Resource Allocation of Multi-user Workloads in Cloud and Edge Data-Centers Using Reinforcement Learning

Julian Jimenez*, Paola Soto*, Danny De Vleeschauwer†, Chia-Yu Chang†
Yorick De Bock*, Steven Latre*, Miguel Camelo*
*University of Antwerp - imec, IDLab - Department of Computer Science, Sint-Pietersvliet 7, 2000 Antwerp, Belgium
†Nokia Bell Labs, Copernicuslaan 50, 2018 Antwerp, Belgium

***Abstract*—Cloud and edge Data-center (DC) are designed to allocate computing resources dynamically to users based on the agreed Service Level Agreement (SLA). However, the ever-increasing demand for beyond 5G services necessitates an efficient workload management. A key challenge in this regard is auto-scaling, a dynamic process that adjusts computing resources to meet fluctuating system demands, optimizing resource utilization and cost efficiency. Traditional auto-scaling algorithms, which rely on fixed thresholds or control-theory, may face limitations in modern DC which are characterized by diverse, dynamic, and multi-user workloads. In this paper, we propose a Reinforcement Learning (RL)-based controller that extends the capacity of the state-of-the-art RL-based auto-scalers to the multi-user workload scenario. We compare the proposed RL agent against the well-known Proportional–Integral (PI) controller and a Threshold (THD)-based controller in a multi-user workload scenario in terms of created Cloud-native Network Functions (CNFs) and peak latency performed in a discrete event simulator.**

***Index Terms*—Auto-scaling, Workload Management, Reinforcement Learning, Cloud-Native Network Function.**

I. Introduction

In the rapidly evolving landscape of cloud and edge Data-center (DC), the demand for beyond 5G services has created new challenges for managing workloads efficiently [1]. One such challenge is the need for auto-scaling, a process that dynamically adjusts the available resources to meet the changing demands of the system. Auto-scaling is pivotal in ensuring optimal resource utilization and cost-efficiency in cloud and edge DC [2]. An example of technological transformations that have triggered the need for novel 5G workloads auto-scaling mechanisms is Mobile Network Operator (MNO) migrating to a Telco Cloud environment. This has been possible as the latest 5G Standalone (5G-SA) network framework includes Cloud-native Network Functions (CNFs), which empowers 5G network functions to operate within a cloud environment. These adaptations grant MNOs the ability to establish a "web-scale" Telco Cloud capable of scaling up and down in response to varying user demands for tailored services. Therefore, the dynamic demands of a "web-scale" Telco Cloud imposes the need for innovative auto-scaling algorithms to manage resource allocation and ensure optimal performance effectively.

In traditional single-user workload management, a cloud provider allocates resources independently to each user [3]. However, this model is not well-suitable for some of the novel beyond 5G workloads, where the cloud provider needs to dynamically allocate and distribute resources among multiple users and their respective workloads, ensuring fair and efficient utilization while adapting to fluctuations in demand. Examples of these workloads are those generated in a Neutral Host Service Provider (NHSP) [4]. A NHSP offers an infrastructure that can be shared among multiple MNOs. As resources are virtualized for each MNO using the Neutral Host (NH) infrastructure, it is critical to ensure that the resource allocation for the different CNFs of each MNO fulfills the MNOs Service Level Agreement (SLA). Therefore, designing a multi-user auto-scaling mechanism that can cope with the dynamics of such environments is fundamental.

Up to date, auto-scaling algorithms in DC have relied on fixed thresholds or control-theory to adjust resource allocation [5]. These algorithms monitor system metrics such as Central Processing Unit (CPU), storage, memory utilization, or number of replicas and trigger scaling actions when predefined thresholds are crossed. While these approaches have been effective in simple scenarios, they face limitations in modern DC characterized by a) heterogeneous, b) dynamic, and c) multi-user workloads. For example, fixed thresholds may fail to capture the complexities of user demands, leading to over provisioning or under provisioning of resources. This is where Machine Learning (ML), particularly Reinforcement Learning (RL), has emerged as a powerful solution to address the first two aspects [3], [6]. By leveraging historical data, user behavior patterns, and system performance metrics, RL algorithms can learn and adapt to changing user demands and optimize resource allocation through direct interaction with the environment.

Most current auto-scalers for multi-user workloads still rely on traditional approaches, which may not capture the complex nature of multi-user workloads, or ML trained in a supervised approach, which requires full training if workloads change drastically [1], [7], [8]. Therefore, there is a clear need to

This work has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement no. 101017109 "DAEMON" and the imec.icon project 5GECO. 5GECO is co-financed by imec and receives financial support from Flanders Innovation & Entrepreneurship (project nr. HBC.2021.0673) and/or Innoviris.

978-3-903176-59-1 ©2023 IFIP

design novel algorithms that achieve similar performance as the ones developed for single-user workload scenarios but are tailored specifically to handle the multi-user complexities.

In this paper, we present a solution to the auto-scaling problem by designing a RL-based controller that extends the capacity of the state-of-the-art RL-based auto-scalers to the multi-user workload scenario. The proposed controller can learn diverse user demands and optimize resource allocation simultaneously. This approach goes beyond state-of-the-art since a) it removes the need of having multiple single-user controllers for each type of workload, which have to be also tailored to each type of workload or adapted to work sub-optimal on all of them, and b) it achieves a good performance in terms of peak latency while bounding the number of used resources in multi-user workload scenarios.

II. System Model

Auto-scaling is a technique used in cloud and edge computing to dynamically adjust computational resources in response to workload changes. It aims to optimize resource allocation by either increasing or decreasing computing resources like CPU and memory (vertical scaling) or by adjusting the number of servers/micro-services (horizontal scaling) for functions, services, or applications. Auto-scaling controllers typically target various objectives, such as reducing application delays, improving resilience, or load balancing. However, this paper focuses on two primary objectives: meeting Key Performance Indicators (KPIs) specified in SLA agreements and avoiding over-provisioning to optimize resource utilization and minimize costs, especially in edge DCs, where resources are limited and cost considerations are crucial [9].

In the last years, RL-based algorithms have been proposed to solve the auto-scaling problem with outperforming results compared to traditional methods such as expert and ruled-base methods and control theoretic approaches in single user's workload setups [3], [6], [10], [11]. These results have demonstrated that RL-based auto-scalers can learn the allocation of computing resources directly from complex workload patterns without needing expert knowledge or complex and time-consuming engineering design procedures. However, the traditional approach of using single-user scalers in parallel to dynamically allocate and distribute resources in a multi-user setup can not ensure a fair and efficient utilization of resources while adapting to fluctuations in demand as their decisions are based on only local information, missing the side-effects of other scalers making decisions over a shared infrastructure.

In order to design RL-based auto-scalers that can manage the resource allocation of multi-user workload simultaneously, we first need to extend the single-user's workload Markov Decision Process (MDP) framework to cover the multi-user workload setup. An MDP is defined by a discrete-time stochastic framework for modeling decision-making problems that can be solved by algorithms such as RL. This process is defined by a tuple $(\mathcal{S}, \mathcal{A}, p, r)$ where $\mathcal{S}$ is a set of states, $\mathcal{A}$ is a set of actions, p is the transition probability between states s and s' after action a is taken, and r is the immediate reward obtained after acting with a in state s. The policy, defined as π, is a mapping function from states to actions. The solution to an MDP is an optimal policy that maximizes the expected long-term reward (discounted sum of immediate rewards). To solve the MDP, several tools can be employed, RL being one of those. The optimal policy is found in RL after many agent interactions with the environment.

For the auto-scaling problem, the state can be defined using the information retrieved from the computing infrastructure. More precisely, at any time step t, and set U of users, a system state $s^{(t)}$ can be defined as:

1) Mean CPU usage c_u^t among the active CNFs associated to a given user $u \in U$.
2) Peak (maximum) latency d_u^t from the active CNFs associated to a given user u.
3) The number of active CNFs n_u^t associated to a given user $u \in U$.

Based on this information, the RL agent decides if the number of CNF instances of each user must be increased, decreased, or kept the same. In other words, the agent needs to take a discrete action a_u^t per user u, given the state $s(t)$ and CNF traffic a_u^t. This specific action definition is a multi-discrete action space, which refers to a RL scenario where an agent has to select multiple actions simultaneously from separate action dimensions.

Finally, and without loss of generalization, in this article, we assume that the SLA associated with the user workload is the same for all users; this allows us to serve all users with identical type of CNF, the number of CNF will be determined by each user workload patterns.

The reward function is defined similarly as in [3] but extended to the multi-user workload setup. Our agent takes multi-discrete actions to maintain a given continuous variable (e.g., latency) at a certain level while controlling the number of CNF instantiated per user. Consequently, the agent is rewarded if the actions lead towards that goal. More specifically, the reward function at time step t is defined as

$$r^{(t)} = \begin{cases} 1/3 & |d_u^{(t)} - d_{tgt}| < \epsilon \cdot d_{tgt} \vee \\ & |cpu_u^{(t)} - cpu_{tgt}| < \epsilon \cdot cpu_{tgt}, \forall u \in U \\ 0 & |d_u^{(t)} - d_{tgt}| \geq \epsilon \cdot d_{tgt} \vee \\ & |cpu_u^{(t)} - cpu_{tgt}| \geq \epsilon \cdot cpu_{tgt} \\ -100 & \text{in episode termination cases} \end{cases} \quad (1)$$

where $d_u^{(t)}$ is the peak latency from the active CNFs at time step t for user u (taken from the system state), d_{tgt} is the target latency as defined by the SLA and ϵ is a range of tolerance allowed by the SLA (e.g., 20% above or below the target value). Notice that the reward function considers all the users equally important; therefore, if the computing resource is well optimized, the reward sum equals one. Also, similar to [3], the reward considers both the peak latency and CPU usage to trade them off; otherwise, the agent will take the most obvious action: to keep increasing the number of CNF instances, disregarding the economic impact of such a decision, or

maintain the CPU in a range reducing it to a threshold-based algorithm. Finally, the agent is highly penalized if it incurs a termination situation, e.g., very high latency or the number of CNF instances surpasses the maximum capacity of the infrastructure (see Section III).

Traditionally, RL algorithms fall into two categories based on how they determine the optimal policy [12]. Action-value methods learn action values to make action selections, while policy gradient approaches directly acquire a parameterized policy for action decisions, bypassing action-value estimates. While action-value RL algorithms offer the advantage of learning both the optimal policy and value function simultaneously, they face challenges in environments with large or continuous action spaces, requiring explicit estimation of action values for all state-action pairs. In contrast, policy gradient methods stand out in handling extensive and continuous action spaces by directly optimizing the policy function but may encounter high gradient estimate variance, affecting learning stability.

As described in the MDP model, given the multi-discrete action space of the problem and continuous state space, policy gradients are better suited. In this paper, we have selected the policy-gradient Proximal Policy Optimization (PPO) algorithm, which combines some of the performance improvements that have been introduced by Advantage Actor Critic (A2C), such as having multiple workers, with a trust region to improve the learning of the actor component from Trust Region Policy Optimization (TRPO) [13]. In general, the main idea of PPO is that after an update, the new policy should not be too far from the old policy. For that, PPO uses clipping to avoid too large updates. PPO agents have demonstrated a good balance between sample complexity, ease of implementation, computational cost, and performance [13].

III. Experimental Setup and Evaluations

A. Experimental scenario

Our base computing resource scenario comprises two servers, three users, and three dedicated load balancers (one for each user), as depicted in Figure 1. Each CNF instance requests resources from the servers based on the workload to be processed while ensuring that the specified thread limit is not exceeded. As explained in Section II, a user workload in our system model can represent a set of computing jobs or aggregated traffic to be processed by a computing unit, where each workload has a specific pattern. We use DynamicSim, a simulator that enables the creation of edge-cloud network scenarios and provides several metrics such as CPU usage and *peak latency*. Notice that DynamicSim tracks the latency of the jobs and reports the maximum of those latencies per tick as peak latency. More details about the simulation platform can be found in [3]. The parameters used to constrain the servers operation and the CNF are described next. Each server can execute up to 20 CNF instances with up to 16GB of RAM. On the other hand, a CNF instance can process a maximum of 300 jobs per tick (i.e., every discrete time step in the simulator). In addition, the expected performance, i.e., SLA, is predetermined between well-defined boundaries.

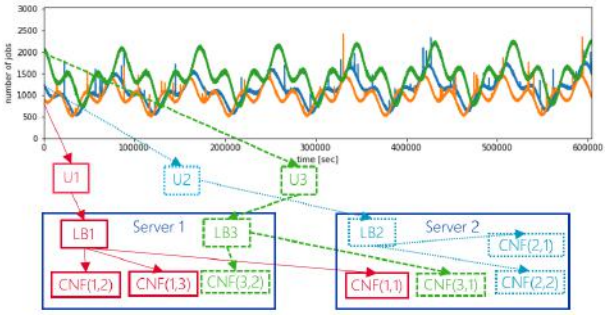

Fig. 1. Scenario used for the evaluation which is composed of three users' workloads and two servers.

First, a CPU consumption of 75% provides a target value to avoid the non-linear energy consumption above this level [14] and a latency threshold of 20ms to process the jobs, which is a value expected in processing cloud-based interactive streaming content [15], with an accepted deviation margin for both measures set at 20%. Notice also that instantiating new CNF instances is facilitated through load balancers, which use the least-load distribution strategy to allocate CNF instances across servers.

In terms of the workload generation, the traces used to generate the workloads are based on the facebooklive18 dataset [16], which periodically fetched Facebook live video broadcasts and viewers metadata, such as geo-location and video resolution, using the APIs provided by Facebook (which were disabled in 2019). Specifically, data was captured in consecutive intervals of 300sec, and the dataset is available during several days in May, June, and July of 2018 for all geographic regions (Europe, Asia, North America, South America, Asia, Africa, and Oceania). To use such metadata for our purpose, we assume that the work (i.e., number of jobs) the Facebook live servers do is proportional to the number of live streams in nearby geographic locations and their video resolution. This assumption makes sense for live streams as they rely on low or even zero-latency video encoding using simple devices. In contrast, further video compression/post-processing can be done at the edge server with more computing resources.

Moreover, we extracted seven consecutive days from three geographic regions (Europe, North America, and South America) and applied cubic spline interpolation (with some added Gaussian noise) to increase the granularity to 1sec. For the different implemented algorithms, the first five days of the trace can be used as training data to find/learn the controller parameters, and the last two days are used for testing/validation. This strategy is very well known in the ML community to evaluate the performance of the algorithms under unseen input data (generalization).

B. Implemented algorithms

In order to evaluate the performance of the RL agent designed in the previous section, we implement a PPO-based controller to realize it together with two more controllers as baselines: one based on rules and one based on control theory. The details of their implementation are below:

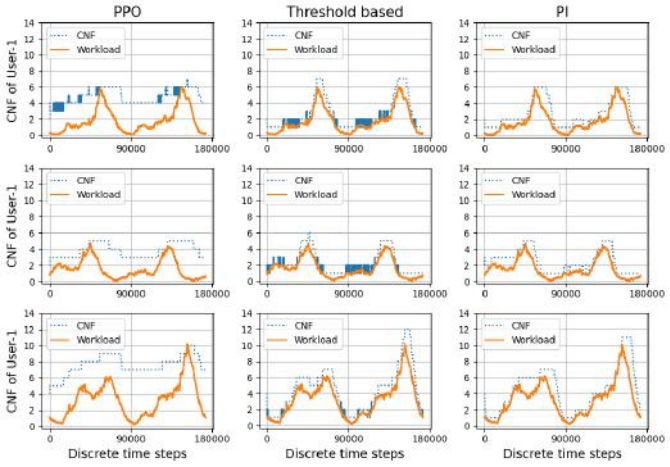

Fig. 2. Performance of the three proposed agents in terms of the number of created CNFs and peak latency, together with the actual users' workloads.

TABLE I
COMPARISON OF THE CNFS AND PEAK LATENCY PERFORMANCE

Metric	Method	Users	Mean	Std	min	Max
#CNF	PPO	User 1	4.76	0.93	2	7
		User 2	3.82	0.86	2	5
		User 3	7.59	1.24	4	10
	THD	User 1	**2.59**	1.94	1	7
		User 2	**2.28**	1.46	1	6
		User 3	4.40	2.82	1	12
	PID	User 1	2.82	1.37	2	6
		User 2	2.55	0.96	2	5
		User 3	**4.28**	2.35	2	11
Peak Latency	PPO	User 1	**0.0064**	0.0030	0.0033	0.1377
		User 2	**0.0076**	0.0153	0.0038	0.4819
		User 3	0.0154	0.0865	0.0041	1.0957
	THD	User 1	0.0082	0.0023	0.0033	0.0267
		User 2	0.0083	0.0018	0.0043	0.0243
		User 3	0.0086	0.0011	0.0056	0.0195
	PID	User 1	0.0076	0.0018	0.0033	0.0228
		User 2	0.0077	0.0017	0.0033	0.0223
		User 3	**0.0085**	0.0015	0.0033	0.0234

Threshold based controller: Threshold (THD)-based resource allocation in a multi-client scenario involves using reactive scalers, one per user. They employ THD-based rules, which rely on the observed performance metric (e.g., service latency) to execute the pre-set scaling actions (e.g., increase, decrease, nothing). For our implementation, we use CPU usage and peak latency as metrics to monitor and define the thresholds, similar to the RL agent reward function and fully aligned to the objectives to optimize. More precisely, the latency thresholds are set at 24ms (upper) and 16ms (lower), and CPU thresholds are set at 90% (upper) and 60% (lower). An action is triggered if a monitored metric goes above or below any of the two thresholds. After taking action, the system continues to monitor the system metrics. If the action was adequate, the metrics should return to acceptable levels. If not, further action may be needed.

Proportional–Integral (PI)-based controller: A PI controller is commonly used for auto-scaling in various systems, including cloud computing, industrial processes, and network management. In our implementation, the PI controller uses the current d_u^t and previous d_u^{t-1} peak latency to decide how to set the number of CNF instances. In particular, it keeps track of a variable ϕ_u^t at time step t:

$$\begin{aligned} &\text{if } d_u^t > \theta_U : \phi_u^{t+1} = \phi_u^t + \alpha\left(d_u^t - \theta_U\right) + \beta\left(d_u^t - d_u^{t-1}\right) \\ &\text{if } d_u^t < \theta_L : \phi_u^{t+1} = \phi_u^t + \alpha\left(d_u^t - \theta_L\right) + \beta\left(d_u^t - d_u^{t-1}\right) \end{aligned} \quad (2)$$

with θ_U, θ_L, α and β tunable parameters. Notice that if d_u^t lies between the two thresholds, i.e., θ_L and θ_U, referred to as the dead zone, ϕ_u^t is not updated. If, at the beginning of time step $t+1$,

- if ϕ_u^{t+1} exceeds the the number of CNF instances by more than 1, that number of CNF instances is increased by 1,
- if ϕ_u^{t+1} subceeds the number of CNF instances by more than 1, that number of CNF is decreased by 1,
- otherwise the number of CNF instances is kept the same.

The integral and proportional elements are represented by the second and third components in equation (2), correspondingly. The integral element aims to maintain the maximum delay close to the upper threshold if it exceeds that threshold and close to the lower threshold if it succeeds that threshold. In contrast, the proportional component is designed to actively respond to changing trends in latency progression. If the latency lies in the dead zone between the two thresholds, the PI-based controller does not react. Note that the PI controller only requires the maximum latency (both current and past values) for its input and does not require information about CPU utilization. The optimal values for its parameters, θ_U, θ_L, α, and β, are usually established through several trial runs on training data. In the experiments in Section III-C, we found $\theta_U = 0.012$ (sec), $\theta_L = 0.007$ (sec), $\alpha = 10$ (1/sec) and $\beta = 300$ (1/sec) by running an extensive set of tests with different parameters on the training set (of two days).

PPO-based controller: In this paper, we use the PPO implementation provided by Stable Baselines[1] with its default parameters since a) this is a popular and widely-used library that provides a collection of robust and reliable RL algorithms to provide benchmarks for reproducibility, and b) it offers the implementation of state-of-the-art algorithms, such as PPO, which have been extensively tested and optimized. We trained the RL-based controller using 1, 2, 3, and 4 days of workload traces. After testing each one of them, we observed that the agent trained for one day yielded superior results, and we selected it for the performance evaluation.

C. Performance evaluation results

Figure 2 shows the distribution of different user workloads and, together with Table I, how each controller can manage them in terms of the number of created CNFs and peak latency. We can see that THD restricts the creation of CNF more effectively in relation to the workloads compared to the other two algorithms, providing the best performance in terms of created CNFs. This feature makes THD a suitable candidate for scenarios where cost savings, energy efficiency,

[1]https://stable-baselines3.readthedocs.io/en/master/

and more efficient resource utilization are the most critical priority. Moreover, it is also suitable when there are computing constraints to run the controller algorithm.

Results also reveal that the PPO is the controller with the lowest performance in the number of created CNFs, on average, although their maximum values are better than the other two. However, the proposed PPO algorithm outperforms the other two algorithms when considering latency, which could be particularly beneficial in real-time applications where low latency is crucial, such as high-frequency trading, video streaming, or online gaming. Of course, having a lower latency directly results from having a more significant number of CNFs than required (over-provisioning), but the over-provisioning is still bounded. The performance of the PPO controller also demonstrates that extending the RL agent to a multi-user workload setup is more challenging than its single-user counterpart. Some hypotheses of this performance can be related to the multi-discrete action spaces of our MDP, which impacts exploration and credit assignment problems. More precisely:

Exploration: Multi-discrete action spaces tend to have a more significant number of possible action combinations compared to simple action spaces, resulting in the agent exploring a more extensive set of combinations, which can lead to slower learning or convergence to suboptimal policies.

Sample efficiency: Learning in multi-discrete action spaces often requires significantly more samples (i.e., interactions with the environment) than single action spaces. This also makes it challenging to gather enough diverse experiences to effectively train the agent.

Credit assignment: Determining which actions led to the obtained rewards, e.g., the assigning credit problem, is more difficult in multi-discrete action spaces since it may not be straightforward to identify which specific combinations were responsible for the observed outcomes, making it harder for the agent to learn from its experiences.

The PI algorithm, while not excelling in either latency or CNF creation, offers a balanced performance that might be preferable in certain situations. PI can be a viable choice when neither latency nor CNF creation is the sole deciding factor, but rather a combination of both. Moreover, PI controllers are well known for their simplicity and robustness when deployed, making them a good choice for systems where simplicity and reliability are important, and the resources to run the algorithm are constrained. However, their time-consuming design and lack of adaptability when workloads change drastically may limit their usability.

IV. Conclusions and Future Work

In this paper, we proposed a RL-based controller that extends the capability of state-of-the-art RL-based auto-scalers to the multi-user workload scenario and compared its performance with two well-known controllers, one THD-based and one PI-based, which have been adapted to run in this setup. Although our RL-based controller did not outperform the other two in all evaluations, the THD-based controller outperformed the others in terms of the lower number of CNFs and the PI-based one trading off both CNFs and peak latency, it did provide better latency control while bounding better the maximum number of created CNFs. However, some deficiencies need to be studied further. First, it is necessary to investigate how to design a reward function for a MDPs that strikes a better balance between optimization objectives. Second, this paper did not consider complex scenarios, such as distributed and federated domains. In these cases, it is required to adapt our algorithm to enable multi-agent and federated operation by combining federated learning techniques with multi-agent RL, aiming for more robust and scalable resource allocation in such scenarios.

References

[1] E. A. Mazied, D. S. Nikolopoulos, Y. Hanafy *et al.*, "Auto-scaling edge cloud for network slicing," *Frontiers in High Performance Computing*, vol. 1, p. 1167162, 2023.

[2] T. Lorido-Botran, J. Miguel-Alonso, and J. A. Lozano, "A review of auto-scaling techniques for elastic applications in cloud environments," *Journal of grid computing*, vol. 12, no. 4, pp. 559–592, 2014.

[3] P. Soto, D. De Vleeschauwer, M. Camelo *et al.*, "Towards autonomous vnf auto-scaling using deep reinforcement learning," in *2021 Eighth International Conference on Software Defined Systems (SDS)*, 2021, pp. 01–08.

[4] J. F. Nunes Pinheiro, C.-Y. Chang, T. Collins *et al.*, "5geco: A cross-domain intelligent neutral host architecture for 5g and beyond," in *IEEE INFOCOM 2023 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2023, pp. 1–6.

[5] T. Chen, R. Bahsoon, and X. Yao, "A survey and taxonomy of self-aware and self-adaptive cloud autoscaling systems," *arXiv preprint arXiv:1609.03590*, 2016.

[6] J. Santos, T. Wauters, B. Volckaert *et al.*, "gym-hpa: Efficient auto-scaling via reinforcement learning for complex microservice-based applications in kubernetes," in *NOMS 2023-2023 IEEE/IFIP Network Operations and Management Symposium*, 2023, pp. 1–9.

[7] S. K. Garg, S. K. Gopalaiyengar, and R. Buyya, "Sla-based resource provisioning for heterogeneous workloads in a virtualized cloud datacenter," in *International conference on Algorithms and architectures for parallel processing*. Springer, 2011, pp. 371–384.

[8] Y. Tan, F. Wu, Q. Wu *et al.*, "Resource stealing: a resource multiplexing method for mix workloads in cloud system," *The Journal of Supercomputing*, vol. 75, pp. 33–49, 2019.

[9] F. Haouari, E. Baccour, A. Erbad *et al.*, "Qoe-aware resource allocation for crowdsourced live streaming: A machine learning approach," in *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, 2019, pp. 1–6.

[10] P. Soto, M. Camelo, D. De Vleeschauwer *et al.*, "Network intelligence for nfv scaling in closed-loop architectures," *IEEE Communications Magazine*, vol. 61, no. 6, pp. 66–72, 2023.

[11] J. Santos, T. Wauters, B. Volckaert *et al.*, "Towards low-latency service delivery in a continuum of virtual resources: State-of-the-art and research directions," *IEEE Communications Surveys and Tutorials*, vol. 23, no. 4, pp. 2557–2589, 2021.

[12] R. Sutton and A. Barto, "Reinforcement learning: An introduction," *IEEE Transactions on Neural Networks*, vol. 9, no. 5, pp. 1054–1054, 1998.

[13] X. Wang, S. Wang, X. Liang *et al.*, "Deep reinforcement learning: A survey," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–15, 2022.

[14] F. Armenta-Cano, A. Tchernykh, J. M. Cortés-Mendoza *et al.*, "Heterogeneous job consolidation for power aware scheduling with quality of service," in *Russian Supercomputing Days 2015*, 2015, pp. 687–697.

[15] R. Sharp, "Latency in cloud-based interactive streaming content," *Bell Labs Technical Journal*, vol. 17, no. 2, pp. 67–80, 2012.

[16] E. Baccour, A. Erbad, K. Bilal *et al.*, "Facebookvideolive18: A live video streaming dataset for streams metadata and online viewers locations," in *2020 IEEE International Conference on Informatics, IoT, and Enabling Technologies (ICIoT)*, 2020, pp. 476–483.