Optimal Resource Allocation in SDN/NFV-Enabled Networks via Deep Reinforcement Learning

Jing Su, Suku Nair
AT&T Center for Virtualization
Southern Methodist University
Dallas, USA
{suj, nair}@smu.edu

Leo Popokh
CMS
Hewlett Packard Enterprise
Dallas, USA
leonid.i.popokh@hpe.com

***Abstract*—Software-Defined Networking (SDN) and Network Functions Virtualization (NFV) are two emerging paradigms that enable the feasible and scalable deployment of Virtual Network Functions (VNFs) in commercial-off-the-shelf (COTS) devices, which deliver a range of network services with reduced cost. The deployment of these services requires efficient resource allocation that fulfills the requirements in terms of Quality of Service (QoS) and Service-Level Agreement (SLA) while considering the constraints of the underlying infrastructure, such as maximum latency tolerance and affinity policies. To address this issue, we study the resource allocation problem in SDN/NFV-enabled networks, which involves numerous optimization variables resulting from the multidimensional space of system component parameters and states. Using deep reinforcement learning, we propose a policy gradient-based algorithm with an invalid action masking approach to efficiently tackle the resources allocation problem while handling system constraints in industrial settings. The simulation results unequivocally show the effectiveness and performance of the proposed learning approach for this category of problems.**

***Index Terms*—Resource Allocation, NFV, VNF Placement, Deep Reinforcement Learning**

I. Introduction

With the introduction of mobile edge computing (MEC), the core components of cloud has been moved to the mobile edge network in order to reduce resource demands and user experience delays. In the edge network that virtualizes a cloud-like environment with computing, storage, and networking resources, the users can distribute requested content or virtual network functions (VNFs) across adjacent clouds. This method mitigates the deployment and maintenance challenges for conventional data centers, enables fast service introduction, and decreases operating costs. Network Function Virtualization (NFV) and Software-Defined Networking (SDN) proposed by several leading industrial telecommunication operators in 2012 [1] enable the feasible and scalable deployment of VNFs in commercial-off-the-shelf (COTS) devices. NFV allows network operators to accelerate the deployment of new virtual devices in a more dynamic and automated approach that has the potential to lead to reductions in operating expenses and capital expenses [2]. It enables virtualization of network functions that can be deployed as virtual machines on general purpose server hardware in cloud environments, which effectively reduces costs in deployment and operational aspects [3]. NFV focuses on using COTS hardware, i.e., general-purpose servers, for the substrate NFVI network to dynamically deploy VNFs on-demand on the Virtualized Infrastructure Management (VIM) layer and facilitate the migration of important infrastructure services, such as the Evolved Packet Core (EPC), to NFV Infrastructure (NFVI).

The combination of SDN, NFV and MEC technologies are widely considered to be essential in the development of 5G [4]. To reap the benefits of SDN, NFV, and MEC, VNFs must be provisioned with sufficient resources on edge servers without compromising network quality of service (QoS) and Service-Level Agreement (SLA). An optimized resource allocation result can benefit the network in various aspects, such as energy-saving, performance boost, and latency reduction. Resource allocation optimization is widely recognized as an NP-hard problem [5]–[7] in the telecommunication operation and maintenance area, which is nontrivial to address in a complex network as it involves a vast number of optimization variables resulting from the multidimensional space of network component parameters and states. Accordingly, determining the optimal resource allocation is an important and challenging problem to examine in SDN/NFV-enabled networks.

Machine learning (ML) has attracted considerable attention from researchers in recent years due to its capacity for large-scale data processing and intelligent decision-making. Reinforcement Learning (RL) is a machine learning technique for the process of making informed decisions by learning from experience. However, in the traditional tabular-based RL algorithms, the maintenance overhead of discrete $(state, action)$ to reward information Q table limited the dimensionality of the target problem. The introduction of deep neural networks made deep reinforcement learning (DRL) possible to process high-dimensional inputs without the need for hand-crafted feature representations. Deep reinforcement learning has apparent advantages in resource allocation, which can realize a speedy execution time than search-based approaches and thoroughly learn the problem structure while expanding the learning capacity. The trained models are effortless to use and deploy in a distributed manner, and the inference process can be accelerated by existing hardware (i.e., graphics card).

In this context, this paper presents a novel approach for efficient resource allocation optimization utilizing a policy gradient-based DRL agent and NFVI simulation DRL environment. The paper thus aims to contribute to a general and practice-oriented scheme for resource allocation in SDN/NFV-enabled networks.

The contribution in this paper can be summarized as follows:

1) We model the resource allocation problem in SDN/NFV-enabled networks as Markov decision process (MDP) and utilize DRL to solve it.
2) We propose a novel approach for the DRL agent to categorize and handle the constraints that are predominantly present in industrial settings.
3) We conduct a theoretical analysis of the agent algorithm on the DRL environment and further construct a PyTorch-based environment to evaluate its performance. The simulation results demonstrate that our approach outperforms the search-based constraint programming method implemented on Google OR-Tools [8] in terms of resource efficiency and processing time.

The remainder of this article is organized as follows. Section II addresses the related work. In Section III and IV, the proposed approach and the DRL implementation are described, respectively. Section V presents simulation results and analyzes the performance of the proposed approach. Finally, Section VI concludes this article and discusses future research.

II. Related Work

Resource allocation optimization has been extensively studied within the framework of SDN/NFV-enabled networks [9]–[13]. Due to their importance for telecommunication operators, VM resource allocation and VNF deployment are well-studied and researched problems in the literature and industry. This section reviews three prevalent approaches to optimize resource allocation and VNF deployments.

A. Search-based Approaches

Bari *et al.* (2015) [9] introduced an Integer Linear Programming (ILP) model for VNF orchestration. The model is solved to determine the optimal number of VNFs and place them at the optimal locations to optimize network operational cost and resource utilization. The ILP model-based solution is suitable for small networks; however, more extensive networks' heuristics may not resemble large production networks.

Harutyunyan *et al.* (2019) [14] proposed ILP techniques to formulate a joint Service Function Chain (SFC) placement, user association, and resource allocation problem that took end-to-end latency and data rate constraints into account. The authors then developed a heuristic to overcome the scalability issue of ILP-based solutions. Our work is flexible and scalable to adapt to various situations that better meet the various challenges of cloud computing and fog computing environments.

B. Dynamic Programming

Ghribi *et al.* (2016) [10] utilized a dynamic programming-based algorithm to solve the VNF placement problem. We find it is hard to implement constraints and need many times of backtracking for the optimal result and is therefore inefficient in complex networks.

Forootani *et al.* (2021) [15] employed a stochastic dynamic programming method to solve the resource allocation problem. Nevertheless, only exact dynamic programming approaches were applied and therefore were incapable of extending to a general-purpose MDP framework.

C. End-to-end learning

Wang *et al.* (2018) [16] introduced a k-nearest neighbor (k-NN) supervised learning classifier to solve the resource allocation problem. It has no optimality guarantees and is nontrivial to represent the problem itself.

Xiao *et al.* (2019) [11] presented a policy gradient based adaptive DRL approach to acquire dynamic network state transitions and optimize the deployment of service function chains (SFCs). The authors did not include handling constraints that are broadly present in commercial circumstances. Pei *et al.* (2020) [12] presented a Double Deep Q Network (DDQN) based off-policy DRL algorithm to determine the optimal VNF placement solution. It cannot be directly applied to large action space since it relies on finding the action that maximizes the action-value function, which needs an iterative optimization process at each step in the continuous valued situation since there are no trainable parameters in Q-learning that control probabilities of action.

Sun *et al.* (2021) [17] proposed a data-driven method to combine DRL and Graph Neural Networks (GNN) to solve the VNF placement problem. The authors lack the inclusion of processing delay between the DRL and GNN.

In contrast to the described approaches, our approach models the NFVI environment and virtual resources that satisfy ETSI standards [18] and are capable of being extended to general production environments. We used a Proximal Policy Optimization (PPO) [19] based on-policy DRL algorithm for the large and continuous action space settings. Constraints are widely present in real scenarios and considered hard to implement in end-to-end learning approaches [20]. We utilized an invalid action masking mechanism to handle the hard constraints and introduced a penalty factor to address the soft constraints.

III. Methodology

The following section will discuss the methodology behind our work, including modeling the problem and the proposed solution to solve the VM batch arrival and compute host overload problem.

A. Markov Decision Process

The Markov decision process (MDP) [21] formally describes an environment for reinforcement learning. We model the resource allocation optimization problem in SDN/NFV-enabled networks as an MDP tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ where:

- $\mathcal{S}$: a finite set of NFVI states while the states include a representation of available resources.

- $\mathcal{A}$: a finite set of actions. An action is an allocation decision or a placement act by the DRL agent, e.g., placing a virtual machine (VM) to a compute host (CH).
- $\mathcal{P}$: a state transition probability matrix, the current state depends only on its immediate previous state, $\mathcal{P}_{ss'}^a = \mathbb{P}[S_{t+1} = s' \mid S_t = s, A_t = a]$.
- $\mathcal{R}$ is a reward function, $\mathcal{R}_s^a = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a]$.
- γ is a discount factor, $\gamma \in [0, 1]$.

In the initial state, all of the remaining resources for the current state of NFVI are available. The state transition matrix $\mathcal{P}$ defines the transition probabilities from all states s to all successor states s'. We reward the agent based on the impact of placement actions on resource utilization, and the agent aims to maximize cumulative rewards. The state reward $\mathcal{R}_s$ is the expected reward over all the possible states in which one can transition from state s. In these settings, uncertainty about the future available resources states may not be fully represented. Therefore the discount factor γ is introduced to present future rewards. The return G_t defines the total discounted reward from the time step t, and the final return reward will be multiplied by the discount to avoid infinite returns in cyclic Markov processes as:

$$G_t = \mathcal{R}_{t+1} + \gamma\mathcal{R}_{t+2} + \ldots = \sum_{k=0}^{\infty} \gamma^k \mathcal{R}_{t+k+1} \qquad (1)$$

In general, we implement a decision maker, referred to as an agent, interacts with the NFVI environment $\mathcal{E}$ by selecting placement actions a in response to observations o of the present state s. The set of all potential states (state space) is denoted by $\mathcal{S}$ and can contain either a finite or an infinite number of elements. Similarly, the set of possible placement actions is denoted by $\mathcal{A}$ and is referred to as the action space.

B. *Reinforcement Learning Environment*

We utilize the OpenAI Gym [22] to construct the RL environment. The NFVI environment can be considered to be not constantly changing, since it is a virtual mapping of physical hardware. Thus, the trained agent can be reused to predict the optimal resource allocation and VM placement until a modification to the hardware architecture. Agent training and hardware upgrades can be performed simultaneously to avoid additional service downtime and SLA degradation.

There will be one or several VMs arriving at the time step t. Our approach maintains a VM arrival queue to implement the QoS and SLA provision. Each VM will be assigned a priority attribute to determine its position in the queue. The priority value can be calculated on the basis of the QoS and SLA requirements or the arrival sequence. Fig. 1 depicts the reinforcement learning environment and agent observation. The observation space is defined as a 2-dimensional discrete tuple consisting of CH occupancy and resource availability data, and the resource demand of VM that is at the top of the current arrival queue.

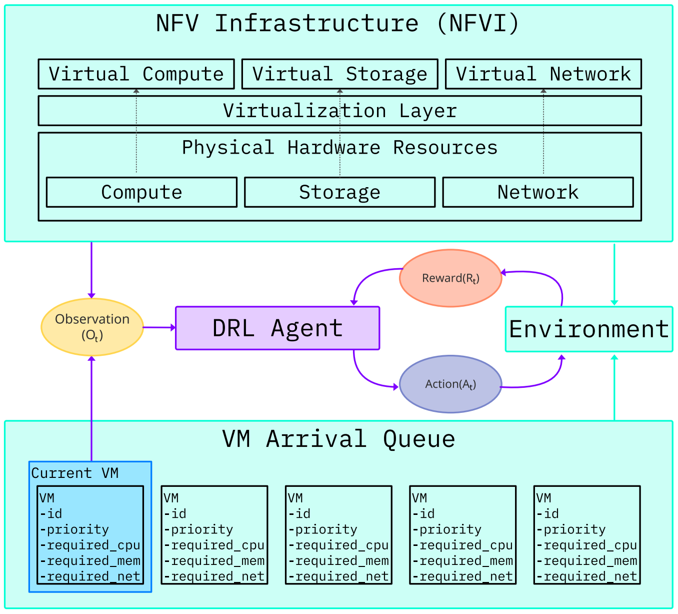

Fig. 1. The RL environment consists of NFVI and VM arrival queue.

C. *Action Space*

An action is an allocation decision or placement act by the agent, and thus the size of the action space depends on the total number of compute hosts. In the NFVI environment, the quantity of compute hosts is a countable integer. In industrial settings, an NFVI often involves multiple data centers, and each data center can have an arbitrary number of compute hosts. Redundant compute hosts for load balance and fault-tolerant purposes commit this number enormously in most cases. In summary, the action space in this problem should be considered large and discrete.

We expect the agent to choose the optimal action based on its observations. Overloading the compute host results in deterioration in application performance, resource shortages, and a significant increase in power usage. To avoid placing a VM to a CH incapable of fulfilling its resource requirement, we use a predefined overload penalty hyperparameter to feedback a considerable negative reward. The agent will learn to avoid overloading the compute host to obtain reward reduction through the training process.

IV. DRL Agent

This section gives an overview of our DRL agent, then discusses the proposed solution to overcome the issues that arise in the resource allocation optimization problem.

A. *Overview*

Existing studies that apply DRL to resource allocation adopt mainly value-based agent algorithms such as Deep Q-Network (DQN) [12], [23], [24]. However, it is likely intractable to successfully train DQN-like networks in this context since DQN relies on finding the action that maximizes the action-value function, and it is challenging to explore large action spaces efficiently. In addition, naive discretization of action spaces needlessly throws away information about the structure of the action domain, which is essential to solve the allocation problem [25]. In contrast, policy-based approaches have the advantages of unbiasedness and stability since they directly optimize the quantity of interest while remaining stable under the function approximation. Their biggest drawback is sample inefficiency, since policy gradients are estimated from rollouts and the variance is often extreme [26]. We adopt a policy-based algorithm PPO to train the agent in our approach and integrate randomizers to mitigate the sample inefficiency problem. PPO is one of the state-of-the-art reinforcement learning approaches and has been widely applied to control tasks [27]–[29]. Fig. 2 demonstrates an overview of the DRL framework and the agent learning procedure. The agent observes the state of the environment and uses Multi-Layer Perceptron (MLP) to parameterize the distribution of decision policies. Invalid actions determined by the environment are masked out from the resulting action value provided by the agent. In this context, invalid action masking helps increase the efficiency of the learning process and enforce hard constraints. The feedback reward from the environment will then deduct the penalty factor based on the impact of the action. This adjusted reward is the ultimate objective that the agent learns while interacting with the environment through multiple trials and errors.

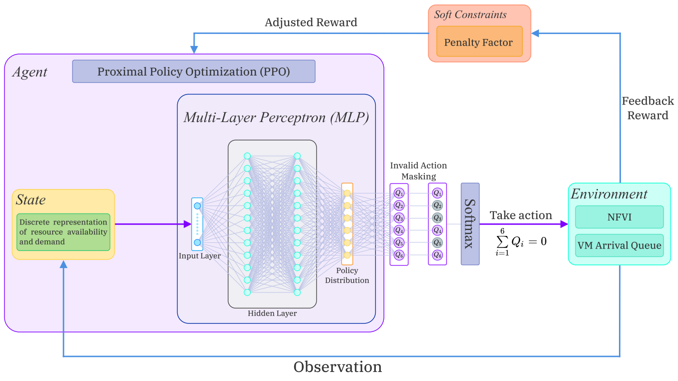

Fig. 2. The DRL framework and agent learning procedure.

B. Invalid Action Masking

To avoid the agent overloading the compute host, we can impose a large negative number as a reward for feedback. However, this method is inefficient in the learning process since the agent needs to learn this hard constraint thoroughly by numerous epochs. Invalid action masking is a common technique implemented to avoid repeatedly generating invalid actions in large discrete action spaces [30]. It is typically performed by substituting a large negative number near $-\infty$ for the actions to be masked. In policy gradient algorithms, actions are sampled based on the probability distribution $a \sim \pi_\theta(\cdot \mid s)$ for policy π_θ at state s while $\pi_\theta = (\cdot \mid s) = \operatorname{softmax}(l(s))$. When we apply the mask, the logits $l(s)$ outputted by policy π_θ given state s associated with the impossible action are at $-\infty$. Let inv_s denote the invalid action masking function given state s, then we have the updated policy gradient for π'_θ:

$$\pi'_\theta(\cdot \mid s_t) = \operatorname{softmax}(\operatorname{inv}_s(l(s))) \tag{2}$$

$$\operatorname{inv}_s(l(s))_i = \begin{cases} l_i & \text{if } a_i \text{ is valid in } s \\ -\infty & \text{otherwise} \end{cases} \tag{3}$$

The logits of the impossible action are replaced with $-\infty$ when the mask is applied. In the implementation, we use the smallest representable number in the data type used. For example, the minimum value for the float32 data type in PyTorch [31] is $-3.4028e + 38$.

C. Action Selection

Exploration and exploitation are two action selection strategies [32]. Exploration finds more information about the environment and selects an action randomly, while exploitation exploits known information to maximize reward and applies a greedy policy towards an optimal selection. When optimizing resource allocation, we value the long-term weight of a state more than the value of an action. i.e., for an incoming entity, it may have many placement probabilities. The action of placing it is less important than the impact of the placement on the available resources that affect the forthcoming entities. To balance the proportions of exploration and exploitation, we implement an ϵ-greedy policy based on equation (4). This policy generates a random number from the range $(0, 1)$ for comparison with the ϵ-greedy value. With a small probability of ϵ, the agent tends to explore and not exploit what it has learned so far.

$$a_t = \begin{cases} \operatorname{rand} A(s), & \text{if } \operatorname{rand}(0,1) < \epsilon \\ \arg\max_a Q(s_t, a), & \text{otherwise.} \end{cases} \tag{4}$$

D. Soft and Hard Constraints

One challenge in adopting RL for the optimization problem is the difficulty in handling constraints that are vastly present in production environments [20]. In the context of SDN/NFV-enabled networks, we categorize constraints into two different types: soft and hard constraints:

Soft constraints: are defined as a less forcing nature of the system, such as placing VMs under the same network service or VNF as close as possible, e.g., in the same data center or even in the same rack. In this case, local optimality is maximized while global optimality is achieved. We introduce a penalty factor to handle soft constraints. We define a set of fine-grained penalty functions $p(C_i)$ for each constraint C_i that in the definition of soft constraints C_s. This function assigns a penalty value based on the satisfaction of the action to the corresponding soft constraint. The penalty factor will be the sum of all penalty function results. As shown in equation (5), the final reward value R'_a for action a is obtained by subtracting the penalty factor from the original reward value R_a.

$$R'_a = R_a - \sum_{i \in C_s} p(C_i) \tag{5}$$

Hard constraints: are defined as a hard limitation that cannot be broken, such as the maximum latency tolerance of a VNF, and affinity policies. For comparison, hard constraints must hold while soft constraints may be violated but as many as possible should be satisfied [33]. Hard constraints are addressed by invalid action masking in our approach. The DRL environment collected all hard constraints and generated the action mask vector for state s_t based on observation O_t at time step t. The action mask vector is a list of binary values that represent the validity of the action a_i in state s and will be accessed in the invalid action masking function inv_s. We can make specific allocation actions impossible to select to achieve hard constraints by manipulating the generation of action masks vector.

E. Agent Training

The PPO-based training procedure is listed in Algorithm 1. The procedure starts by building the environment based on the NFVI configuration and initializing the observation and action space. The hyperparameter N pre-defined the total episodes for training. For each episode, the randomizer will generate resource demand from pre-defined ranges. The environment will collect hard constraints and mask invalid actions for the current state. The agent performs action predictions based on the ϵ-greedy policy.

Algorithm 1: PPO-based Training Procedure

```
Initialization :
    Build environment based on NFVI configuration;
    Initialize observation and action space;
for $episode \leftarrow 1, N$ do
    $t \leftarrow 0$;
    Initialize state $\mathbf{s}_t$;
    Generate resources demand queue randomly;
    repeat
        Mask invalid actions
         $\pi'_\theta(\cdot \mid s_t) = softmax(inv_s(l(s)))$;
        if $rand(0,1) < \epsilon$ then
            $a_t = rand(A(s))$;
        else
            $a_t = \arg\max_a Q(s_t, a)$;
        end
        Take action $\mathbf{a}_t$ according to policy $\pi'(a_t|\mathbf{s}_t;\theta)$;
        Compute reward $R_t$;
        Compute penalty factor $P_{a_t}$ for action $\mathbf{a}_t$;
        $R_t \leftarrow R_t - P_{a_t}$;
        Observe new state $\mathbf{s}_{t+1}$;
        $t \leftarrow t + 1$;
    until $s_t$ is terminal;
end
```

After an action is received, the environment will calculate the reward and penalty factor based on soft constraints for the predicted allocation action. The agent will learn from the penalty-adjusted reward.

V. Simulation Results

This section demonstrates the performance evaluation of our approach in terms of training efficiency, time overhead, and resource efficiency. We used Google Colab to conduct the simulations. Google Colaboratory (also known as Colab) is a research project for prototyping machine learning models and provides a serverless Jupyter notebook environment for interactive development [34]. We constructed a PyTorch-based environment for evaluation using the PyTorch 1.11.0 version [31]. OpenAI Gym [22] is used to construct the RL environment, while Stable-Baselines3 [35] is utlized for agent training. All the simulations are conducted on a Colab notebook with an NVIDIA Tesla V100 GPU.

A. Experimental Setup

1) Agent Training Efficiency: To better illustrate the training efficiency of the proposed approach with invalid action masking (InvPPO), we compare it against the original PPO agent in the same environment and hyperparameters. We conducted 10 separate experiments in different NFVI environments to collect unbiased results.

2) Resource Efficiency and Processing Time: OpenStack [36] is widely regarded as the de-facto open-source cloud management system at the infrastructure as a service (IaaS) layer in the industry [37]. Default scheduler in OpenStack uses a worst-fit algorithm for VM allocation which leaves large fragments of memory in compute nodes [38]. The worst-fit assigns a VM to the lowest partially filled compute host that fits it; otherwise, a new compute host is created [39]. In the older OpenStack versions, VMs are distributed evenly across compute nodes which is the default OpenStack behavior when creating VMs of the same flavor [40]. We call these two strategies *worst-fit* and *evenly* and use them as baselines to evaluate our approach.

In the simulation, we implemented the *worst-fit* and *evenly* algorithms with Google OR-Tools [8] for efficient implementation. The number of search workers for constraints programming (CP) solver is set to 1, and the device for the PyTorch inference is set to *cpu* to prevent multithreading and hardware acceleration advantages. In order to reduce the bias and jitter of the experiment, we set up a fixed NFVI layer and randomly generated VM arrival queue as the experimental environment since NFVI as an infrastructure layer does not usually change during operation. We include 10 compute hosts in the NFVI layer settings and randomly generate their available resources. The maximum capacity of the VM arrival queue is set to 16. For experimental uniformity, we step the CP solver and DRL agent per VM arrival. In this way, the CP solver tackles a sequential arrival allocation problem that fits our approach.

B. Results and Analysis

PPO is a policy gradient algorithm that optimizes policy (process for deciding actions) performance. The policy gradient loss is the mean magnitude of the policy loss function that correlates with the degree to which the policy varies. Fig. 3 displays the aggregated policy gradient loss of the 10 experiments to show the mean and 95% confidence interval over $500,000$ timesteps. It turns out that our approach using invalid action masking (InvPPO) demonstrates low variance and stable convergence compared to the agent using the original PPO algorithm and proven training efficiency.

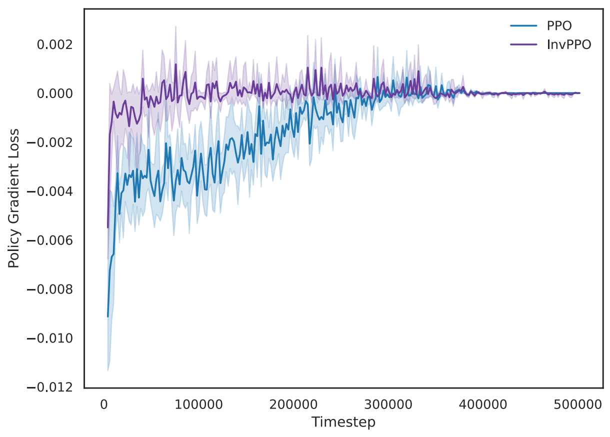

Fig. 3. Policy gradient loss vs. timestep.

We performed the experiment on 1,000 randomly generated VM arrival queues with a random number of VMs. In Fig. 4, we evaluate the resource occupancy in terms of occupied CH and processing time in milliseconds. For the same number of VMs prerequisites, the lower occupied CH number and wall time cost are better. When dealing with fewer VMs, *worst-fit* has a slight advantage because it assigns a VM to the lowest partially filled compute host that fits it. In this case, no new CH allocation is involved, resulting in lower occupied CH. Other than that, our approach (InvPPO) exhibits a lower average resource occupancy than *worst-fit* and *evenly* while delivering an overwhelming fast processing time.

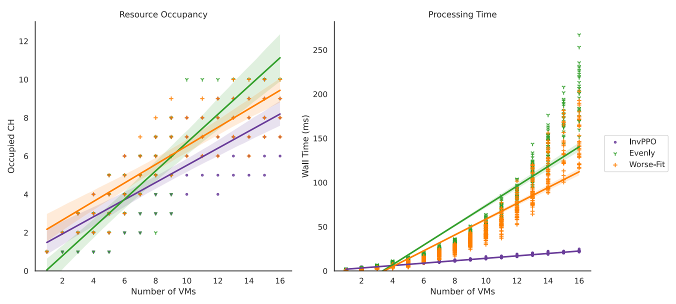

Fig. 4. The resource occupancy and processing time regression model for 1000 executions.

VI. Conclusions and Future Work

In this article, we studied the resource allocation problem in SDN/NFV-enabled networks. In order to solve the problem, we first formulated it as an MDP model aiming to maximize resource efficiency. Then we proposed a novel PPO-based DRL approach to make optimal allocation decisions while addressing the constraints. We have given a detailed analysis of our DRL agent, then constructed a PyTorch-based environment and conducted various simulations to evaluate its performance. The evaluation results show that our approach demonstrates high performance in terms of training efficiency, processing time, and resource efficiency.

For future work, we plan to implement multi-dimensional constraints, such as network topology and data center policy, into our environment and conduct a comprehensive evaluation of latency and power consumption aspects. We plan to study and integrate multiple levels of RL agents. We also plan to combine reinforcement learning and constraints programming for intricate constraints scenarios.

References

[1] J. Matias, J. Garay, N. Toledo, J. Unzilla, and E. Jacob, “Toward an SDN-enabled NFV architecture,” *IEEE Communications Magazine*, vol. 53, no. 4, pp. 187–193, 2015. [Online]. Available: https://dx.doi.org/10.1109/mcom.2015.7081093

[2] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. De Turck, and R. Boutaba, “Network Function Virtualization: State-of-the-Art and Research Challenges,” *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 236–262, 2016. [Online]. Available: https://dx.doi.org/10.1109/comst.2015.2477041

[3] D. B. Oljira, K.-J. Grinnemo, J. Taheri, and A. Brunstrom, “A model for QoS-aware VNF placement and provisioning,” in *2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*. IEEE, 2017. [Online]. Available: https://dx.doi.org/10.1109/nfv-sdn.2017.8169829

[4] N. Kiran, X. Liu, S. Wang, and C. Yin, “VNF Placement and Resource Allocation in SDN/NFV-Enabled MEC Networks,” in *2020 IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*. IEEE, 2020. [Online]. Available: https://dx.doi.org/10.1109/wcncw48565.2020.9124910

[5] F. Bari, S. R. Chowdhury, R. Ahmed, R. Boutaba, and O. C. M. B. Duarte, “Orchestrating Virtualized Network Functions,” *IEEE Transactions on Network and Service Management*, vol. 13, no. 4, pp. 725–739, Dec. 2016.

[6] T.-W. Kuo, B.-H. Liou, K. C.-J. Lin, and M.-J. Tsai, “Deploying chains of virtual network functions: On the relation between link and server usage,” in *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, Apr. 2016, pp. 1–9.

[7] J. Liu, Y. Li, Y. Zhang, L. Su, and D. Jin, “Improve Service Chaining Performance with Optimized Middlebox Placement,” *IEEE Transactions on Services Computing*, vol. 10, no. 4, pp. 560–573, Jul. 2017.

[8] L. Perron and V. Furnon, “OR-Tools,” Google, Apr. 2020. [Online]. Available: https://developers.google.com/optimization/

[9] M. F. Bari, S. R. Chowdhury, R. Ahmed, and R. Boutaba, “On orchestrating virtual network functions,” in *2015 11th International Conference on Network and Service Management (CNSM)*. IEEE, 2015. [Online]. Available: https://dx.doi.org/10.1109/cnsm.2015.7367338

[10] C. Ghribi, M. Mechtri, and D. Zeghlache, “A Dynamic Programming Algorithm for Joint VNF Placement and Chaining,” in *Proceedings of the 2016 ACM Workshop on Cloud-Assisted Networking*, ser. CAN ’16. New York, NY, USA: Association for Computing Machinery, Dec. 2016, pp. 19–24. [Online]. Available: https://doi.org/10.1145/3010079.3010083

[11] Y. Xiao, Q. Zhang, F. Liu, Jia Wang, J. Wang, Jia Wang, Miao Zhao, M. Zhao, Z. Zhang, and J. Zhang, “NFVdeep: Adaptive online service function chain deployment with deep reinforcement learning,” *Proceedings of the International Symposium on Quality of Service*, pp. 1–10, Jun. 2019.

[12] J. Pei, P. Hong, M. Pan, J. Liu, and J. Zhou, "Optimal VNF Placement via Deep Reinforcement Learning in SDN/NFV-Enabled Networks," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 2, pp. 263–278, 2020. [Online]. Available: https://dx.doi.org/10.1109/jsac.2019.2959181

[13] L. Popokh, J. Su, S. Nair, and E. Olinick, "IllumiCore: Optimization Modeling and Implementation for Efficient VNF Placement," in *2021 International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, Sep. 2021, pp. 1–7.

[14] D. Harutyunyan, N. Shahriar, R. Boutaba, and R. Riggio, "Latency-Aware Service Function Chain Placement in 5G Mobile Networks," in *2019 IEEE Conference on Network Softwarization (NetSoft)*. IEEE, 2019. [Online]. Available: https://dx.doi.org/10.1109/netsoft.2019.8806646

[15] A. Forootani, M. Tipaldi, M. Ghaniee Zarch, D. Liuzza, and L. Glielmo, "Modelling and solving resource allocation problems via a dynamic programming approach," *International Journal of Control*, vol. 94, no. 6, pp. 1544–1555, Jun. 2021. [Online]. Available: https://doi.org/10.1080/00207179.2019.1661521

[16] J.-B. Wang, J. Wang, Y. Wu, J.-Y. Wang, H. Zhu, M. Lin, M. Lin, and J. Wang, "A Machine Learning Framework for Resource Allocation Assisted by Cloud Computing," *IEEE Network*, vol. 32, no. 2, pp. 144–151, Apr. 2018.

[17] P. Sun, J. Lan, J. Li, Z. Guo, and Y. Hu, "Combining Deep Reinforcement Learning With Graph Neural Networks for Optimal VNF Placement," *IEEE Communications Letters*, vol. 25, no. 1, pp. 176–180, 2021. [Online]. Available: https://dx.doi.org/10.1109/lcomm.2020.3025298

[18] L. Popokh, P. Olive, I. Aldama, Y. Al-Doori, and S. Nair, "Physical and Virtual Resources Inventory Modeling for Efficient VNF Placement," in *2020 IEEE 10th International Conference on Consumer Electronics (ICCE-Berlin)*. Berlin, Germany: IEEE, Nov. 2020, pp. 1–6. [Online]. Available: https://ieeexplore.ieee.org/document/9352159/

[19] J. Schulman, Filip Wolski, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," *arXiv: Learning*, Jul. 2017.

[20] Y. Bengio, A. Lodi, and A. Prouvost, "Machine Learning for Combinatorial Optimization: A Methodological Tour d'Horizon," *arXiv:1811.06128 [cs, stat]*, Mar. 2020. [Online]. Available: http://arxiv.org/abs/1811.06128

[21] R. A. Howard, *Dynamic Programming and Markov Processes.*, ser. Dynamic Programming and Markov Processes. Oxford, England: John Wiley, 1960.

[22] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI Gym," *arXiv:1606.01540 [cs]*, Jun. 2016. [Online]. Available: http://arxiv.org/abs/1606.01540

[23] M. Bunyakitanon, X. Vasilakos, R. Nejabati, and D. Simeonidou, "End-to-End Performance-Based Autonomous VNF Placement With Adopted Reinforcement Learning," *IEEE Transactions on Cognitive Communications and Networking*, vol. 6, no. 2, pp. 534–547, Apr. 2020.

[24] W. Mao, L. Wang, J. Zhao, and Y. Xu, "Online Fault-tolerant VNF Chain Placement: A Deep Reinforcement Learning Approach," in *2020 IFIP Networking Conference (Networking)*, Jun. 2020, pp. 163–171.

[25] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv:1509.02971 [cs, stat]*, Jul. 2019. [Online]. Available: http://arxiv.org/abs/1509.02971

[26] O. Nachum, M. Norouzi, K. Xu, and D. Schuurmans, "Bridging the gap between value and policy based reinforcement learning," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17. Red Hook, NY, USA: Curran Associates Inc., Dec. 2017, pp. 2772–2782.

[27] J. Rabault, M. Kuchta, A. Jensen, U. Réglade, and N. Cerardi, "Artificial neural networks trained through deep reinforcement learning discover control strategies for active flow control," *Journal of Fluid Mechanics*, vol. 865, pp. 281–302, Apr. 2019.

[28] E. Bøhn, E. M. Coates, S. Moe, and T. A. Johansen, "Deep Reinforcement Learning Attitude Control of Fixed-Wing UAVs Using Proximal Policy Optimization," in *2019 International Conference on Unmanned Aircraft Systems (ICUAS)*, Jun. 2019, pp. 523–533. [Online]. Available: http://arxiv.org/abs/1911.05478

[29] H. Tang, J. Rabault, A. Kuhnle, Y. Wang, and T. Wang, "Robust active flow control over a range of Reynolds numbers using an artificial neural network trained through deep reinforcement learning," *Physics of Fluids*, vol. 32, no. 5, p. 053605, May 2020. [Online]. Available: https://aip.scitation.org/doi/10.1063/5.0006492

[30] S. Huang and S. Ontañón, "A Closer Look at Invalid Action Masking in Policy Gradient Algorithms," *arXiv:2006.14171 [cs, stat]*, Jun. 2020. [Online]. Available: http://arxiv.org/abs/2006.14171

[31] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[32] M. Coggan, "Exploration and exploitation in reinforcement learning," *Research supervised by Prof. Doina Precup, CRA-W DMP Project at McGill University*, 2004.

[33] M. A. Salido and F. Barber, "How to Classify Hard and Soft Constraints in Non-binary Constraint Satisfaction Problems," in *Research and Development in Intelligent Systems XX*, F. Coenen, A. Preece, and A. Macintosh, Eds. London: Springer, 2004, pp. 213–226.

[34] E. Bisong, "Google Colaboratory," in *Building Machine Learning and Deep Learning Models on Google Cloud Platform: A Comprehensive Guide for Beginners*, E. Bisong, Ed. Berkeley, CA: Apress, 2019, pp. 59–64. [Online]. Available: https://doi.org/10.1007/978-1-4842-4470-8\\_7

[35] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-Baselines3: Reliable Reinforcement Learning Implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021. [Online]. Available: http://jmlr.org/papers/v22/20-1364.html

[36] "Open Source Cloud Computing Infrastructure." [Online]. Available: https://www.openstack.org/

[37] A. Kanso, N. Deixionne, A. Gherbi, and F. F. Moghaddam, "Enhancing OpenStack Fault Tolerance for Provisioning Computing Environments," in *2017 IEEE 18th International Symposium on High Assurance Systems Engineering (HASE)*. IEEE, 2017. [Online]. Available: https://dx.doi.org/10.1109/hase.2017.27

[38] P. K. Prameela, P. Gadagi, R. Gudi, S. Patil, and D. G. Narayan, "Energy-Efficient VM Management in OpenStack-Based Private Cloud," in *Advances in Computing and Network Communications*, ser. Lecture Notes in Electrical Engineering, S. M. Thampi, E. Gelenbe, M. Atiquzzaman, V. Chaudhary, and K.-C. Li, Eds. Singapore: Springer, 2021, pp. 541–556.

[39] F. F. Moges and S. L. Abebe, "Energy-aware VM placement algorithms for the OpenStack Neat consolidation framework," *Journal of Cloud Computing*, vol. 8, no. 1, 2019. [Online]. Available: https://dx.doi.org/10.1186/s13677-019-0126-y

[40] B. Karacali and J. M. Tracey, "Experiences evaluating OpenStack network data plane performance and scalability," in *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2016. [Online]. Available: https://dx.doi.org/10.1109/noms.2016.7502923