PAPER

# Dynamic VNF Scheduling: A Deep Reinforcement Learning Approach

**Zixiao ZHANG**[†], **Fujun HE**[†], ***Nonmembers***, ***and*** **Eiji OKI**[†], ***Fellow***

**SUMMARY** This paper introduces a deep reinforcement learning approach to solve the virtual network function scheduling problem in dynamic scenarios. We formulate an integer linear programming model for the problem in static scenarios. In dynamic scenarios, we define the state, action, and reward to form the learning approach. The learning agents are applied with the asynchronous advantage actor-critic algorithm. We assign a master agent and several worker agents to each network function virtualization node in the problem. The worker agents work in parallel to help the master agent make decision. We compare the introduced approach with existing approaches by applying them in simulated environments. The existing approaches include three greedy approaches, a simulated annealing approach, and an integer linear programming approach. The numerical results show that the introduced deep reinforcement learning approach improves the performance by 6-27% in our examined cases.

***key words:*** *Virtual network function, scheduling, deep reinforcement learning, asynchronous advantage actor-critic*

## 1. Introduction

Software defined networks (SDN) are a novel class of networking solutions in which the traditional paradigm of distributed networking has been replaced with a centralized networking approach where a central controller or a group of controllers make the optimal decisions and the nodes become merely forwarding switches [1]. Network function virtualization (NFV) is a technology that provides the network function made through software based on virtualization technology to a high-performance universal server, separating the existing hardware-dependent network equipment into the hardware part and the software part [2]. NFV brings scalability, flexibility, and rapid development for network functions. A cluster of virtual network functions (VNFs) can serve massive fluctuating traffic flows by distributing packets across multiple instances [3]. According to the characteristics above, SDN-NFV network architecture is regarded as a key framework for processing large amounts of data.

In an SDN-NFV architecture based network, VNFs are combined into service function chains (SFCs) to provide specified services. An SFC defines these VNFs in a certain order, and performs on-demand forwarding and filtering of user traffic [4]. NFV nodes are devices which can process multiple types of VNFs by changing virtual machines (VMs) inside of itself. In a VNF placement problem, the physical resource of NFV nodes are allocated by particular VNFs of given services; the objective is minimizing the total cost of resources in the network. In VNF scheduling, the execution timings of embedded VNFs on NFV nodes are scheduled to minimize the makespan (i.e., the time period from the execution of the first VNF to the completion of last VNF among all the scheduled VNFs for all the services) [5]. The VNF scheduling problem is an essential problem in the construction of SDN-NFV network architecture; an algorithm to solve the scheduling problem affects the the quality of service of the whole network. Existing studies have shown that the classical VNF scheduling problem can be formulated as a job-shop problem [5, 6, 7], which is an NP-hard combinational optimization problem [5].

The existing approaches to obtain the approximate optimal solution of the VNF scheduling problem include approximation algorithms such as greedy approaches and metaheuristic algorithms such as simulated annealing (SA). The approximation approaches are easy to implement, but their performances are less desirable compared with metaheuristic approaches and degrade fast when environment changes. Meanwhile, the metaheuristic approach takes a long time to obtain the result and has an unstable performance in general.

Reinforcement learning (RL) is an essential type of machine learning in which agents find suitable actions to take in an environment in order to maximize a reward [8]. With the development of research in RL, it is applied to solve problems in wide range of areas, including mixed-integer linear programming (MILP) problem formed by VNF scheduling. The work in [5] introduced an RL approach to obtain the approximate optimal solution of VNF scheduling. However, the scenario considered in [5] is static, where the total number of services and the arrival time of each service are given. During the scheduling process, it assumed that there is no new service coming into the system, which can be different from a realistic situation. In a dynamic scenario, the requested services in the system may be different in each time slot. The scheduling plan obtained by the approach in [5] does not consider new coming requests. The scheduling plan can be no longer effective in the next time slot, if we simply repeat the approach in [5] in each time slot for a dynamic scenario. With this defect, the performance of SDN-NFV network cannot be expected when it is applied for such a scenario. Then, a question raises: how can we handle VNF scheduling in a dynamic scenario with RL?

This paper introduces a deep reinforcement learning (DRL) approach to address the VNF scheduling problem in a dynamic scenario. We consider that the requests for different services keep coming during the scheduling process. We describe the VNF scheduling problem in a static scenario, and connect it with the dynamic scenario by modifying the

[†]The authors are with Kyoto University, Kyoto, Japan.

DOI: 10.1587/transcom.E0.B.1

objective of the scheduling problem. Rather than considering the system integrally, we divide the scheduling problem into decision process of each NFV node. Our DRL approach is based on the asynchronous advantage actor-critic (A3C) algorithm. In A3C, the master learning agent is trained with help of multiple worker agents which simultaneously work with each other. The master agent adjusts the policy distribution based on an advantage function. We assign a master agent and an amount of worker agents to each node; the master agent is trained to make decisions on processing which VNF for which request after observing the waiting queue on the node. We define the state, action, and reward to make the learning agents improve their performances. We compare the introduced DRL approach with existing ones in two simulated cases. The numerical results show that the introduced DRL approach improves the performance by 6-27% in different cases.

The rest of the paper is organized as follows. Section 2 introduces previous works that relate to the paper. Section 3 gives a mathematical model description on the static scenario. Section 4 shows the steps of defining the problem in the dynamic scenario and the details of the learning algorithm. Section 5 observes the performances of approaches in different cases. Section 6 summarizes the paper.

## 2. Related work

Similar to our work, several works considered the scheduling problem in dynamic scenarios [11, 12, 13]. The work in [11] introduced an RL method with composite rewards for production scheduling in a smart factory. A smart agent is assumed to make arrangements on the order of jobs to be processed for machines in factory, by observing data from sensors. The problem formulated in this work is a production scheduling problem, while new jobs keep coming. In order to solve such the problem, a Q-learning algorithm was utilized to the agent. During the training, a target network is introduced to improve the efficiency of parameter updating. After taking an action, the reward is composed with four reward parameters, with consideration of makespan, production profits, machine utilization, and workload balance, respectively. The work in [12] considered job-shop production (JSP) problem with dynamic job arrival via DRL. The arrival time and the complete time for each job is recorded. All of the jobs share the same deadline which is related with average workload of machines. The objective is minimizing the sum of the tardiness of jobs. The work in [13] considered on a multi-task workflow scheduling problem with objectives of minimizing the makespan and cost; it is solved by multi-agent RL. This work introduced a directed acyclic graph connecting tasks, so the precedence dependency of tasks can be obtained from graph.

Our work is related to the A3C algorithm. Several works applied the A3C algorithm into combinatorial optimization problems in the communication network field [14, 15]. The work in [14] introduced an A3C algorithm to solve the virtual network embedding (VNE) problem. In an advantage actor-critic (A2C) algorithm, there is an actor network which generates policy and a critic network which estimates the values of states. The two networks cooperate together to indicate how much of an improvement that this action brings to the current state. In A3C, a master-worker construction is used to speed up the training process. Agents are implemented by A2C regardless of master or worker. Worker agents perform training in parallel with each other and share the same parameters of the actor network and the critic network which come from the master agent regularly. The master agent collects the experiences of all workers, adjusts the parameters of both actor network and critic network of itself, and passes the updated parameters to all workers in each collection step. In this way, more experiences are obtained at the same time. The work in [15] considered the routing, modulation, and spectrum placement problem in elastic optical networks (EONs) with the A3C algorithm, where requests of establishing links between two nodes with a specified bandwidth and lasting time keep coming. The system is required to assign a route with suitable frequency slots in the given network topology to each of the request. Multiple parallel actor-learners are used to obtain more abundant and diversified samples. Parameters in deep neural network of each actor-leaner is synchronized with global parameters when a fixed number of requests are served.

In our work, NFV nodes make decisions independently with each other; the role of decision strategy is decisive to the performance of the whole system. Several works introduced effective approaches to make a decision when there are multiple jobs waiting to be processed in a queue [9, 10]. The work in [9] introduced a DRL-based job scheduling model for packet transfer in wide area networks. The work focused on a single sending host, which is required to finish data transfer jobs within their deadlines. In the model, there can be several jobs waiting to be processed, while the sending host can only work on one job each time. So, the host needs to decide the job to be processed in every time slot before transmission. The decision progress is called job scheduling. In order to make the host find the most suitable choice, a policy gradient-based DRL approach has been applied. The thought is recording states of jobs (including generated time, deadline time, total data size, and finished and unfinished data size in terms of time) at the start of each time slot and input them into a deep neural network, so that the neural network can generate the probability distribution of jobs to be processed as a policy. With the predefined reward, policy parameters can be updated by the neural network, so that a more suitable choice of job in each time slot can be given a higher probability to be chosen. The work in [10] considered a downlink scheduling model with a single base station and multiple users. The objective of the system is optimizing throughput while minimizing the average delay. It formed the system as an Markov decision process (MDP) problem with the finite number of states and actions. In the MDP problem, the backlog of network queue for each user and the amount of data arriving to each user is learned in each time

slot. Moreover, the amount of data that can be transmitted over each channel at each time slot is given. The information of queue, data arriving and transmission capacity mentioned above is represented as the system state. The base station is able to serve one user at every time slot, and is required to decide which user is chosen to be served. In addition to this, the base station needs to decide how much data is admitted to arrive into the queue. The decision of user to be served and admitted amount of data arriving into the queue is represented as an action. The queueing dynamics can be represented with queue information in current time slot, a user chosen to be served, transmission capacity for chosen channel and admitted amount of data arriving into the queue. With the conditions above, the work introduced an RL algorithm to obtain an optimal policy.

## 3. Model description

Consider an SDN-NFV architecture-based network system in which the numbers of nodes host a set of VNFs. Let $N$ and $V$ denote a set of nodes and a set of VNFs, respectively. We consider a set of services, each of which is constructed by a number of sequential VNFs. We assume that the placement of VNFs in each service is given.

**Table 1** List of frequently used notations

| Notation | Meaning |
|---|---|
| $N$ | Set of nodes |
| $S$ | Set of services |
| $T$ | Number of total time slots |
| $R$ | Set of requests |
| $D_r$ | Maximum admissible delay of request $r \in R$ |
| $V$ | Set of VNFs |
| $V_r$ | Set of remaining VNFs for request $r \in R$ |
| $v_i^r$ | $i$th VNF in $V_r$ |
| $\rho_{rvn}$ | Number of time slots required to process request $r \in R$ for VNF $v \in V$ on node $n \in N$ |
| $f_{rv}^n$ | Binary parameter indicating whether request $r$ uses VNF $v \in V$ on node $n \in N$ |
| $l_r$ | Binary variable indicating whether delay constraint of $r \in R$ is satisfied |
| $c_r$ | Completion time of request $r \in R$ |
| $h_{rvn}^t$ | Binary variable indicating whether node $n$ starts processing VNF $v \in V$ for request $r \in R$ at time slot $t \in [1, T]$ |
| $R_t$ | Set of unfinished requests at time slot $t$ |
| $D_t^r$ | Maximum admissible delay of request $r \in R_t$ at time slot $t$ |
| $V_t^r$ | Set of remaining VNFs for request $r \in R_t$ |
| $v_t^{ir}$ | $i$th VNF in $V_t^r$ |
| $g_r$ | Coming time of request $r \in R_t$ |
| $\alpha_t$ | Number of new coming requests at time slot $t$ |
| $\omega_t$ | Number of finished requests satisfying the delay constraint at time slot $t$ |

Given a set of requests, each of which belongs to one of the services, we consider the schedule for processing the requests during time slots [1, T]. Let $R$ denote the set of requests. Request $r \in R$ has a maximum admissible delay time of $D_r$ according to the requirement of service which it belongs to. At time slot 1, let $V_r$ denote the set of remaining VNFs for request $r$. Denote $v_i^r$ as the $i$th VNF in $V_r$. Consider $f_{rv}^n$ as a given binary parameter; it equals one if request $r$ uses VNF $v \in V$ placed on node $n \in N$, and zero otherwise. The number of time slots required to process request $r$ for VNF $v$ on node $n$ is denoted as $\rho_{rvn}$. For each request $r \in R$, binary variable $l_r$ indicates whether the maximum admissible delay of $r$ is satisfied. $l_r$ equals one if the delay constraint of $r$ is satisfied, and zero otherwise.

The scheduler decides on which request to be processed for each node at each time slot, and there is no preemption allowed. During processing of each node, the status of node is regarded as unavailable and the node cannot suspend or cancel the current work to process a new work. $h_{rvn}^t$ is a binary variable which equals one if node $n$ starts processing VNF $v$ for request $r$ at time $t$, and zero otherwise. After all VNFs finish processing, request $r$ is given a time $c_r$ as its completion time. The notations frequently used in this paper are shown in Table 1.

The objective of scheduler is maximizing the ratio of the number of requests whose maximum admissible delays are satisfied to the total number of requests in $R$, which is represented as:

$$\max \frac{1}{|R|} \sum_{r \in R} l_r. \tag{1}$$

$l_r$ is obtained by:

$$l_r = \begin{cases} 1, & c_r \leq D_r \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

Each node can only process on one request at the same time, which is expressed by:

$$\begin{aligned} &h_{rvn}^t + \sum_{v' \in V} \sum_{r' \in R: r' \neq r} h_{r'v'n}^{t'} \leq 1, \forall r \in R, v \in V, \\ &n \in N, t \leq t' < \min(T+1, t+\rho_{rvn}), t \in [1, T]. \end{aligned} \tag{3}$$

Equation (3) indicates that, if node $n$ starts to process VNF $v$ for request $r$ at time slot $t$, or $h_{rvn}^t = 1$, the node cannot start to process any other VNFs for the next $\rho_{rvn}$ time slots.

The start time of VNF for any request cannot be earlier than the finish time of previous VNF according to the service which the request belongs to, which is expressed by:

$$\begin{aligned} &\sum_{n \in N} \sum_{t=1}^{T} \left( h_{rv_{i+1}^r n}^t f_{rv_{i+1}^r}^n - h_{rv_i^r n}^t f_{rv_i^r}^n \right) t \geq \\ &\sum_{n \in N} \sum_{t=1}^{T} h_{rv_i^r n}^t f_{rv_i^r}^n \rho_{rv_i^r n}, \forall r \in R, i \in [1, |V_r| - 1]. \end{aligned} \tag{4}$$

Equation (4) indicates that the difference between the start time of the $i+1$th VNF and that of the $i$th VNF for request $r$ cannot be smaller than $\rho_{rv_i^r n}$ time slots.

Each request has only one start time for each VNF, which is expressed by:

$$\sum_{t \in T} h_{rvn}^t = f_{rv}^n, \forall r \in R, v \in V, n \in N. \tag{5}$$

Equation (5) indicates that, if VNF $v$ of request $r$ is assigned to node $n$, there is one and only one start time for VNF $v$ of request $r$; otherwise, there is no start time for VNF $v$ of request $r$ on node $n$.

To judge whether the delay for request $r \in R$ satisfies the given maximum admissible value, we have:

$$\sum_{n\in N}\sum_{t=1}^{T} h^t_{rv^r_{|V_r|}n} f^n_{rv^r_{|V_r|}} \left(t + \rho_{rv^r_{|V_r|}n}\right) l_r \le D_r, \forall r \in R. \tag{6}$$

Equation (6) indicates that, if the finish time of the last VNF in $V_r$ is no larger than $D_r$, the maximum delay constraint of request $r$ is satisfied and $l_r$ is set to one considering the objective function; otherwise, the constraint is not satisfied and $l_r$ is forced to zero.

In (6), the term $h^t_{rv^r_{|V_r|}n} l_r$ is not linear; we express it in a linear form by introducing a binary variable, $z^t_{rv^r_{|V_r|}n}, r \in R, n \in N, t \in [1,T]$, where $z^t_{rv^r_{|V_r|}n} = h^t_{rv^r_{|V_r|}n} l_r$. The equation of $z^t_{rv^r_{|V_r|}n} = h^t_{rv^r_{|V_r|}n} l_r$ is expressed as:

$$z^t_{rv^r_{|V_r|}n} \le h^t_{rv^r_{|V_r|}n}, \forall r \in R, n \in N, t \in [1,T] \tag{7a}$$

$$z^t_{rv^r_{|V_r|}n} \le l_r, \forall r \in R, n \in N, t \in [1,T] \tag{7b}$$

$$z^t_{rv^r_{|V_r|}n} \ge h^t_{rv^r_{|V_r|}n} + l_r - 1, \forall r \in R, n \in N, t \in [1,T] \tag{7c}$$

$$z^t_{rv^r_{|V_r|}n} \in \{0,1\}, \forall r \in R, n \in N, t \in [1,T]. \tag{7d}$$

Equations (7a)-(7d) indicate that, if both $h^t_{rv^r_{|V_r|}n}$ and $l_r$ equal one, $z^t_{rv^r_{|V_r|}n} = 1$; otherwise, zero.

Equation (6) is transformed to:

$$\sum_{n\in N}\sum_{t=1}^{T} z^t_{rv^r_{|V_r|}n} f^n_{rv^r_{|V_r|}} \left(t + \rho_{rv^r_{|V_r|}n}\right) \le D_r, \forall r \in R. \tag{8}$$

## 4. Dynamic scenario

We consider a dynamic scenario. A set of nodes $N$, a set of VNFs $V$, and a set of service $S$ are given initially. Let $R_t$ represent the set of unfinished requests at time slot $t$. Similarly, let $D_{rn}(t)$ and $V_{rn}(t)$ express the maximum remaining admissible delay and the set of remaining VNFs at time slot $t$ for request $r$ on node $n$, respectively. $R_t$, $D_{rn}(t)$, and $V_{rn}(t)$ are updated at the beginning of each time slot considering the processing procedure and the new coming and finished requests. The number of requests that the system is able to handle at the same time is limited; the system stops receiving new coming requests from buffer when $R_t$ reaches the size limitation $\mathcal{R}$. Let $g_r$ represent the coming time of request $r \in R_t$. Denote $\alpha_t$, $\epsilon_t$ and $\omega_t$ as the number of new coming requests, the number of requests that leave the system, and the number of finished requests satisfying the delay constraint at time slot $t$, respectively. For requests which stay in the system with exceeding their delay constraints at time slot $t$, the system collects the number of them as $\delta_t$.

At the beginning of each time slot, the system observes the state of each request on each node, which includes the information of whether the VNF of the request is waiting to be processed, the remaining processing time of VNF of the request on the node, the remaining VNFs of the request, the remaining admissible delay of the request, and the waiting time of requests on the node. In addition, the system observes the number of new coming requests and the number of finished requests satisfying the delay constraint, which are $\alpha_t$ and $\omega_t$, respectively. The scheduler decides a VNF to be processed for each node in this time slot. The objective of scheduler is maximizing the ratio of the number of requests whose maximum admissible delays are satisfied to the total number of new coming requests along the time, which is given by:

$$\max \frac{\sum_{t=2}^{\infty} \omega_t}{\sum_{t=1}^{\infty} \alpha_t}. \tag{9}$$

### 4.1 State

For each request, it is distributed with a system number of $r \in [1, \mathcal{R}]$ which does not change until it leaves the system. For each node, it records the state of all requests currently associated with it at each time slot. The state of request $r$ on node $n$ is defined as $s_{rn}(t) = [m_{rn}(t), \theta_{rn}(t), V_{rn}(t), D_{rn}(t), W_{rn}(t)]$. The state of a node is represented as $S_n(t) = [s_{1n}(t), \cdots, s_{\mathcal{R}n}(t)]$; the state of the system is represented as $\mathcal{S}(t) = \left[S_1(t), \cdots, S_{|N|}(t)\right]$. $m_{rn}(t)$ is a variable which indicates whether the VNF of request $r$ is waiting to be processed or is being processed on node $n$ at time slot $t$. $m_{rn}(t)$ equals one if the VNF of request $r$ is being processed on node $n$, and two if the VNF of request $r$ is waiting to be processed. $\theta_{rn}(t)$ indicates the remaining processing time of VNF of request $r$ on node $n$ at time slot $t$. If the VNF is waiting to be processed, $\theta_{rn}(t)$ equals $\rho_{rvn}$. $V_{rn}(t)$ is an array which expresses the remaining VNFs of request $r$ and their processing sequence. For example, if there are six types of VNF in total, $V_{rn}(t) = [0,3,1,2,0,0]$ indicates that the first VNF of request $r$ to be processed is $v_3$, then $v_4$ and $v_2$. $D_{rn}(t)$ is the maximum remaining admissible delay of request $r$ at time slot $t$. $D_{rn}(t)$ is calculated by $D_{rn}(t) = D_r - t + g_r$. If the value of $D_{rn}(t)$ is negative, it indicates that the staying time of request $r$ in the system has already exceeded $D_r$ at time slot $t$. $W_{rn}(t)$ is the waiting time of request $r$ on node $n$ at time slot $t$. Denote the coming time of request $r$ on this node as $j_r$. $W_{rn}(t)$ is calculated by the difference between current time $t$ and the coming time of the request on this node, i.e., $W_{rn}(t) = t - j_r$. For a request which does have neither VNF being processed nor VNF waiting to be processed on the node, its state on this node is represented as $[0, 0, [0, \cdots, 0], 0, 0]$.

### 4.2 Action

We define the action of the system as $A(t) = [a_1(t), a_2(t), \cdots, a_{|N|}(t)]$, which is constructed by the action of each node. Action variable $a_n(t)$ is the system number of request $r$, which indicates that node $n$ is scheduled to start to process a VNF of request $r$ from current time slot $t$. For each node, the maximum number of choices is $\mathcal{R}$. Our approach is making each node correspond to one learning agent; different agents work in parallel. In this way, for each step of each agent, the size of action set is $\mathcal{R}$.

### 4.3 State transition

The state transition of requests on nodes depend on progress of VNF processing and actions. $m_r(t)$ changes from zero to one or two when a request comes into the node, according to whether the VNF of this request is scheduled to be processed immediately. When the VNF processing of the request is finished on this node, all of its state variables on this node become zero, including $m_r(t)$. For the request whose VNF is being processed, the value of $\theta_r$ becomes one smaller than previous value after each time slot. When the VNF processing of request $r$ is finished, the corresponding number in $V_r(t)$ becomes zero; the request moves to the next node according to the placement of its next VNF to be processed. $D_t^r$ and $W_t^r$ are calculated directly from $D_r$, $g_r$, $j_r$, and current time $t$.

### 4.4 Reward

The reward of a pair of state and action is calculated from the information of the following state after the action is taken. The reward is separately calculated on each node. To fit the objective of the system, the reward is given as:

$$U(\mathcal{S}(t), A(t)) = \phi_1 \frac{\omega_{t+1}}{\epsilon_{t+1}} + \phi_2 \frac{\sum_{r \in R_{t+1}} \sum_{n \in N} D_{rn}(t+1)}{\sum_{r \in R_{t+1}} D_r} - \phi_3 \frac{\delta_{t+1}}{|R_{t+1}|}, \tag{10}$$

where $\phi_1$, $\phi_2$, and $\phi_3$ are weight coefficients. The first item in (10) is the ratio of the number of requests leaving the system with their delay constraints satisfied to the total number of requests leaving the system after taking action $A(t)$ for state $\mathcal{S}(t)$. It is calculated from the value of $\omega_{t+1}$ and $\epsilon_{t+1}$ observed at the beginning of the next time slot. The ratio of requests satisfying the delay constraint is directly regarded as a reward according to the objective. The second item in (10) is the ratio of the sum of remaining admissible delay of requests to that of the maximum ones, calculated from the values of $D_{rn}(t+1)$ and $D_r$ of each request. For requests whose delay constraints are not exceeded yet, the remaining admissible delay is added as a reward. On the other hand, for requests staying in the system with their delay constraints already been exceeded, the exceeding time which is the absolute value of $D_{rn}(t')$ is subtracted from the reward as a penalty. The third item in (10) is the ratio of the number of requests staying in the system with their delay constraints that have been exceeded to the total number of the remaining requests. It is calculated from the value of $\delta_{t+1}$ and the size of $R_{t+1}$ without considering the new coming requests at time $t+1$, which is directly regarded as a penalty.

The reward becomes higher if the pair of state and action leads the system to a state in which there are as much as possible requests leaving the system with their delay constraints satisfied, while those which do not satisfy its delay constraint have an exceeding time as small as possible, and the ratio of requests that remain in the system with delay constraints that have been exceeded to the total number of remaining requests is reduced. Setting $\phi_1$ to a higher value makes the system try to maximize the ratio of requests which satisfy their delay constraints to those leaving the system, but some of the requests may end with an intolerable delay. Setting $\phi_2$ to a higher value makes the system try to minimize the exceeding time for requests which cannot satisfy the delay constraint, but can lead to a state in which more requests exceed their delay constraints a little. Setting $\phi_3$ to a higher value makes the system try to leave fewer requests that exceed their delay constraints, which can also make some of the requests end with an intolerable delay.

### 4.5 Learning algorithm

We apply the A3C algorithm [14], [15] in our dynamic scenario. Processing decisions of nodes are made in parallel; each of them is generated by a number of learning agents in different types. There is a system server which manages the placement of VNFs for new coming requests and the state transition of nodes. For each node, there is a master agent which is responsible for collecting experiences from worker agents, updating the parameters of neural network and sending them back to worker agents. The number of worker agents for each node is given as $W$. The total number of agents in the system then equals $(W+1)|N|$, with $|N|$ master agents and $W|N|$ worker agents. Moreover, for each agent there is an actor neural network and a critic network equipped; thus the number of networks is $2(W+1)|N|$ in total. At the beginning of each time slot, the master agent sends the information of the state of this node to all worker agents. The parameters in both actor and critic networks are also sent to worker agents. The critic neural networks of master agent and worker agents simultaneously generate a state value of the current time slot. Then worker agents start to take actions in parallel. Worker agents generate the same probability distribution of actions as they share the same actor neural network, while they take actions independently from each other, according to the probability distribution of actions. Each worker agent obtains a new virtual state based on the action that it chooses, which does not consider the new coming requests in the real time. For each worker agent, it generates the state value of the next virtual state independently. Then, the worker agent sends the experience back to

the master agent, which includes the information of current state value, the action that it takes, the reward, and the state value of the next virtual state. The master agent collects the experience of all worker agents.

Based on the experience, the master agent firstly updates the parameters in its actor neural network based on state-action pairs, their rewards, and the corresponding virtual state value of the next time slot. The real action is generated by the updated actor neural network. Let $\theta_{\mathrm{v}}$ denote the parameter of the state value function. $A\left(S(t), a_w(t); \theta_{\mathrm{v}}\right)$ denotes an advantage function, which is expressed as:

$$A\left(S(t), a(t); \theta_{\mathrm{v}}\right) = U(S(t), a(t)) + \gamma V(S(t+1); \theta_{\mathrm{v}}) - V(S(t); \theta_{\mathrm{v}}). \tag{11}$$

Equation (11) indicates the difference between the state value that the action brings, which is $U(S(t), a(t)) + \gamma V(S(t+1); \theta_{\mathrm{v}})$, and that it is estimated, which is represented as $V(S(t); \theta_{\mathrm{v}})$.

Let $\theta$ denote the parameters of the policy. The actor neural network is trained to reach the objective of minimizing the value of the loss function $L_\theta$, which is represented as:

$$L_{\theta,\theta_{\mathrm{v}}}(t) = -\frac{1}{W}\sum_{w=1}^{W} \log p\left(a_w(t)|S(t), \theta\right) A\left(S(t), a_w(t); \theta_{\mathrm{v}}\right), \tag{12}$$

where $a_w(t)$ is the action taken by worker agent $w$ at time slot $t$. To reach the training objective, for the state-action pair which has a value of $A\left(S(t), a(t); \theta_{\mathrm{v}}\right)$ larger than zero, the value of the term $-\log p\left(a(t)|S(t), \theta\right)$ is supposed to decrease. This is performed by raising the probability of choosing this action under the state, which is represented as $p\left(a(t)|S(t), \theta\right)$. If the value of the term $A\left(S(t), a(t); \theta_{\mathrm{v}}\right)$ is smaller than zero, the value of the term $-\log p\left(a(t)|S(t), \theta\right)$ is supposed to increase; it can be performed by decreasing the value of $p\left(a(t)|S(t), \theta\right)$. For the state-action pair which has a value of $A\left(S(t), a(t); \theta_{\mathrm{v}}\right)$ as zero, the value of $p\left(a(t)|S(t), \theta\right)$ stays the same with before.

The actor network distributes the probabilities to available actions based on the state and the policy parameter $\theta$. The policy parameter is updated with the method of policy ascent to reach the training objective based on the loss function and the advantage function, which is represented as:

$$\theta \leftarrow \theta + \eta \nabla_\theta L_{\theta,\theta_{\mathrm{v}}}(t), \tag{13}$$

where $\eta$ is a weight coefficient. The rationale behind the gradient update in (13) is that the policy parameter is updated to optimize the probability distribution of actions for each state to minimize the value of the loss function.

The master agent updates the state value parameter in its critic neural network $\theta_{\mathrm{v}}$ based on the reward of each state-action pair and the state values of the current state and virtual state of the next time slot. The critic neural network is trained to reach the objective of minimizing the value of loss function $L_{\theta_v}$, which is represented as:

$$L_{\theta_{\mathrm{v}}}(t) = \frac{1}{W}\sum_{w=1}^{W} (U(S(t), a_w(t)) + \gamma V(S(t+1); \theta_{\mathrm{v}}) - V(S(t); \theta_{\mathrm{v}}))^2. \tag{14}$$

The loss function of $\theta_{\mathrm{v}}$ indicates the squared difference between the obtained state value in real and the estimated one. When the value of the loss function of $\theta_{\mathrm{v}}$ is minimized, the critic neural network is able to give accurate state value to each state. To reach the training objective, the state value of current state $V(S(t); \theta_v)$ is supposed to decrease when the reward that the worker agent obtains in real does not reach the expectation. Otherwise, the state value of current state is to be increased.

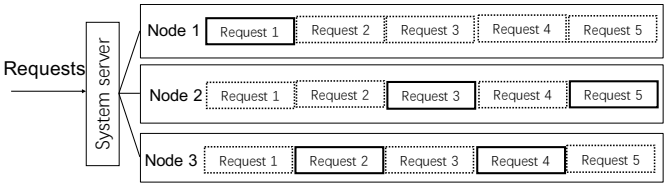



**Fig. 1** Example of system.

The state value parameter is updated based on the gradient of the loss function; the updating process is represented as:

$$\theta_{\mathrm{v}} \leftarrow \theta_{\mathrm{v}} + \lambda \nabla_{\theta_{\mathrm{v}}} L_{\theta_{\mathrm{v}}}(t), \tag{15}$$

where $\lambda$ is a weight coefficient.

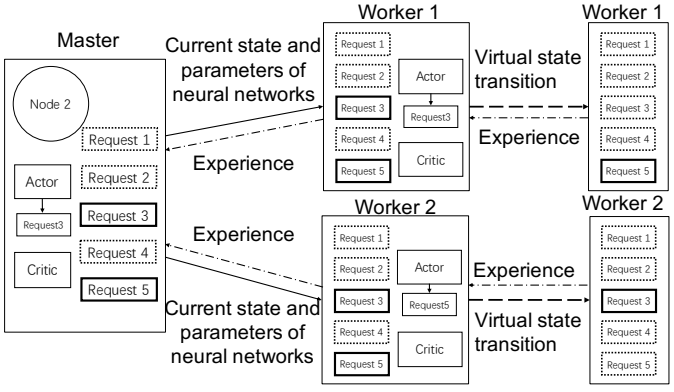



**Fig. 2** Simplified example of training frame on node 2.

Figure 1 shows a system with a system server and three NFV nodes, while the value of $\mathcal{R}$ is five. The system server keeps receiving requests until the number of requests in the system reaches $\mathcal{R}$. In each node, the rectangle with solid edges represents the request which has a VNF waiting to be processed or being processed on the node. The rectangle with dotted edges represents the request which has no VNF to be processed on the node. It shows that request 1 is on node 1, requests 3 and 5 are on node 2, and requests 2 and 4 are on node 3 currently. Figure 2 shows a simplified

frame on node 2; for simplicity, only two worker agents are shown. Worker 1 makes a decision of processing the VNF of request 3. In the next virtual time slot the processing is finished, and request 3 leaves from node 2. Worker 2 makes a different decision compared with worker 1, and obtains an opposite result. The experiences of workers 1 and 2 are sent back to the master agent, and the master agent takes the action of processing the VNF of request 3 after updating its actor neural network.

## 5. Performance evaluation

### 5.1 Evaluation baselines

We compare the two performance metrics of introduced DRL approach with those of existing approaches. The first metric is the ratio of the number of requests whose maximum admissible delays are satisfied to the total number of new coming requests in a fixed number of time slots. It is denoted by $Z = \frac{\sum_{t=2}^{\infty} \omega_t}{\sum_{t=1}^{\infty} \alpha_t}$, which corresponds to the objective value in (9). We call $Z$ a delay-satisfied ratio. The second metric is the computation time. For each approach we give the same initial environment where the parameters of the network system are uniform.

We construct a simulated server and a set of nodes. Requests are represented by arrays, which indicate the VNF deployment information and the required processing time. For example, $[[1, 2], [3, 1]]$ represents a request which includes two VNFs: the first VNF needs to be processed on node 1 for two time slots, and the second VNF needs to be processed on node 3 for one time slot. Each of request belongs to a service, which has a set of VNFs with fixed deployment parameters and processing time. An array is used to indicate the arrival times of requests. At the beginning of each time slot, the server receives requests whose arrival time is exactly this time slot and distributes them to nodes. After distribution, each idle node makes a decision on which VNF of which request needs to be processed according to an approach.

In greedy approaches, the request with the maximum or minimum value of an attribute is always selected. We consider three greedy approaches: minimal remaining admissible delay first (DG), in which the request with the minimal remaining admissible delay is always selected; minimal number of remaining VNFs first (VNG), in which the request with the minimal number of remaining VNFs is always selected; maximal waiting time first (WG), in which the request with the maximal waiting time on this node is always selected.

In a simulated annealing (SA) approach, there is a function placed in the server which gives the solution for requests remaining in the system in a limited number of time slots. The solution is repeatedly generated in random and converges to an approximate optimal or local optimal one. The result of SA approach changes in each trial, and we use the largest value among ten trials to represent the result of SA approach.

In the ILP approach, there is a function which is based on the IBM(R) ILOG(R) CPLEX(R) Interactive Optimizer with the version of 20 [1] placed in the server. The function gives the value of $h_{rvn}^t$ for each node with parameters $f_{rv}^n$ and $\rho_{rvn}$ in (5), (6), and (8) as the initial condition. After a distribution process in each time slot, the initial condition parameters of the ILP problem are updated by the requests remaining in the system. The function solves the ILP problem described in Section 3 in a limited number of time slots, and then each node makes processing decision of this time slot by referencing on the value of $h_{rvn}^t$ given by the function.

### 5.2 Parameter settings

We first consider a dynamic scenario which is similar with the static scenario. In this case, the requests arrive densely in several periods, and in other time slots there is no request. We name this case case 1, which has a bursty traffic. Generally, a cluster includes at least three nodes. We consider a small sized cluster with four nodes for case 1. Moreover, we consider five common VNFs: deep packet inspection (DPI), firewall (FW), network address translation (NAT), quality of service (QoS), and load balancing (LB). The processing time of VNFs in [17] is 2 to 100 microseconds. We set the unit of time slot in case 1 as 10 microseconds and the processing time of each VNF is one to four time slots. The total number of time slots considered in this case is 100, which is one second. We give three service function chains in case 1: DPI-FW-NAT, FW-LB, and QoS-LB-DPI. The cluster capacity usually depends on the memory capacity on nodes and is hard to estimate; we set the buffer capacity to ten requests in this case. Further, we set the interval between periods as 20 time slots.

We then consider a dynamic scenario where the requests come as Poisson flows, which are used as a common traffic model [18]. We name the case case 2. We keep using the small sized network as case 1. There are four nodes and three service function chains; the number of types of VNFs is five. The buffer capacity is 10 requests. The arrival rates of services are all set to 1. The simulation duration is 100 time slots.

For each request, we first calculate its minimal completion time, which is the sum of its arrival time and the total processing time of its VNFs. Then we generate a random time between the minimal completion time and future 10 time slots as its delay constraint for each request.

In all cases, the initial temperature of SA approach is set to 10 and the decay coefficient is set to 0.95. At the beginning of each time slot, the approach iterates the SA process until the temperature is below 0.05.

In the introduced DRL approach, we construct the actor neural network with 30, 20, and 20 neurons on the three hidden layers respectively and rectified linear units (ReLU) as the activation function. On the output layer, we use the softmax function to convert the output vector into the policy. The critic neural network also has 30, 20, and 20 neurons on the three hidden layers and ReLU as the activation function.

The parameters in the networks are initially generated by standard normal distribution and are updated gradually. Each master agent works with 5 worker agents. We train the neural network with different parameters in different cases. The changable parameters include the learning rate of networks, which are represented by $\eta$ and $\lambda$ in (13) and (15); the coefficients $\phi_1$, $\phi_2$, and $\phi_3$ in (10); the weight coefficient $\gamma$ in (11). Table 2 shows an example of the delay-satisfied ratios, $Z$, of the introduced DRL approach after training with different parameters in case 1 with buffer size 10. We choose the largest ratio as the result of the introduced DRL approach in case 1.

**Table 2** Delay-satisfied ratios $Z$ of introduced DRL approach after training with different paramters

| Parameters | | | | Learning rate | | |
|---|---|---|---|---|---|---|
| $\phi_1$ | $\phi_2$ | $\phi_3$ | $\gamma$ | 0.01 | 0.02 | 0.03 |
| 0.4 | 0.2 | 0.4 | 0.5 | 0.31 | 0.38 | 0.26 |
| 0.4 | 0.4 | 0.2 | 0.5 | 0.33 | 0.19 | 0.24 |
| 0.2 | 0.4 | 0.4 | 0.5 | 0.21 | 0.41 | 0.21 |
| 0.4 | 0.2 | 0.4 | 0.3 | 0.33 | 0.21 | 0.24 |
| 0.4 | 0.4 | 0.2 | 0.3 | 0.40 | 0.43 | 0.36 |
| 0.2 | 0.4 | 0.4 | 0.3 | 0.19 | 0.29 | 0.21 |

## 5.3 Training

We train the neural network before we apply the introduced DRL approach to cases. In both cases, the simulation time of an episode is 100 time slots. We train the neural network in different cases separately. In each episode of case 1, a number of requests of each service arrive in system every 20 time slots. In each episode of case 2, the arrival of requests follows Poisson distribution, where the average rate keeps consistent. The delay constraints of requests in each episode of both cases are different. In the training process, the worker first obtains the state array of the current state, and creates a virtual node which has the same waiting queue with the real node. The worker agent keeps taking virtual actions on the virtual node until no request is left on the virtual node. The worker agent records the initial state, the action, the next state, and the reward after taking each action. These results are recorded in a number of arrays as the training data. The arrays of training data are sent to the according master agent in the end of each episode. The master agent then updates its neural network with the training data. Each virtual action is decided by the Roulette strategy. In the Roulette strategy, the probability of an action to be chosen is proportional to the according value in the policy distribution. The policy distribution is generated by the actor network of the worker agent. In the end of each episode, each master agent counts up the accumulated reward in this episode by the arrays of training data that it received.

Figures 3 and 4 show the accumulated reward of master agents of each episode for each node and the delay satisfied ratio, $Z$, of each episode for whole system, respectively, when we train the neural network in case 2 with the buffer size of 12 requests. In Fig. 3, the increment of the accumulated rewards is not obvious. This is because the accumulated reward of each master agent is calculated independently rather than jointly; the increment of accumulated reward of a master agent may cause the decrement of that of another master agent. Nevertheless, in Fig. 4, the increment of delay satisfied ratio $Z$ of each episode is obvious; we observe that the result converges before the 20th episode, which is similar to Fig. 3.

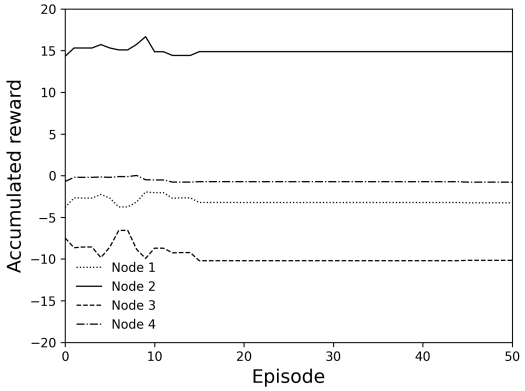



**Fig. 3** Accumulated reward by each episode.

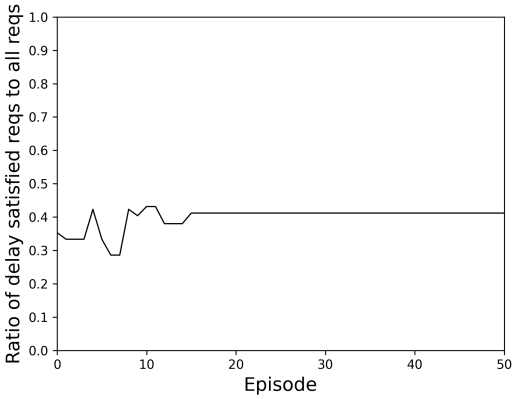



**Fig. 4** Delay satisfied ratio $Z$ by each episode.

## 5.4 Result comparison

Figure 5 shows the delay-satisfied ratios, $Z$, of approaches in case 1. DG, VNG, and WG represent the minimal remaining admissible delay first greedy approach, the minimal number of remaining VNFs first greedy approach, and the maximal waiting time first greedy approach, respectively. We observe that the ILP approach and the introduced DRL approach performs best in case 1. Meanwhile, the computation time of the introduced DRL approach is 231 times less than that of

the ILP approach; the computation times will be discussed in Table 3 later. In case 1, since each set of requests arrives at the same time slot and the arrival times of requests from different sets are far apart, the problem in the dynamic scenario can be approximately regarded as a sequence of independent problems in the static scenario. The result obtained by the ILP approach can be regarded as the optimal result.

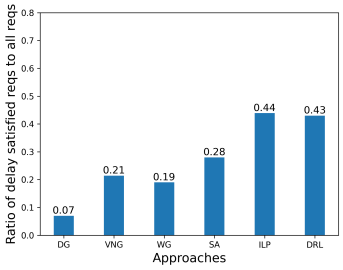



**Fig. 5** Delay-satisfied ratios $Z$ of approaches in bursty traffic (case 1).

Figure 6 shows the delay-satisfied ratios, $Z$, of approaches over the change of buffer size in case 1. We observe that $Z$ of the introduced DRL approach and that of the ILP approach perform better than greedy approaches for different buffer sizes. Further, we observe that the ratios, $Z$, of approaches tend to decrease when the buffer size grows. This is because, with increasing of buffer size, the server receives more requests; meanwhile, the number of nodes is limited, which means that the maximum number of requests that can be finished in one time slot is limited. The increment of the number of requests which are finished with satisfying their delay constraints is smaller than that of the total number of received requests in the system. As a result, the ratio of requests satisfying the delay constraint to the total requests decreases. Moreover, we observe that the introduced DRL approach performs better than greedy approaches and the SA approach when the buffer size increases.

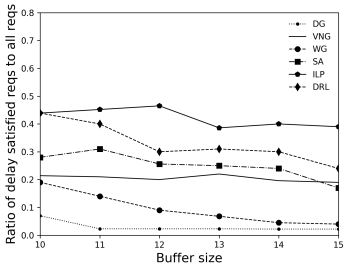



**Fig. 6** Delay-satisfied ratios $Z$ of approaches over buffer size in bursty traffic (case 1).

Figure 7 observes the delay-satisfied ratios, $Z$, of approaches in case 2. Similar to case 1, we observe the results of the introduced DRL approach after training with different parameters. We choose the result trained with the most suitable parameters as the result of the introduced DRL approach. We observe that the ILP approach and the introduced DRL approach are larger than those of the greedy approaches and SA approach. The performance of delay-satisfied ratios $Z$ of the introduced DRL approach is worse than that of the ILP approach, while the computation time of the introduced DRL approach is 315 times less than that of the ILP approach; the computation times will be discussed in Table 3 later.

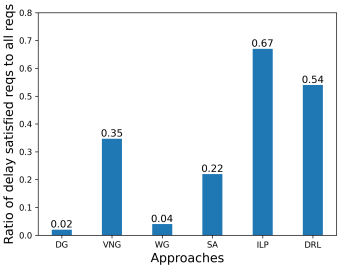



**Fig. 7** Delay-satisfied ratios $Z$ of approaches in Poisson traffic (case 2).

Figure 8 shows the delay-satisfied ratios, $Z$, of approaches over the change of buffer size in case 2. We observe that similar to case 1, the ratios, $Z$, of approaches tend to decrease when the buffer size grows, while the ratio performances of ILP and introduced DRL approaches are better than those of other approaches. We also observe that the ratio performance of VNG approach improves obviously when the buffer size grows from 11 to 12 and drops back when the buffer size grows from 12 to 13. This is because with the increment of buffer size, requests tend to gather in one node in the VNG approach. If the buffer size becomes small, the queue can include a higher ratio of requests which have a strict delay constraint to all requests in the queue. If the buffer size becomes large, the node cannot service the queue immediately, which can cause some requests stay too long in the queue to meet the delay constraints. In this case, the buffer size of 12 requests is the optimal choice for the VNG approach. In case 2, the performance of the introduced DRL approach is reliable; the decrement is not obvious when the buffer size increases.

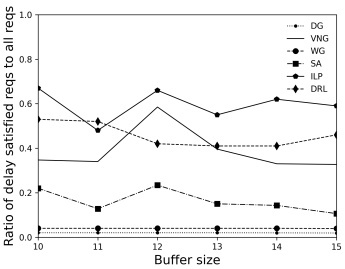



**Fig. 8** Delay-satisfied ratios $Z$ of approaches over buffer size in Poisson traffic (case 2).

We apply the approaches in case 3, which is based on the data of Alibaba cluster-usage traces in [16]. The data set includes 3865 machines in a period of eight days. The workload consists of a set of tasks, where each task runs on a single machine. Each task belongs to a single job; a job may have multiple tasks. According to this characteristic, we regard each job as a request coming into the SDN-NFV system, with the tasks in a job as VNFs. A machine where one or more tasks are placed is considered as an NFV node. The data set consists of the information of tasks. In the table of tasks, we obtain the identifier of each task and the job that it belongs to. The identifier of each task indicates the dependency relationship with other tasks in the job.

Table 3 shows an example of a job in the dataset. In job j_2, there are two tasks: M1 and R2_1. Task R2_1 can only be started after M1 is finished. The arrival time of this job is 87076. The tasks are classified into 12 types. The unit of the dataset is an instance. For each task in each job, the processing of batch data is divided into multiple instances; different instances are placed on different machines in general. In other words, a task is placed on multiple machines in this dataset, which is different from our considered condition. According to the condition, we use another method to assign the placement of tasks. We consider 300 machines and equally divide them into 12 sections; each section is in charge of an exact task type. According to its type, we randomly distribute each task to a machine in the corresponding section. We can also obtain the start time and end time of each task. The unit of the times is second. The difference between the start time and the end time of the task is regarded as its required processing time. The start time of the first task in each job is regarded as the arrival time of the job. The data set does not include the delay constraint information for each job. Hence, for each job we generate a random number which is lager than its earliest finish time as its delay constraint. We capture a part of the data, which includes 2225 jobs.

**Table 3** Part of dataset

| Task name | Job name | Task type | Start time [s] | End time [s] |
|---|---|---|---|---|
| R2_1 | j_2 | 1 | 87076 | 87086 |
| M1 | j_2 | 1 | 87076 | 87083 |

Figure 9 shows the ratios, $Z$, of approaches in case 3. In case 3, the ILP problem cannot be solved. This is because the sizes of $N$, $R$, $V$, and $T$ in the static model in Section 3 are so large that the ILP problem cannot be solved in the server. We observe that $Z$ of the introduced DRL approach is obviously larger than those of other approaches. Meanwhile, $Z$ of remaining approaches are comparable. The computation time of introduced DRL approach is also larger than those of other approaches.

Figure 10 shows the ratios, $Z$, of approaches over the change of buffer size in case 3. We observe that in case 3, the change of $Z$ of approaches when the buffer size grows is less obvious than those of cases 1 and 2; there are two

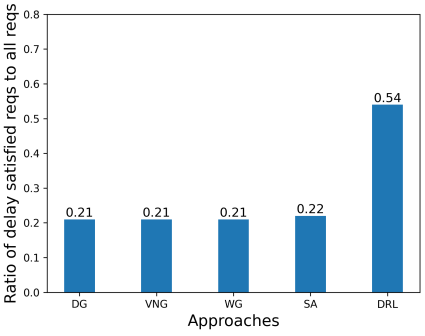



**Fig. 9** Delay-satisfied ratios $Z$ of approaches in data of Alibaba cluster-usage traces (case 3).

reasons. Firstly, the total number of nodes in case 3 is large enough; meanwhile, the assignment of the VNF placement of requests is scattered. The maximum number of requests that can be finished in one time slot in case 3 is less limited than those of cases 1 and 2. Secondly, the time intervals between the arrival times of requests are larger than those of cases 1 and 2, which causes a smaller number of requests to stay in the system in most of time slots. In case 3, the performance of the introduced DRL approach is reliable, which is similar to that in case 2.

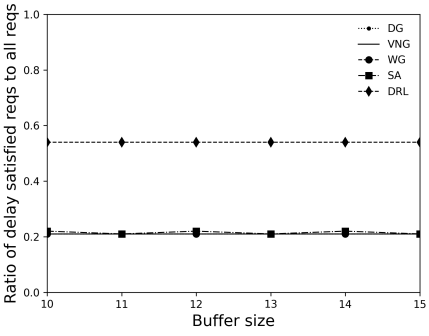



**Fig. 10** Delay-satisfied ratios $Z$ of approaches over buffer size in data of Alibaba cluster-usage traces (case 3).

The computation times of approaches in different cases are shown in Table 4. The introduced DRL approach runs slower than greedy approaches, but much faster than the ILP approach. The introduced DRL approach runs slower than greedy approaches, but the ratio of delay satisfied requests to all requests obtained by the introduced DRL approach is the same or better comparing with those obtained by greedy approaches in all three cases. The ratios, $Z$, obtained by the ILP approach are close to that of the introduced DRL approach in some cases, but the computation times of the ILP approach are excessively longer than those of others. The introduced DRL approach can obtain the best ratio in a suitable computation time, compared with other approaches.

**Table 4** Computation times of approaches

| Approaches | Computation times [s] | | |
|---|---|---|---|
| | Case 1 | Case 2 | Case 3 |
| DG | 0.074 | 0.13 | 1.03 |
| VNG | 0.11 | 0.196 | 1.05 |
| WG | 0.138 | 0.186 | 1.03 |
| SA | 5 | 5 | 52.95 |
| ILP | 2327 | 3568 | - |
| DRL | 10 | 12 | 111.43 |

## 6. Conclusion

This paper introduced a DRL approach to solve the VNF scheduling problem in dynamic scenarios. We described the mathematical model in the static scenario. The scheduling problem in the static scenario was reformed to fit the dynamic cases. We defined the state, action, and reward for learning agents. We introduced the A3C algorithm in the DRL approach and the training process. The ratios of delay satisfied requests to all requests of the introduced DRL approach were compared with those of the existing approaches in two simulated cases and the data of Alibaba cluster-usage traces. Numerical results observed that our introduced DRL approach improves the number of requests satisfying the delay constraints and is reliable when environment changes.

## Acknowledgements

This work was supported in part by JSPS KAKENHI, Japan, under Grant 21H03426.

### References

[1] U. Ashraf, "Placing controllers in software-defined wireless mesh networks," in *2018 Int. Conf. Comput. Math. Eng. Technol. (iCoMET 2018) on*, Mar. 2018, pp. 1–4.

[2] S. I. Kim and H. S. Kim, "A VNF placement method based on VNF characteristics," in *2021 Int. Conf. Inf. Netw. (ICOIN 2021)*, pp. 864–869, 2021.

[3] Z. Wang, J. Zhang, H. Wei, and T. Huang, "Hieff: Enabling efficient VNF clusters by coordinating VNF scaling and flow scheduling," in *2020 IEEE 39th Int. Perform. Comput. Commun. Conf. (IPCCC 2020)*, pp. 1–8, 2020

[4] X. Zhao, X. Jia, and Y. Hua, "An efficient VNF deployment algorithm for SFC scaling-out based on the proposed scaling management mechanism," in *2020 Inf. Commun. Technol. Conf. (ICTC 2020)*, pp. 166–170, 2020.

[5] J. Li, W. Shi, N. Zhang, and X. Shen, "Delay-aware VNF scheduling: A reinforcement learning approach with variable action set," *IEEE Trans. Cogn. Commun. Netw., vol. 7, no. 1*, pp. 304–318, 2021.

[6] J. F. Riera, E. Escalona, J. Batallé, E. Grasa, and J. A. García-Espín, "Virtual network function scheduling: Concept and challenges," in *2014 Int. Conf. Smart Commun. Netw. Technol. (SaCoNeT 2014)*, pp. 1–5, 2014.

[7] J. F. Riera, X. Hesselbach, E. Escalona, J. A. García-Espín, and E. Grasa, "On the complex scheduling formulation of virtual network functions over optical networks," in *2014 16th Int. Conf. Transparent Opt. Netw. (ICTON 2014)*, pp. 1–5, 2014.

[8] N. Yamaguchi, O. Fukuda, and H. Okumura, "Model-based reinforcement learning with missing data," in *2020 8th Int. Symp. Comput. Netw. Workshops (CANDARW 2020)*, pp. 168–171, 2020.

[9] K. Shiomoto and T. Kurimoto, "Policy gradient-based deep reinforcement learning for deadline-aware transfer over wide area networks," in *2021 7th IEEE Int. Conf. Netw. Softwarization (NetSoft 2021)*, pp. 166–170, 2021.

[10] J. Bae, J. Lee, and S. Chong, "Beyond max-weight scheduling: A reinforcement learning-based approach," in *2019 WiOPT*, pp. 1–8, 2019.

[11] T. Zhou, D. Tang, H. Zhu, and L. Wang, "Reinforcement learning with composite rewards for production scheduling in a smart factory," *IEEE Access, vol. 9*, pp. 752–766, 2021.

[12] S. Yang and Z. Xu, "Intelligent scheduling for permutation flow shop with dynamic job arrival via deep reinforcement learning," in *2021 IEEE 5th Adv. Inf. Technol. Electron. Autom. Control Conf. (IAEAC 2021)*, vol. 5, pp. 2672–2677, 2021.

[13] Y. Wang, H. Liu, W. Zheng, Y. Xia, Y. Li, P. Chen, K. Guo, and H. Xie, "Multi-objective workflow scheduling with deep-Qnetwork-based multi-agent reinforcement learning," *IEEE Access, vol. 7*, pp.

39974–39982, 2019.

[14] Z. Yan, J. Ge, Y. Wu, L. Li, and T. Li, "Automatic virtual network embedding: A deep reinforcement learning approach with graph convolutional networks," *IEEE J. Sel. Areas Commun., vol. 38, no. 6*, pp. 1040–1057, 2020.

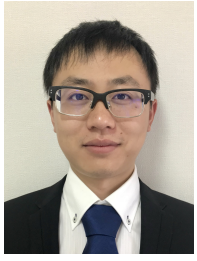
[15] X. Chen, B. Li, R. Proietti, H. Lu, Z. Zhu, and S. J. B. Yoo, "DeepRMSA: A deep reinforcement learning framework for routing, modulation and spectrum assignment in elastic optical networks," *J. Light. Technol, vol. 37, no. 16*, pp. 4155–4163, 2019.

[16] J. Guo, Z. Chang, S. Wang, H. Ding, Y. Feng, L. Mao, and Y. Bao, "Who limits the resource efficiency of my datacenter: An analysis of alibaba datacenter traces," in *2019 IEEE/ACM 27th Int. Symp. Qual. of Serv. (IWQoS 2019)*, pp. 1–10, IEEE, 2019.

[17] S. Geissler, S. Lange, F. Wamser, T. Zinner, and T. Hoßfeld, "KOMon — Kernel-based Online Monitoring of VNF Packet Processing Times," *2019 International Conference on Networked Systems (NetSys), 2019*, pp. 1-8.

[18] J. Zhang, J. Tang, X. Zhang, W. Ouyang, and D. Wang, "A survey of network traffic generation," *Third International Conference on Cyberspace Technology (CCT 2015), 2015*, pp. 1-6.

**Zixiao Zhang** is pursuing the M.E. degree at Graduate School of Informatics, Kyoto University, Kyoto, Japan. He received the B.E. degree from Beijing University of Posts and Telecommunications, China, in 2020. His research interests include machine learning, scheduling, optimization, and resource allocation.



**Fujun He** received the B.E. and M.E. degrees from University of Electronic Science and Technology of China, Chengdu, China, in 2014 and 2017, respectively, and the Ph.D. degree from Kyoto University, Kyoto, Japan, in 2020. His research interests broadly lie in communication and computer networks. Recently, his work focuses on network virtualization/softwarization, optical networks, and artificial intelligence for network control.

**Eiji Oki** is a Professor at Kyoto University, Kyoto, Japan. He was with Nippon Telegraph and Telephone Corporation (NTT) Laboratories, Tokyo, from 1993 to 2008, and The University of Electro-Communications, Tokyo, from 2008 to 2017. From 2000 to 2001, he was a Visiting Scholar at Polytechnic University, Brooklyn, New York. His research interests include routing, switching, protocols, optimization, and traffic engineering in communication and information networks.