s and domain k.

B. Traffic Generation and Resource Modeling

To ensure the study's relevance and credibility, a sophisticated traffic generation model that reflects real-world scenarios is employed within the testbed. The traffic flow within a 5G/6G networks is simulated, incorporating source and destination modules representing the User Equipment (UE) and the Data Network (DN), respectively. This setup facilitate a close examination of the interaction between traffic generation and network function performance across different 5G/6G E2E domains. The generated traffic, is formulated following the equation (3) as the sum of sub-sinusoidal functions with varying periods and amplitudes, inspired by the work in [14]:

$$Y(t, T_i) = a_{0,i} + \sum_{1}^{k} a_k \sin\left[2\pi \frac{t - t_k}{T_i}\right] + \epsilon_t \tag{3}$$

Where $a_{0,i}$ represents the base amplitude varying by weekday, a_k denotes the amplitudes of constituent functions, ϵ_t is random noise, and T_i is the traffic period.

Throughout the paper, data rate refers to the amount of data generated per unit time, measured in bytes per second (B/s). For each data packet, a fixed data length is configured, and the number of packets generated per second is determined based on the data rate produced by the traffic generation equation. This calculation involves inter-packet production time, following a *Poisson* process with parameter λ.

On the other hand, resource consumption is modeled by calculating the CPU usage of each network function based on the incoming data received (in Byte) and a function-specific CPU Consumption Per Byte (CPB) parameter. This relationship is expressed by the following equation:

$$\mathit{cpu}_{\text{cons}} = \frac{\mathit{dataReceived} \times \mathit{cpu}_{\text{cpb}}}{\Delta t} \tag{4}$$

Where, $\mathit{cpu}_{\text{cons}}$ is the CPU consumption in a time unit Δt and $\mathit{cpu}_{\text{cpb}}$ is the CPU consumption per byte.

III. Scaling Network Functions With DRL

A. Deep Reinforcement Learning overview

Reinforcement Learning (RL) is a method for solving complex control problems through sequential decision-making within an environment E. An agent learns by taking actions a_t at time t. It receives rewards r_t based on action quality, aiming to maximize the expected cumulative discounted return R_t defined as $R_t = \mathbb{E}\left[\sum_t \gamma^t r_t\right]$, where $\gamma < 1$ is the discount factor ensuring the series convergence. This process is formulated as a Markov Decision Process (MDP) with State space S, Action space A, Reward function $R(s_t, a_t)$, and Transition probability $P(s_{t+1}|s_t, a_t)$.

Built on RL's fundamentals, Deep Reinforcement Learning uses Deep Neural Networks (DNNs) to represent complex policies and value functions, offering significant advantages in environments with large state and action spaces, typical of network function scaling challenges. A key development in DRL is Actor-Critic methods, where an Actor learns policy functions by mapping states to actions, and a Critic evaluates these state or state-action pair values, steering the Actor toward better policies. For this work, we choose the Deep Deterministic Policy Gradient (DDPG) algorithm [15], a renowned actor-critic approach that employs DNNs for both Actor and Critic models. DDPG merges Q-learning and off-policy learning, ideal for handling continuous action spaces, making it well-suited for dynamic and efficient network function scaling.

B. Formulation of the Markov Decision Process

To appy the DDPG algorithm to horizontal scaling problem, we first formulate it as an MDP, defining state, action, and reward accordingly:

- **State** : we propose a state vector reflecting key network aspects to capture traffic dynamics and resource consumption influences. At time step t, it is defined as:

$$\left[\frac{cpu_{\text{max}} - cpu_{\text{cur}}}{cpu_{\text{max}}}, \frac{\Phi_{cur}^{n'_{k,s}}}{\Phi_{\text{max}}^{n'_{k,s}}}, \Delta dr_t \times \frac{dr_t}{dr_{t-1}}\right] \tag{5}$$

  The first component measures the gap between current and maximum CPU consumption, aiming for optimal resource efficiency. The second quantifies the current versus maximum allowed Network Function (NF) instance replicas, with $\Phi_{\text{max}}^{n'k,s}$ set higher for training. The third component tracks data rate change over time, with $\Delta dr_t = dr_t - dr_{t-1}; t > 0$ indicating the direction of this change.
- **Action** : To allow the DRL agent to make precise scaling decisions, the action space is continuous, where any action a_t falls within $[-\beta, \beta]$, with β defining the action space boundary. The continuous action a_t is then rounded to the nearest integer $\tilde{a}_t$.
- **Reward** : The goal is to maximize CPU resource efficiency for Network Functions while avoiding service degradation or over-scaling. We divide the reward into two parts

$$R_{\text{global}} = R_{\text{eff}} + R_{\text{penal}} \tag{6}$$

  With the R_{eff} we aim to encourages the agent to approach $cpu_{\text{max}} \times \alpha_{n'_{k,s}}$, penalizing deviation from this target value:

$$R_{\text{eff}} = clip\left(\frac{-|cpu_{\text{cur}} - cpu_{\text{max}} \times \alpha_{n'_{k,s}}|}{cpu_{\text{max}}}, -1, 0\right) \times R_{\text{max}} \tag{7}$$

  With the R_{penal}, we aim at penalizing the agent from taking dangerous action. For our case, the most dangerous action is the deletion of slice by deleting all its NFs, for that, we penalize the agent based on its deviation from the minimum instance number (which should be one). Additionally, to avoid create an infinite number of NFs replicas in the system, a threshold, $\Phi^{n'k,s}$max, is set as the maximum allowable number of NF instances. Actions leading to NF instances exceeding this limit result in penalties for the agent.

$$R_{penal} = \begin{cases} clip\left(\frac{\Phi_{cur}^{n'_{k,s}} + a_t}{\Phi_{\text{cur}}^{n'_{k,s}}}, -1, 0\right) \times R_{\text{max}}, & \Phi_{cur}^{n'_{k,s}} + a_t \leq 0 \\ -1 \times R_{max}, & \Phi_{cur}^{n'_{k,s}} + a_t > \Phi_{max}^{n'_{k,s}} \\ 0, & else \end{cases} \tag{8}$$

  Here, R_{max} bounds the reward to prevent excessive penalties. The agent's maximum reward is zero, motivating proximity to target values while respecting slice requirements, and assuring service continuity by keeping the slice active.

This MDP formulation is well-suited for continuous action space, off-policy algorithms such as DDPG. In our design, the agent is not confined to a set of discrete actions but can take any scaling step, providing a broader range of actions compared to discrete action DRL agents. For each NF in slice s within domain k, a specific Actor Network is trained to take actions in a distributed, multi-domain environment.

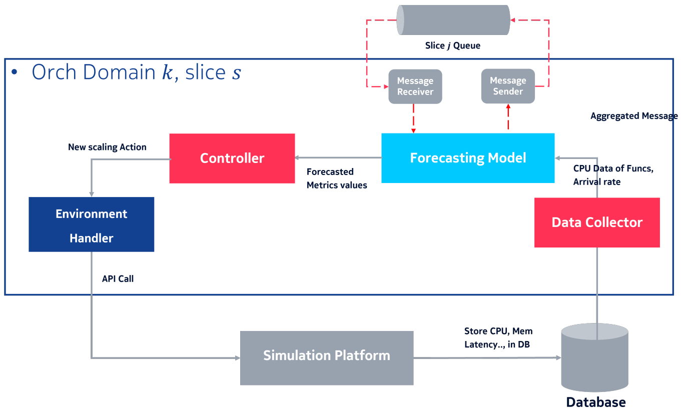

Fig. 1: Distributed Scaling Framework & Orchestrator Components.

IV. Distributed DRL Scaling Algorithm

To assess the performance of our scaling algorithms, we have developed a distributed scaling framework integrated with a specifically designed testbed. The testbed was developed based on OMNeT++ [16] and aimed at examining the impact of inter-domain communication and information sharing among autonomous orchestrators on scaling decision quality. At the heart of this setup is the orchestrator, depicted in Figure 1, which includes a sequence of components including a Data Collector, the Forecasting Model, a Controller, and the Environment Handler. These components work in concert to gather metrics, forecast network behavior, make scaling decisions, and execute these decisions, ensuring a synchronized, proactive, and resource-efficient network operation. It also integrates RabbitMQ for asynchronous and efficient messaging between independent orchestrators. This orchestrated approach provides each network slice within a domain with a dedicated orchestrator, ensuring real-time data exchange, proactive decision-making, and precise resource management, thereby optimizing service delivery.

In Algorithm 1, we present a DDPG-based distributed scaling algorithm with inter-domain message exchange for optimizing network slice scaling in domain k. Our methodology integrates a sophisticated scaling algorithm, specifically DDPG, coupled with an LSTM-based traffic forecasting model. This model analyzes historical data to predict future data rate evolution, sharing these predictions with neighboring domains to support more informed scaling decisions. The algorithm operates continuously, divided into training and evaluation phases. Initially, we set up the forecasting model, DDPG actor, critic networks, and target networks' parameters (lines 1 to 4), followed by acquiring the current network state (line 5). During the training phase (lines 7 to 27), a scaling action is generated using the actor network (line 8), then executed to obtain a new state s', a reward r_t, and a completion

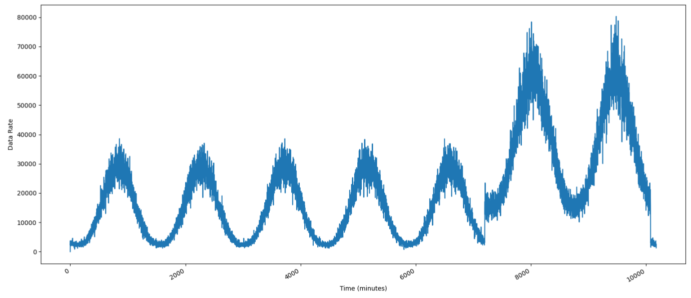

Fig. 2: Data rate pattern in RAN and Core.

signal d (line 9); these are stored as a transition in the buffer D (line 10). Upon reaching the update interval (lines 11 to 19), a batch of size M is randomly sampled from buffer D (line 12), and the target value is computed as per the equation in line 14. Subsequently, the critic network is updated using gradient descent on the squared loss (line 15), and the actor network is updated through gradient ascent (line 16). Target networks are updated using Polyak averaging (line 18). Finally, the state is refreshed (line 20), and upon reaching a terminal state (if done is true), the environment is reset to a new initial state (lines 22 and 23). In the evaluation phase, we distinguish domains as sender or receiver. For sender domains (lines 28 to 30), upon reaching the forecasting threshold, we predict future data rate values and transmit these forecasts to the receiver domain. On the receiver domain's side (lines 35 to 40), if we receive a message, we extract the forecasted data rate values and we update the state only if there's an anticipated increase in traffic. Subsequently, a scaling action is determined based on the actor network's learned policy. This action is then executed, and the state is updated accordingly (lines 41 to 42).

V. Simulation Results

To evaluate our solution, we simulated an E2E 5G network with two domains: the Radio Access Network (RAN) and the Core Domain, linked by virtual connections and hosting a custom substrate network. An E2E slice was deployed across these domains, featuring three adaptable Network Functions: Central Unit (CU) and Distributed Unit (DU) in the RAN, and a User Plane Function (UPF) in the Core. Given that CU and DU scaling is less common in 5G deployments, our focus was primarily on UPF scaling within the Core domain, with UPF scaling parameters provided in Table I. During the validation phase, NF parameters were altered to assess the robustness of our approach.

Figure 2 illustrates the traffic pattern used for training and validation, generated according to the model described in subsection II-B. We generate a traffic for one week with daily period (60×24 in minutes) and a weekly period with a higher amplitude during the weekend (6th and 7th days). Added to that a noise following a normal distribution $\epsilon_t \in \mathcal{N}(0, 0.2)$.

Figure 3 shows the penalty and efficiency reward during the learning phase. Initially, the agent shows bad efficiency and penalty rewards. This can be explained by the period of

Algorithm 1: DRL-based Distributed Scaling Algorithm.

```
1  begin
     Input: $s$, $k$, $\tau$, $q$, T, $send$, $training$
2    Initialize Forecasting model $F$ with $\theta^F$, $\sigma, \delta$
3    Initialize Networks Critic $Q$ and Actor $\mu$ with
       parameters $\theta^Q$, $\theta^\mu$, Reply buffer $D$
4    Initialize Target networks $Q'$, $\mu'$
5    $s_t$ = GetSimulationState $(s, k)$
6    while $TRUE$ do
7      if $training$ then
8        $a_t = \mu(s_t|\theta^\mu) + \epsilon$ s.t $\epsilon \sim \mathcal{N}$
9        $r_t, s', d$ = ExecuteAction $(a_t)$
10       Store $(s_t, a_t, s', r_t, d) \rightarrow D$
11       if $update\ time$ then
12         Sample $B = \{(s_t, a_t, s', r_t, d)\} \leftarrow$D
13         Compute targets
14         $y = r_t + \gamma(1 - done)Q_\theta'(s', \mu_\theta'(s'))$
           Update Critic and Actor networks:
15         $\nabla_{\theta^Q} \frac{1}{|B|} \sum_{s_t, a_t, s', r_t, d}(Q_\theta(s_t, a_t) - y)^2$
16         $\nabla_{\theta_\mu} \frac{1}{|B|} \sum_{s_t} Q_\theta(s_t, \mu_\theta)$
17         Update target Networks :
18         $\theta^{Q'} \leftarrow \tau\theta^Q + (1 - \tau)\theta^{Q'}$
            $\theta^{\mu'} \leftarrow \tau\theta^\mu + (1 - \tau)\theta^{\mu'}$
19       end if
20       $s_t \leftarrow s'$
21       if $d$ then
22         RestEnvironment ()
23         $s_t$ = GetSimulationState $(s, k)$
24       end if
25     end if
26     else
27       $dr$ = GetDataRate $(s, k)$
28       if $len(dr) > \sigma$ $and$ $send$ then
29         $dr_{fr} = F_{\theta^F}(dr, \delta)$
30         SendMessage $(dr_{fr}$, $q)$
31       end if
32       else
33         $msg$ = $(q)$
34         if $msg$ $is$ $not$ $empty$ then
35           Extract forecasted data rate $dr_{fr}$
36           if $dr_{fr} > dr$ then
37             $s_t$ = UpdateState $(s_t$, $dr_{fr})$
38           end if
39         end if
40       end if
41       $a_t = \mu(s_t|\theta^\mu)$
42       $r_t, s', d$ = ExecuteAction $(a_t)$; $s_t \leftarrow s'$
43     end if
44   end while
45 end
```

Phase	cpu_{max}	cpu_{cpb}	α_{UPF}
Training	400	0.2	80%
Validation	500	0.4	80%

TABLE I: UPF parameters for training and validation.

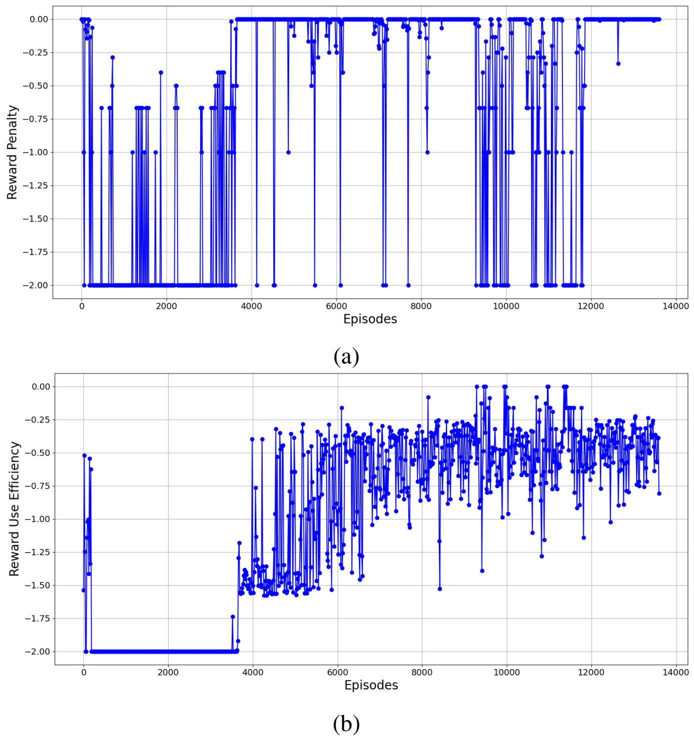

Fig. 3: (a) Penalty reward per episode. (b) Use Efficiency reward per episode.

exploration where the agent explore different actions including prohibited ones, like deleting service or over-dimensioned scaling actions (Fig. 3a). Over time, the agent gradually learns to avoid these actions, allowing it to focus on resource usage efficiency, which reaches near zero reward at the end of the training (Fig. 3b).

In the validation phase, we assess the DRL agent's performance within a distributed scaling framework, facilitating message exchange over a week. This setup allows us to compare DRL-based scaling directly against the Horizontal Pod Autoscaler (HPA) heuristic from Kubernetes, a reactive method that scales according to the demand at each time instance t. Our comparison focuses on the actions, and scaling patterns of both methods. To further evaluate our method's effectiveness, we introduce two metrics: Loss Rate (LR), indicating the percentage of data loss due to insufficient CPU resources, and Resource Use Efficiency (RUE), measuring CPU usage efficiency against incoming traffic relative to an NF's maximum CPU capacity, with the main objective is to be close to the targeted threshold.

$$LR = \max(\frac{ReceivedData}{MaxDataConsumption} - 1, 0) \qquad (9)$$

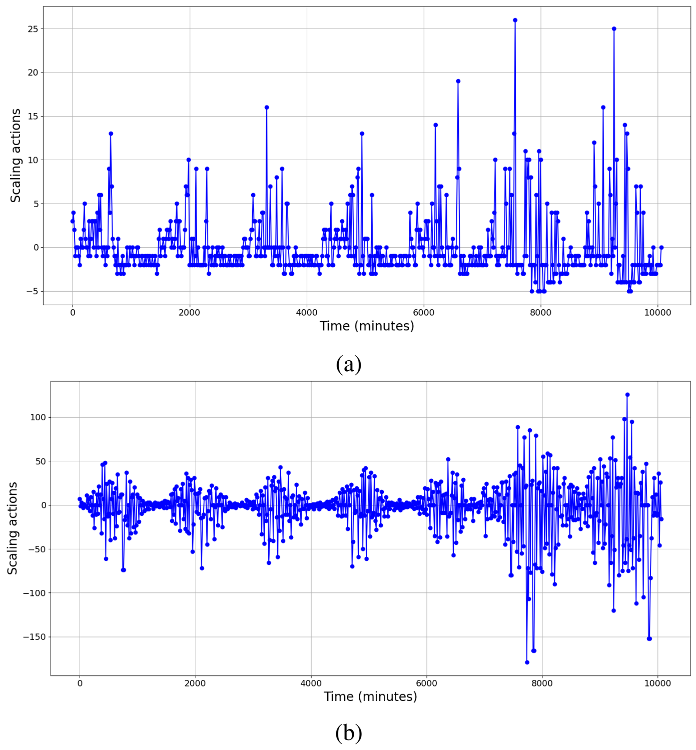

Fig. 4: (a) Sacling actions taken by DDPG. (b) Scaling actions taken by HPA.

$$RUE = \frac{cpu_{cur}}{cpu_{max}} * 100 \qquad (10)$$

In Figure 4, we illustrate the actions taken by the DDPG agent and HPA algorithm on the UPF over a week. In 4a, we observe that DDPG generates harmonized actions with steady growth proportionally to the data rate fluctuations. In the other side (Fig. 4b), HPA's actions present significant oscillation with frequent and successive creation and deletion actions. This unstable behavior can be explained by the reactive nature of the HPA algorithm which strives to match the data rate fluctuations and its implications on the resource consumption. On the other side, DDPG explore the traffic pattern and data rate predictions exchanged between domains to generate more accurate actions avoiding corrective actions. Consequently, we have a compact action bound of [-5, 25] for DDPG while HPA take bigger actions both in creation and deleting [-150, 100] which is a huge performance enhancement.

Figure 5 presents the RUE values for both DDPG and HPA algorithms. This results are the mean RUE value over five weeks executions. We compare these methods in terms of respecting the desired RUE threshold which is 80%. We can see that DDPG consistently achieves an average RUE close to the higher threshold, even during weekends where the data rate is more important (last two days). In contrast, HPA shows an average RUE around 90% even during weekdays which present lower data rates. This indicates less efficient resource utilization and less reliable since it fails to respect the desired threshold. These results demonstrate the ability of

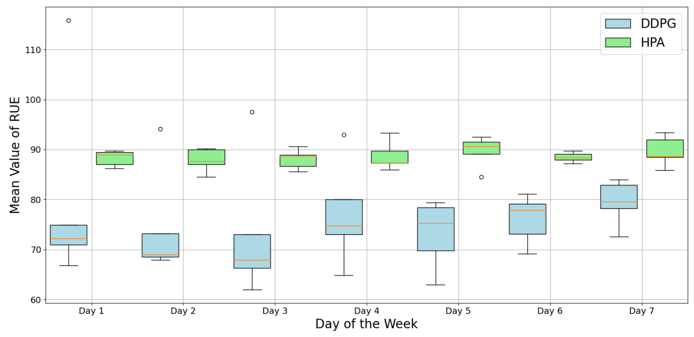

Fig. 5: RUE comparison between DDPG and HPA.

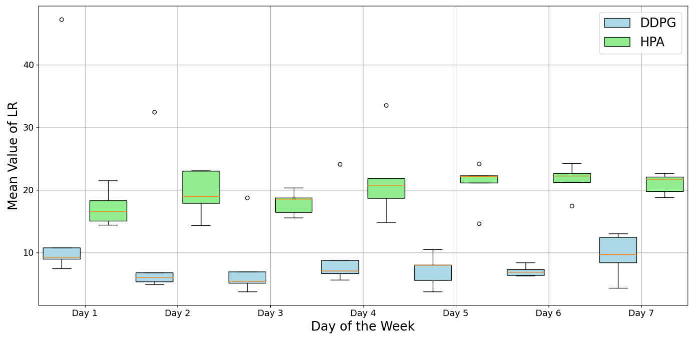

Fig. 6: Loss Rate comparison between DDPG and HPA.

DDPG's to meet slice KPIs even when applied to scenarios with different parameters than those used for training. We also observe some days with suboptimal RUE levels (around 70%) for DDPG. This can be attributed to the influence of forecasted values on scaling actions. By exploiting messages from other domains, the DDPG agent allocate slightly more resources than necessary to anticipate future data rate increases.

In Figure 6, we illustrate the Loss Rate (LR) results from the same five weeks trial for DDPG and HPA. The results reveal a significant improvement in reducing packet loss during the seven days with the DDPG agent compared to HPA. This decrease is attributed to DDPG's stable scaling strategy, which maintains the right number of resources in response to traffic fluctuations. In contrast, HPA shows a variable number of instances due to its wide-ranging scaling actions, aiming to align with the required number of instances to manage traffic, which results in an increased loss rate.

VI. Conclusion

Multi-domain orchestration is a key challenge in 6G networks. It addresses the need to coordinate orchestration actions across different network domains to ensure optimal performance and reliability in end-to-end 6G networks. In this paper, we present a new method for distributed NF scaling based on deep reinforcement learning and message exchange. We formulated NF scaling as an MDP and applied the deep deterministic policy gradient (DDPG) algorithm to solve it. This algorithm involves a training phase, where the agent learns within one domain, and a testing phase, where it is applied to a traffic forecasting and message distribution framework. Our results show that our method outperforms the HPA heuristic in terms of resource efficiency and loss rate. It also offers a more stable and accurate scaling strategy following traffic fluctuations. This research paves the way for further investigation into complex AI-driven orchestration algorithms to fulfill the distributed resource orchestration demands of 6G networks.

References

[1] Xiaohu You et al. Towards 6g wireless communication networks: Vision, enabling technologies, and new paradigm shifts. *Science China Information Sciences*, 64:1–74, 2021.

[2] Csaba Vulkán Azimeh Sefidcon and Markus Gruber. Unext - a unified networking experience. Technical report, Nokia Bell Labs, 2023.

[3] Luong et al. Cloudification and autoscaling orchestration for container-based mobile networks toward 5g: Experimentation, challenges and perspectives. In *2018 IEEE 87th Vehicular Technology Conference (VTC Spring)*, pages 1–7. IEEE, 2018.

[4] Moazzeni Shadi et al. 5g-vios: Towards next generation intelligent inter-domain network service orchestration and resource optimisation. *Computer Networks*, 241:110202, 2024.

[5] Joao F Santos et al. Breaking down network slicing: Hierarchical orchestration of end-to-end networks. *IEEE Communications Magazine*, 58(10):16–22, 2020.

[6] Qiang Liu, Nakjung Choi, and Tao Han. Onslicing: online end-to-end network slicing with reinforcement learning. In *Proceedings of the 17th International Conference on emerging Networking EXperiments and Technologies*, pages 141–153, 2021.

[7] Najmeddine Dhieb, Hakim Ghazzai, Hichem Besbes, and Yehia Massoud. Scalable and secure architecture for distributed iot systems. In *2020 IEEE Technology Engineering Management Conference (TEMSCON)*, pages 1–6, 2020.

[8] Hatim Chergui, Adlen Ksentini, Luis Blanco, and Christos Verikoukis. Toward zero-touch management and orchestration of massive deployment of network slices in 6g. *IEEE Wireless Communications*, 29(1):86–93, 2022.

[9] Yohan Kim and Hyuk Lim. Multi-agent reinforcement learning-based resource management for end-to-end network slicing. *IEEE Access*, 9:56178–56190, 2021.

[10] Alperen Gündoğan, H Murat Gürsu, Volker Pauli, and Wolfgang Kellerer. Distributed resource allocation with multi-agent deep reinforcement learning for 5g-v2v communication. In *Proceedings of the Twenty-First International Symposium on Theory, Algorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing*, pages 357–362, 2020.

[11] Federico Mason, Gianfranco Nencioni, and Andrea Zanella. A multi-agent reinforcement learning architecture for network slicing orchestration. In *2021 19th Mediterranean Communication and Computer Networking Conference (MedComNet)*, pages 1–8. IEEE, 2021.

[12] Zoubeir Mlika and Soumaya Cherkaoui. Network slicing for vehicular communications: a multi-agent deep reinforcement learning approach. *Annals of Telecommunications*, 76(9-10):665–683, 2021.

[13] 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.

[14] Alba P. Vela, Anna Vía, Fernando Morales, Marc Ruiz, and Luis Velasco. Traffic generation for telecom cloud-based simulation. In *2016 18th International Conference on Transparent Optical Networks (ICTON)*, pages 1–4, 2016.

[15] Lillicrap et al. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[16] Andras Varga. Omnet++. In *Modeling and tools for network simulation*, pages 35–59. Springer, 2010.