V2N Service Scaling with Deep Reinforcement Learning

Hsu, C.S.-H.; Martín-Pérez, J.; Papagianni, C.; Grosso, P.

DOI
10.1109/NOMS56928.2023.10154358

Publication date
2023

Document Version
Final published version

Published in
Proceedings of IEEE/IFIP Network Operations and Management Symposium 2023

2023 IEEE/IFIP Network Operations and Management Symposium (NOMS 2023) - Shortpaper

V2N Service Scaling with Deep Reinforcement Learning

Cyril Shih-Huan Hsu*, Jorge Martín-Pérez†, Chrysa Papagianni* and Paola Grosso*
*Informatics Institute, University of Amsterdam, The Netherlands
†Departamento de Ingeniería Telemática, Universidad Carlos III de Madrid, Spain
s.h.hsu@uva.nl, jmartinp@it.uc3m.es, c.papagianni@uva.nl, p.grosso@uva.nl

NOMS 2023-2023 IEEE/IFIP Network Operations and Management Symposium | 978-1-6654-7716-1/23/$31.00 ©2023 IEEE | DOI: 10.1109/NOMS56928.2023.10154358

***Abstract*—The fifth generation (5G) of wireless networks is set out to meet the stringent requirements of vehicular use cases. Edge computing resources can aid in this direction by moving processing closer to end-users, reducing latency. However, given the stochastic nature of traffic loads and availability of physical resources, appropriate auto-scaling mechanisms need to be employed to support cost-efficient and performant services. To this end, we employ Deep Reinforcement Learning (DRL) for vertical scaling in Edge computing to support vehicular-to-network communications. We address the problem using Deep Deterministic Policy Gradient (DDPG). As DDPG is a model-free off-policy algorithm for learning continuous actions, we introduce a discretization approach to support discrete scaling actions. Thus we address scalability problems inherent to high-dimensional discrete action spaces. Employing a real-world vehicular trace data set, we show that DDPG outperforms existing solutions, reducing (at minimum) the average number of active CPUs by 23% while increasing the long-term reward by 24%.**

***Index Terms*—V2N, scaling, DRL, DDPG, A2C**

I. Introduction

Connected and Automated Vehicles (CAVs) is a transformative technology for the automobile industry. CAV applications (real-time situational awareness etc.) require process-intensive and low-latency, reliable computing and communication services. Such characteristics prohibit the use of cloud computing resources that are usually centralized into large data centers. An effective approach to address latency requirements is to leverage Edge computing, moving computing resources closer to where the data is being generated, processed, and consumed.

Due to the ubiquity of the cellular infrastructure, 5G systems are set out to support Cellular Vehicle-to-Everything (C-V2X) communications, ensuring ultra-low latency and ultra-high reliability communications (URLLC) under high-density and -mobility conditions. The C-V2X technology, introduced by 3GPP [1], refers to the low-latency communication system between vehicles and vehicles (V2V), pedestrians (V2P), roadside infrastructure (V2I), and cloud/edge servers (network, V2N). Each of these use cases has different communication requirements. 5G systems are expected to address such requirements by slicing the physical network into several tailor-made logical ones e.g., for autonomous-driving, tele-operated driving etc [2]. In this ecosystem, Edge computing is employed to support dynamic service creation and processing per slice.

Nevertheless, appropriate mechanisms must be put in place to ensure elastic network services in order to meet service level agreements. Broadly speaking, elasticity is the ability to increase and shrink selected resources in a systematic and autonomous manner to adapt to workload changes [3]. Similar to [4], using the vehicular traffic from the streets of Turin, we dynamically scale vertically Edge computing resources, to accommodate the latency requirements for V2N applications. However, in this work we employ DRL for deciding on how to vertically scale computing resources.

The strength of RL approaches lies in their ability to reason under uncertainty and adapt to changes at runtime, which maps well onto the stochastic V2N environment. RL has been investigated before for scaling computing resources; authors in [5]–[8] provide ML/RL-based auto-scaling techniques in the context of cloud resource management. ML has been also employed for scaling virtualized network functions [9]–[14]. For instance, DDPG has been utilized to predict a threshold vector of CPU loads that eventually triggers scaling, but not the scaling actions per se [13]. Authors in [14] formulate the scaling of computing resources as a Markov Decision Process (MDP) and propose an RL approach based on Q-Learning. However to enable flexible scaling decisions that can support surges in network traffic, we need to go beyond approaches that employ a limited action space, as such solutions often scale CPU in increments of one. In consequence, this would lead to scalability problems due to the high-dimensional discrete action space. Instead, we propose the use of a DRL approach with continuous action space, introducing a discretization method supporting the scaling actions. Our contributions of are summarized as follows:

- We investigate the use of DDPG for the V2N scaling problem, introducing a discretization method termed as *Deterministic Ordered Discretization* (DOD), forming the *DDPG-DOD* approach. The DOD method can be further applied to off-the-shelf RL algorithms with continuous action space to address discrete problems with ordering properties.
- We compare the performance of the proposed DRL agents with the traditional, prediction and RL algorithms presented in [4], using road traffic traces from Turin. Furthermore we compare against Advantage Actor Critic (A2C) [15], a discrete DRL approach for scaling resources [16]. A2C has been selected as it outperforms respective DRL methods (i.e., Deep Q-Network) in a variety of RL benchmarks [15].

In the following, we describe the V2N system in §II. Then, we

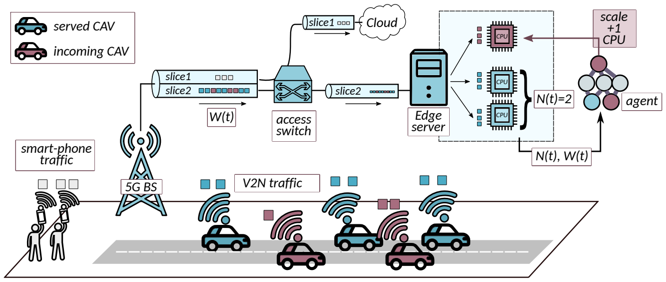


Fig. 1: Considered V2N system.

model the scaling problem as an MDP and introduce DDPG-DOD to address it in §III. Finally, we evaluate the proposed approach (§IV) and highlight our conclusions (§V).

II. V2N system description

For the V2N system, we consider a road segment in the coverage area of a 5G base-station (BS) as depicted in Fig. 1. The BS provides connectivity to smartphone users and CAVs along the road. We assume that every CAV uses V2N-based applications such as remote driving, hazard warning etc., and its traffic is processed in the edge of the network to satisfy latency requirements. We assume smartphones and CAVs are connected to their respective slices i.e., *slice 1* supporting smartphones' traffic processed in the cloud; and *slice 2* supporting V2N traffic processed at the Edge server.

We focus on the workload $W_t \in \mathbb{R}^+$ that V2N services introduce in *slice 2* over time t. If there are V_t vehicles, each of them sending P_v packets/sec, and each CPU c_i processes P_{c_i} packets/sec on time (i.e., satisfying latency requirements); then the workload is expressed as $W_t = P_v V_t / P_{c_i}$. The goal is that the system in Fig. 1 distributes the overall V2N workload W_t among the Edge CPUs to satisfy latency requirements, i.e., the workload $W_t^{c_i}$ dispatched to CPU c_i should satisfy $W_t^{c_i} \leq 1$. Thus, the system should efficiently (vertically) scale the number of CPUs $N_t \in \mathbb{N}^+$ by turning them on/off to process the V2N traffic on time. To that end, we propose using an ML agent (see Fig. 1) that learns the traffic patterns, and anticipates workload fluctuations, meeting delay requirements by scaling up/down the number of CPUs.

III. Problem statement and Proposed Approach

In this section we first discuss the MDP associated to the V2N system described in §II. Then describe how we use DRL to solve the MDP and scale the V2N service.

A. Markov Decision Process

Typically, MDPs are characterized by a tuple $(\mathcal{S}, \mathcal{A}, \mathbb{P}_{a_t}, R)$ denoting the inherent state space $\mathcal{S}$, action space $\mathcal{A}$, transition probability $\mathbb{P}_{a_t}$, and reward function R.

State Space. The state s_t at time t is specified by the number of active CPUs at the Edge server N_t, and the workload W_t associated with the incoming V2N traffic, i.e., the state is defined as the tuple $s_t = (N_t, W_t)$.

Action Space. To increase/decrease or maintain the number of CPUs in the Edge server, we define an action as $a_t \in \mathcal{A} = \{-N_{\max}, \ldots, N_{\max}\}$, with $N_{\max}$ being the maximum number of CPUs in the Edge server.

Transition Probability. Given the current action a_t and state $s_t = (N_t, W_t)$, the transition probability to the next state $s_{t+1} = (N_{t+1}, W_{t+1}) = (N_t + a_t, W_{t+1})$ is determined by $\mathbb{P}_{a_t}(s_{t+1} | \ s_t) = \mathbb{P}(s_{t+1} | \ a_t, s_t)$.

Reward. The reward function R depends on the workload $W_t^{c_i} = \min\left(1, \ x_i \cdot W_t + B_{t-1}^{c_i}\right)$ and backlog $B_t^{c_i} = \max\left(0, \ W_t^{c_i} - 1\right)$ [4]. The workload $W_t^{c_i}$ of a CPU c_i is only a portion $x_i \in [0,1]$ of the total workload W_t plus its prior backlog $B_{t-1}^{c_i}$, i.e., the workload that CPU c_i could not process. As mentioned in §II, the CPU workload should remain below one to satisfy V2N latency requirements, thus clipping at one. Based on the workload and backlog definitions, we define the reward as:

$$R(s_{t+1} | \ s_t, a_t) = \min\{W_{t+1}^{c_i}\}_{i=0}^{N_t+a_t} - \beta \cdot \max\left\{B_{t+1}^{c_i}\right\}_{i=0}^{N_t+a_t} \tag{1}$$

which aims to maximize the CPU utilization by encouraging the least loaded CPU to carry more workload (first term), while penalizing the maximum backlog accumulated by a CPU (second term). The backlog penalty is weighted by a term $\beta \in \mathbb{R}^+$ to control its impact on the reward function.

With the definition of the state and action space, transition probabilities and reward function, we formulate the MDP:

Problem 1 (V2N scaling MDP). *Given the* $(\mathcal{S}, \mathcal{A}, \mathbb{P}_{a_t}, R)$ *tuple, find a policy* π *that maximizes:*

$$\mathbb{E}_{\substack{a_t \sim \pi, \\ s_{t+1} \sim \mathbb{P}_{a_t}(s_t)}} \left[\sum_t \gamma^t \left(\min\{W_{t+1}^{c_i}\}_{i=0}^{N_t+a_t} - \beta \cdot \max\left\{B_{t+1}^{c_i}\right\}_{i=0}^{N_t+a_t} \right) \right] \tag{2}$$

with $\gamma \in [0,1]$ *being the discount factor.*

In other words, we aim to find an optimal policy π to maximize the expected discounted reward. We resort to a model-free RL approach to find an optimal policy π without making assumptions about the transition probabilities.

B. V2N scaling with DDPG-DOD

RL finds an optimal scaling policy π for *Problem 1* using an estimation of the expected discounted reward (2). Such estimation is known as the expected gain $\mathbb{E}[G_t] = \mathbb{E}\left[\sum_t \gamma^t R(s_{t+1} | s_t, a_t)\right]$, and an optimal policy π will take the adequate scaling actions a_t to maximize $\mathbb{E}[G_t]$. In this section we advocate for the following DRL agent to estimate $\mathbb{E}[G_t]$ and look for optimal scaling policies π for *Problem 1*.

DDPG-DOD. DDPG [17] is an RL algorithm that draws from deterministic policy gradient and DQN for learning in continuous action space. Similar to A2C [15], DDPG is based on the actor-critic architecture, where the critic approximates the expected gain $\mathbb{E}[G_t]$ with the action-value function $Q(s_t, \hat{a}_t) = \mathbb{E}[G_t | s_t, \hat{a}_t]$, $\hat{a}_t \in \mathbb{R}$. However, unlike the A2C actor that estimates the probability distribution of actions $\pi(a_t | s_t)$, the DDPG actor learns a deterministic policy $\pi(s_t)$ which generates a real-valued action $\hat{a}_t$.

DDPG is not directly applicable to scaling problems with discrete actions. To map the real-valued action $\hat{a}_t$ to the number of CPUs to scale up/down, we propose a transformation called *Deterministic Ordered Discretization* (DOD). Given the real-valued output of DDPG $\hat{a}_t \in [l, u]$, then DOD is a transformation

$$g(\hat{a}_t) = \underset{a \in \mathcal{A}}{\operatorname{argmin}} \left\| a - \left(\hat{a}_t \frac{2N_{\max}}{u-l} - N_{\max} \frac{u+l}{u-l} \right) \right\| \quad (3)$$

that (i) applies a positive affine transformation that maps the real-valued action $\hat{a}_t$ from the range $[l, u]$ to $[-N_{\max}, N_{\max}]$; and (ii) finds the nearest discrete action $a \in \mathcal{A}$. Using DOD with DDPG presents two advantages:

1) *DOD mitigates the explosion of the action space*. Regardless of the number of available CPUs, DOD always takes a single real-valued action $\hat{a}_t$ given by DDPG and maps it to the discrete action $a_t \in \{-N_{\max}, \ldots, N_{\max}\}$. Scaling DRL solutions in the literature [11] use as many neurons for the output layer as number of CPUs, which makes the output dimension grows as $\mathcal{O}(N_{\max})$.
2) *DOD exploits the internal ordering of the problem*. If the certain action a_t has a higher chance to be selected given the state s_t, its proximate actions (e.g. $a_t \pm 1$) also get higher chances. Learning can become more efficient by leveraging such relations between actions, while typical discrete action RL algorithms (e.g. DQN, A2C) take each action as an independent option, thus the structure of the action space is ignored. The authors in [18], [19] have proposed different approaches to support a similar idea.

Note that we can still update DDPG-DOD agent via policy gradient, as DOD can be seen as a part of the environment (i.e. a step prior to the reward function calculation), which does not play a role in the gradient update.

IV. Performance Evaluation

A. Workload Generation

We consider a real-world dataset with a traffic trace from Corso Orbassano road in Turin, spanning from January 2020 to October 2020. The trace contains the number of cars that pass via certain measuring points every 5 minutes. We split the complete trace in 80:20 ratio for training and testing purposes respectively. Following the assumptions in [4], $V_t = 8$ vehicles using a video-related V2N service (e.g., remote driving) generate a workload of $W_t = 1$. In other words, a single CPU processes on time the traffic sent by 8 vehicles, i.e., $P_c = 8P_v$. We assume that the total workload W_t is distributed across the different CPUs according to the Dirichlet distribution. That is, the load x_i for each CPU c_i satisfies $\sum_i x_i = 1$, while $\mathbb{P}(x_1, \ldots, x_{N_{\max}}) \sim \prod_i x_i^{\alpha_i - 1}$. With $\alpha_i = 1000$ in our evaluation environment, the workload W_t is almost evenly distributed among the CPUs. The weight of the backlog penalty in the reward function is set to $\beta = 1.0$ [4] and the discount factor is set to $\gamma = 0.99$.

B. State of the art solutions

We compare our scaling approach to other solutions, as presented in [4] and [16], namely:

- a **Proportional Integral** (PI) controller [20] that aims to keep the most loaded CPU below a threshold of $\rho = 0.6$;
- a **Long Short-Term Memory** (LSTM) predictor [21] with 2 layers with 4 cells each, where we use a look back of 3 slots (i.e., a prediction is based on the 3 previous values);
- a **Q-Learning** (RL) algorithm [22] with the same state space and reward function as the proposed one, but only three actions: $a_t \in \mathcal{A} = \{-1, 0, 1\}$
- an **A2C** based scaling approach [15]. To have a fair comparison, we apply similar experimental setup as that of DDPG-DOD, as described in the following subsections.

C. A2C and DDPG-DOD setup

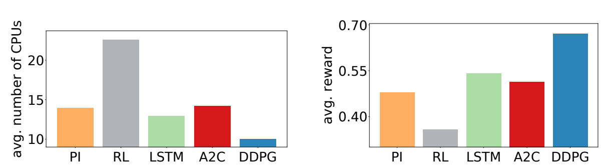
The actor and critic networks of DDPG-DOD and A2C are multi-layer perceptrons (MLPs) with 3 hidden layers of 128 neurons. Learning rate is set to $lr = 3e - 3$. The code implemented by the authors in [4] is used for the environment. We implement A2C and DDPG-DOD with PyTorch 1.10.0. The tests are carried out on a server with an Intel Core i7-10700K CPU and 32 GB of RAM.

D. Metrics and evaluation scenarios

Our goal is to maximize the long term reward of *Problem 1*, which minimizes the operational cost via a proper scaling of computational resources over time. Here we use the *average number of active CPUs* and the *average reward* as metrics. We set up two different evaluation scenarios to assess the performance of the different solutions: (i) a *Performance* scenario where we test every solution over two days in Corso Orbassano using an action space $\mathcal{A} = \{-5, \ldots, 5\}$, $|\mathcal{A}| = 11$; and (ii) a *Scalability* scenario where we study the impact of increasing the action space up to $\mathcal{A} = \{-15, \ldots, 15\}$, $|\mathcal{A}| = 31$, and $\mathcal{A} = \{-25, \ldots, 25\}$, $|\mathcal{A}| = 51$.

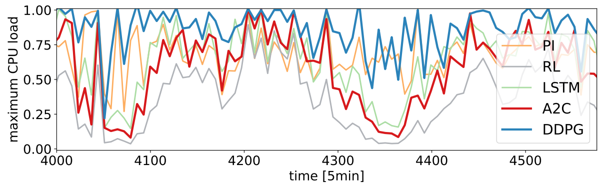
E. Results

Performance. Fig. 2 plots the average number of CPUs and reward that each solution obtains over the testing trace. The Q-learning agent, denoted as RL, maintains on average the largest number of active CPUs, that leads to a reduction in the average reward (first term in formula (1)). The A2C agent performs similar to the PI controller, exhibiting slightly higher average reward (6% increase) for marginally higher number of CPUs (2%). LSTM and DDPG-DOD (denoted hereafter as DDPG for the sake of simplicity) tend to attain larger rewards while keeping the average number of the CPUs low, which essentially reduces the overall cost for the edge resources used. The increased average reward also indicates a reduced backlog for the particular approaches – as indicated by the second term in formula (1). However, DDPG outperforms

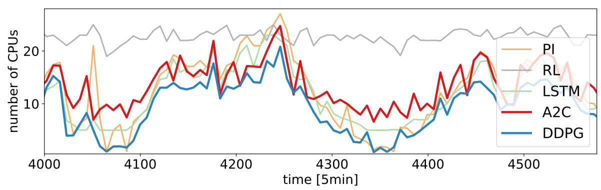


(a) Number of CPUs (b) Reward

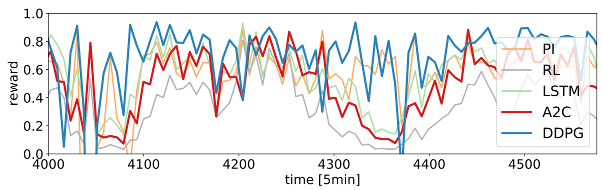
Fig. 2: Average performance metrics



(a) Maximum load of CPUs

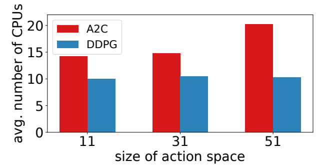


(b) Number of CPUs

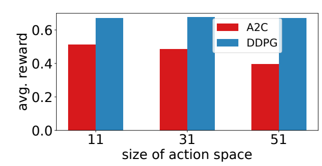


(c) Reward

Fig. 3: Trace of performance over time



(a) Number of CPUs



(b) Reward

Fig. 4: Scalability over various sizes of action space

LSTM, decreasing by approximately 23% the average number of CPUs, while increasing by 24% the average reward.

Fig. 3 illustrates how all approaches perform over a period of two days regarding the maximum CPU load $\max_i\{W_t^{c_i}\}$, number of active CPUs N_t and reward $R(s_{t+1}|\, s_t, a_t)$. The RL agent is the most conservative agent; it maintains a workload of $W_t^{c_i} < 1$ for all CPUs (Fig. 3a) with allocating the largest number of active processors (Fig. 3b), which leads to a reduction in the reward (Fig. 3c). As discussed in the previous paragraph the A2C agent performs similar to the PI controller. However in Fig. 3a we note that A2C avoids overloading the resources compared to PI, while it increases the maximum load compared to more conservative approaches like RL. We further observe in Fig. 3 that DDPG and PI are more aggressive solutions; they activate a sufficient number of CPUs while frequently a CPU gets overloaded, even only for short time period, especially in the case of DDPG (Fig. 3a). However DDPG outperforms PI both in terms of reward as well as cost (average number of active CPUs) as presented also in Fig. 2. Moreover the number of CPUs over time for DDPG (Fig. 3b) evolves smoother compared to the PI case. We argue that the decision boundary of DDPG is smoother than that of the discrete-action approaches as the discretization method takes into account the ordering property of the scaling action space. LSTM and DDPG are the most efficient approaches in terms of the average values of the selected metrics, as discussed in the previous paragraph (Fig. 2). Note that in Fig. 3b DDPG reduces further the number of CPUs compared to LSTM, which leads to the high variation in the maximum load depicted in Fig. 3a. However, Fig. 3c suggests that DDPG keeps the periods with fully loaded CPUs short, thus, the backlog incurred has low impact on the reward.

Scalability. We look into the scalability of the DRL agents regarding the size of the action space. Fig. 4 compares the average reward and the average number of CPUs that are allocated by A2C and DDPG with various dimensions of the action space. The results show that DDPG performs robustly as the size of the action space increases, while for A2C it gets more difficult to converge. DDPG converges in spite of the larger action spaces by exploiting the underlying order between discrete actions. In contrast, A2C takes each action as an independent option. As the structure of the action space is not utilized, thus the learning process is less efficient.

We have also studied how A2C works with the ordinal architecture proposed in [19]. In this case, A2C failed to converge with an action space of size 31 or 51, due to the highly interleaved logits in the forward/backward passes [19] that lead to high memory usage. The converged configuration, for an action space of size 11, performs on average worse than the original A2C with a reward of 0.46 and 14.34 CPUs.

V. Conclusion

In this paper we propose to vertically scale V2N services using DDPG-DOD, a DDPG agent equipped with a non-parametric discretization method that is designed to capture the structure of the scaling decisions and learn discrete actions in a continuous fashion – thus avoiding the action-space explosion. Employing a real-world vehicular trace dataset, we show that DDPG-DOD outperforms state of the art solutions in terms of (i) operational cost as it minimizes the number of active CPUs, (ii) performance; increasing the long-term reward is an indicator of reduced backlog and thus processing delay, and (iii) flexibility in scaling resources as DDPG-DOD performs robustly independently of the size of the action space.

In future work, we plan to investigate the applicability of the proposed approach for V2N service scaling in a Multi-PoP environment, covering a metropolitan area. In such environment with more degrees of freedom in offloading computations, placement decisions should be also taken into consideration.

Acknowledgements

This work has been partially funded by the Spanish Ministry of Economic Affairs and Digital Transformation and the European Commission through the 6G-EDGEDT, 6G-DATADRIVEN and DESIRE6G (grant no. 101095890) projects.

References

[1] 3GPP TS 23.28, "Architecture enhancements for V2X services (Release 16)," Tech. Rep., March 2019.

[2] S. Wijethilaka and M. Liyanage, "Survey on network slicing for Internet of Things realization in 5G networks," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 2, pp. 957–994, 2021.

[3] A. Medeiros, A. Neto *et al.*, "Enabling elasticity control functions for cloud-network slice-defined domains," in *NOMS - IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2020, pp. 1–7.

[4] D. de Vleeschauwer, J. Baranda *et al.*, "5Growth Data-Driven AI-Based Scaling," in *Joint European Conference on Networks and Communications & 6G Summit*. IEEE, 2021, pp. 383–388.

[5] S. Verma and A. Bala, "Auto-scaling techniques for IoT-based cloud applications: a review," *Cluster Computing*, pp. 1–35, 2021.

[6] K. Rzadca, P. Findeisen *et al.*, "Autopilot: Workload autoscaling at google," in *Proceedings of the Fifteenth European Conference on Computer Systems*, ser. EuroSys '20. New York, NY, USA: Association for Computing Machinery, 2020.

[7] L. Toka, G. Dobreff *et al.*, "Machine learning-based scaling management for kubernetes edge clusters," *IEEE Transactions on Network and Service Management*, vol. 18, no. 1, pp. 958–972, 2021.

[8] C. Bitsakos, I. Konstantinou, and N. Koziris, "Derp: A deep reinforcement learning cloud system for elastic resource provisioning," in *2018 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, 2018, pp. 21–29.

[9] T. Subramanya and R. Riggio, "Machine learning-driven scaling and placement of virtual network functions at the network edges," in *Conference on Network Softwarization (NetSoft)*. IEEE, 2019, pp. 414–422.

[10] S. Rahman, T. Ahmed *et al.*, "Auto-scaling VNFs using machine learning to improve QoS and reduce cost," in *International Conference on Communications (ICC)*. IEEE, 2018, pp. 1–6.

[11] Y. Garí, D. A. Monge *et al.*, "Reinforcement learning-based application autoscaling in the cloud: A survey," *Engineering Applications of Artificial Intelligence*, vol. 102, p. 104288, 2021.

[12] L. Cao, P. Sharma *et al.*, "{ENVI}: elastic resource flexing for network function virtualization," in *9th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 17)*, 2017.

[13] H. Chai, "Traffic-aware Threshold Adjustment for NFV Scaling using DDPG," *arXiv:1811.08116*, 2018.

[14] P. Tang, F. Li *et al.*, "Efficient Auto-Scaling Approach in the Telco Cloud Using Self-Learning Algorithm," in *2015 IEEE Global Communications Conference*, 2015, pp. 1–6.

[15] V. Mnih, A. P. Badia *et al.*, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*, 2016, pp. 1928–1937.

[16] W. Beks, "Network Slice Elasticity using Machine Learning [Master's Thesis, VU-UvA]," https://www.ubvu.vu.nl/pub/fulltext/scripties/28_2669090_0.pdf, 2021.

[17] T. P. Lillicrap, J. J. Hunt *et al.*, "Continuous control with deep reinforcement learning," *arXiv:1509.02971*, 2015.

[18] G. Dulac-Arnold, R. Evans *et al.*, "Reinforcement learning in large discrete action spaces.(2015)," *arXiv:1512.07679*, 2015.

[19] Y. Tang and S. Agrawal, "Discretizing continuous action space for on-policy optimization," in *Proceedings of the aaai conference on artificial intelligence*, vol. 34, no. 04, 2020, pp. 5981–5988.

[20] Kiam Heong Ang, G. Chong, and Yun Li, "PID control system analysis, design, and technology," *IEEE Transactions on Control Systems Technology*, vol. 13, no. 4, pp. 559–576, 2005.

[21] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[22] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3, pp. 279–292, May 1992.