Environment-Adaptive Sizing and Placement of NFV Service Chains with Accelerated Reinforcement Learning

Manabu Nakanoya, Yoichi Sato, Hideyuki Shimonishi
System Platform Research Laboratories
NEC Corporation
1753 Shimonumabe, Nakahara-ku, Kawasaki, Kanagawa, Japan
Email: {m-nakanoya@bc, y-sato@jw, h-shimonishi@cd}.jp.nec.com

***Abstract*—As network function virtualization spread, network service providers have been able to deliver various networks flexibly and rapidly. In particular, products and services that build network functions on a wide area network of organizations, such as enterprises, have been spreading. Since the user substrate environment and performance requirement differ in such services, optimal virtualized network function (VNF) resource sizing and placement need to be considered individually. To adapt to such environmental diversity, methods for applying reinforcement learning (RL), which includes an adaptive optimization mechanism, have been proposed. However, current RL methods have difficulty to complete learning on a real network because of too many required explorations. We propose an accelerated RL method that can learn proper VNF sizing and placement on a real network under various environments. Our method divides the RL process into two steps depending on the learning objective. We compared the proposed and a conventional RL methods through three scenarios with different substrates. We confirmed that the conventional RL method cannot learn properly even if it takes ten thousand explorations, whereas, our method can learn a cost-efficient resource sizing and placement that meets the performance requirements within only one thousand explorations.**

***Index Terms*—Resource Sizing, NFV Placement, Reinforcement Learning, NFV service**

I. INTRODUCTION

As network virtualization and softwarization spread through the use of network function virtualization (NFV), a network provider can deliver various virtual networks more flexibly and rapidly than using raw hardware devices. Many solutions have been proposed using these technologies to decrease operating costs and service delivery time in various domains such as carrier networks. Regarding enterprise networks, some products and services that build network functions on-demand on a wide area networks (WANs) of organizations have been proposed. NFV Management and Orchestration formulated by ETSI ISG [1] is a standard of NFV lifecycle management including deployment of virtual network functions (VNFs). By using an implementation [2] [3] compliant with this standard, we can deploy a service function chain (SFC) consisting of VNFs. Also, Cisco provides Enterprise Service Automation [4] as the NFV orchestrator that supports design and deployment of SFCs.

Those who use such a service or product have a dedicated substrate network and communication requirements corresponding to their business requirements. These users need to consider optimal VNF resource sizing and placement individually because network performance differs depending on WAN specifications (bandwidth and latency), hardware where the SFC is deployed, and so on. To determine these configurations, methods for predicting network performance, such as throughput and latency, have been proposed. There are two major methods: one that models packet processing by queuing theory, and the other, which is a machine-learning method that uses measured values as learning data, the modeling method cannot determine enough configurations because an actual network contains factors outside the model. However, with the machine-learning method, which uses supervised learning approaches, it is difficult to prepare proper learning data that can be applied to various environments in advance, although it can take the factors outside the above model into consideration.

To overcome this problem, reinforcement learning (RL) [5] is a promising technology because it has a framework that not only autonomously acquires behavioral models of networks including performance but also generates a configuration-control mechanism based on the models. However, conventional RL methods cannot be easily applied to sizing and placement of SFCs because of the issue of the time required to learn. That is, the complicated task including a decision of a cost-efficient placement and resource sizing of multiple SFCs first requires more than 10,000 of explorations (for example, it takes more than 24 hrs, if each exploration takes 10 seconds.); second is that it is difficult for an SFC running on a dedicated substrate to individually prepare a simulation environment in which real network performance can be effectively and quickly measured.

We propose an SFC resource sizing and placement method using an accelerated RL. The proposed RL method is executable on not only a simulation but a real network environment because it requires less explorations. Our RL method divides the RL process into two steps depending on the learning objective. In the first step, the learning target is the general performance dynamics of the SFC, and in the second step, the target is the desirable configuration that depends on a user's specific substrate network and performance requirements. The advantages of dividing the RL process into two steps are as follows: the first step can be excluded from explorations for individual users because it can be commonly executed before the user service request. In the second step, learning about just the user-specific factor, which depends on the network specification and communication requirements, is sufficient, and effective exploration is possible due to using the knowledge from the first step.

We compared our RL method and a conventional RL method on a real testbed network emulating five distributed sites. We evaluated the methods through three scenarios with different network environments. We confirmed that the conventional RL method cannot learn the proper sizing and placement strategy even if it takes 10,000 explorations, whereas, our method can learn a cost-efficient policy that meets these requirements within only one thousand explorations.

The rest of this paper is organized as follows. We first present related work in Section II then explain the problem and details of the proposed method to solve this problem in Section III. We discuss an evaluation we conducted to compare the proposed RL method and a conventional RL method and present the evaluation results in Section IV and offer concluding remarks in Section V.

II. Related Works

A. VNF Resource Sizing and Placement Problem

The process of deploying SFCs is divided into three steps: the first step involves specifying the necessary network function and building SFCs by assembling VNFs. The second step involves determining the placement location of the VNFs and the resource size from the performance requirements. The third step is placing the VNFs on an actual execution environment based on the design determined in the previous steps. The first step has been studied as designing SFC technology from abstract user requirements; specifically in the fields of Intent-based Networking [6] and Policy Refinement [7]. Many studies reported the third step as virtual network embedding (VNE) [8] and VNF allocation [9] problems. Most of the methods used to tackle these problems are aimed at discovering cost-efficient placement and resource allocation in a specific datacenter (DC) from the given resource allocation [10] [11]. The second step, which is what we address in this paper, has generally been investigated in terms of NFV-performance-prediction technology because we can determine the configuration required for given performance requirements if we know the performance in a particular configuration. There are two major methods of predicting performance: one that models packet processing by queuing theory. The other is a machine-learning method with measured values as learning data. Gallardo et al. [12] proposed a queuing model for the performance evaluation of a virtual switch. It can approximate performance metrics, such as packet latency, with good accuracy to determine the number of CPU cores. However, it is not enough to predict performance of the whole SFC because SFC performance is affected by not only packet processing inside switches but also other processes such as applications running on the same environment and not relevant to the SFC. The difference between our proposed method and that of modeling packet processing is that our method takes all elements affecting SFC performance on the execution environment into account by using the measured end-to-end performance data obtained from a real network. Lei et al. [13] proposed a latency-prediction method using a machine-learning method called random forest regression. Similarly, Gupta et al. [14] proposed a method that generates preferable placement of VNFs using support vector regression (SVR). The SVR model can predict the latency of SFCs from resource usage and the geographical distance between DCs under the assumed situation of fixed SFCs and service level agreement (SLA). Since these methods use supervised learning, they require all learning data in advance for every environment. Therefore, a machine-learning engineer has to generate learning data suitable for the dedicated environment every time because automation of generating such learning data does not yet exist. The proposed method can automatically learn SFC configurations on multiple dedicated environments.

B. Application of Reinforcement Learning

The two advantages of using RL for configuring the sizing and placement of VNF are as follows. The first is the high degree of freedom of the RL model because even if the internal structure is unknown, the RL model can include any variable if observable. The second is that RL does not require learning data that can adapt to every assumed situation in advance because learning progresses incrementally by sequential observations from learning the target environment. However, RL generally takes a considerable number of explorations to complete learning. Therefore, our accelerated RL method overcomes this explorations issue.

There have been attempts in applying RL to deployment optimization for virtual components regarding the resource-allocation problem of Virtual Machines (VMs) in a cloud environment such as large DC. Rao et al. [15] and [16] proposed methods for autonomously learning resource allocation of VMs satisfying SLA with RL. These methods include effective modeling techniques for connecting RL and resource allocation problems.

Tang et al. [17] reported on an application for a scale-out control of VNFs in a telecom operator's network as an application of network management. They gave definitions of the state space and reward function that enables the learning of a scale-control policy that adequately controls resource usage while satisfying SLA in traffic patterns in a telecom network. S. I. Kim and H. S. Kim [18] tackled SFC allocation and presented a learning method that places VNFs on an appropriate node that maximizes the performance of VNFs according to

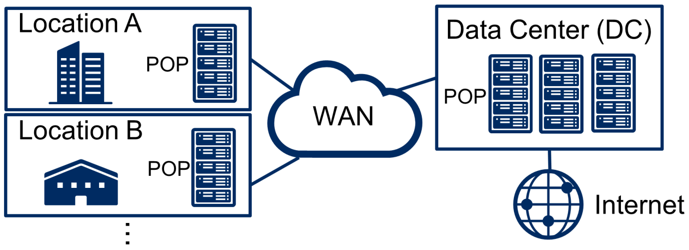

Fig. 1. Network topology.

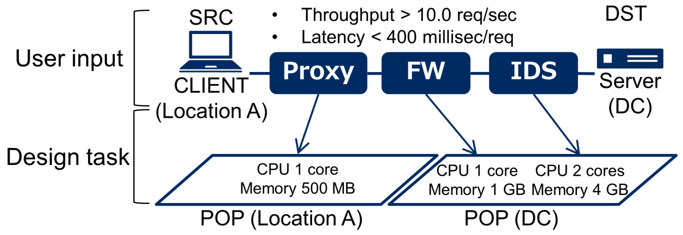

Fig. 2. Flow of service function chain (SFC) sizing and placement.

the load condition of the physical network node and the link with the unique reward function. However, this method cannot be applied to an RL task that has an extensive exploration space including a few placement-location representatives and options of resource allocation because of using table lookup Q-learning [19], which takes a huge amount of time to converge under such a space. Mijumbi et al. [20] proposed a dynamic virtual network allocation method against service requests occurring at any time during service operation. Their method can learn complicated cost-optimal resource allocation with a set of small-space (approximately 5,000) RL tasks using multi-agent RL. However, this method has to be executed on a simulation environment because it requires more than 10,000 explorations to learn the optimal policy. Therefore, it cannot be applied to SFCs including application or VNFs, which is difficult to reproduce with a simulator. It also does not adjust the balance between performance and cost, as mentioned in Section I, because it is focused on the availability of virtual network services.

To summarize, it is difficult to apply conventional RL methods to a network service that must satisfy individual requirements under different network substrates. The advantage of our proposed method is that it can not only learn from an extensive exploration space task but also be executed on a real network.

III. Resource Sizing and Placement Problem Description and Proposed Method

This section presents the resource sizing and placement problem that we tackle and our method that can solve it. We first give an overview of this problem. Next, we give a concrete definition of an RL task that can be applied to the problem. Finally, we discuss our proposed method.

A. Problem Overview

Figure 1 shows a network topology of a network service using NFV. A network service user, such as an enterprise, has several bases of activity (Locations A, B,... in Figure 1). These bases and a DC are connected with a WAN. Communication-destination servers, such as application servers, are placed in the DC, whereas, client terminals are placed at each location. Clients and servers communicate with each other. In addition, we consider inbound traffic from the Internet to the DC as well as outbound traffic from the DC to the Internet. Hardware resources on which a user can deploy VNFs are equipped at each location. We call these resources points of presence (POPs). VNFs to be deployed are deployed on one POP. We assume that local POP resources vary for each POP and have less than that of DC's POP.

Figure 2 shows the flow of SFC sizing and placement. A service user first chooses an SFC corresponding to a function he/she wants to use from the provided SFC service menu. Next, he/she specifies the traffic source (SRC) and communication destination (DST) where the SFC is to be deployed. Additionally, the user inputs performance requirements about the SFC such as throughput and latency. From these inputs, a SFC designer chooses the place where each VNF (for example, proxy, firewall (FW) and intrusion detection system (IDS)) related to the specified SFC is to be deployed. The amount of resources allocated to each VNF is also decided. In this paper, we do not consider the place VNF is deployed other than the SRC and DST. When an NFV service users request these SFC deployments, they need to decide the sizing and placement of the SFCs that fulfill the requirements without manual operation.

B. Definition of Reinforcement Learning Task

In this section, we give a definition of a RL task that can solve the sizing and placement problem of SFCs presented in the previous section. RL proceeds by interaction between the environment and agent. In this paper, the environment corresponds to the substrate network on which VNFs are actually deployed and traffic flows. The agent is the subject of solving the problem and attempts to learn a configuration strategy, which is called a policy, with rewards obtained from the environment as a clue. There are several learning algorithms that differ in how reward is reflected as policy. A policy returns a configuration change (or its probability distribution), called action, such as placing a VNF on a specific POP, allocating CPU resources to a VNF, and so on. A well-learned policy can return actions that can satisfy user requirements with less resource consumption in different SFC configurations, which is called states. Defining the RL task that solves the problem showed in the previous section means that we specify the definition of the state, action, reward, learning algorithm and policy that correspond to the problem.

1) Definition of common variables: Before defining each element comprising an RL task, we give common variables in Table I. an SFC instance represents an instance that can be generated from a user's request specifying SFC, SRC and

TABLE I
COMMON VARIABLES FOR RL TASK DEFINITION.

name	description
$\mathbf{Loc}$	Set of locations (Including DC).
$loc_i \in \mathbf{Loc}(i=1,..,I)$	location i.
$\mathbf{SFC}$	Set of SFC of service menu.
$sfc_j \in \mathbf{SFC}(j=1,..,J)$	SFC j.
$\mathbf{VNF}$	Set of VNF.
$vnf_k \in \mathbf{VNF}(k=1,..,K)$	VNF k.
$\mathbf{SFCI}$	Set of SFC instances.
$sfci(i_1,i_2,j) \in \mathbf{SFCI}$	Element of $SFCI$ where SRC is loc_{i_1} , DST is loc_{i_2} ($i_1 \neq i_2$) and SFC is sfc_j .
$P_{src/dst}(i_1,i_2,j,k)$	Presence of VNF where POP is at SRC or DST, SRC is loc_{i_1}, DST is loc_{i_2}, SFC instance is $sfci(i_1,i_2,j)$ and VNF is vnf_k.
$C_{src/dst}(i_1,i_2,j,k)$	Amount of CPU allocation where POP is at SRC or DST, SRC is loc_{i_1}, DST is loc_{i_2}, SFC instance is $sfci(i_1,i_2,j)$ and VNF is vnf_k.
$M_{src/dst}(i_1,i_2,j,k)$	Amount of memory allocation where POP is at SRC or DST, SRC is loc_{i_1}, DST is loc_{i_2}, SFC instance is $sfci(i_1,i_2,j)$ and VNF is vnf_k.

DST. For example, the SFC menu provides two SFCs (sfc_1, sfc_2), and locations are defined as location-A (loc_1) and DC (loc_2). The $\mathbf{SFCI}$ defined in Table I is represented as follows: **SFCI** = $\{sfci(loc_1, loc_2, sfc_1)$, $sfci(loc_1, loc_2, sfc_2)$, $sfci(loc_2, loc_1, sfc_1)$, and $sfci(loc_2, loc_1, sfc_2)\}$.

2) Definition of state: The states we address in this paper are formed as a vector listing the amounts of resource allocation for each VNF (When a VNF is regarded as a VM, similar state design is adopted [15] [16]). Specifically, state $s \in \mathbf{S} = \{(P_{src}(i_1,i_2,j,k),\ C_{src}(i_1,i_2,j,k),\ M_{src}(i_1,i_2,j,k),\ P_{dst}(i_1,i_2,j,k),\ C_{dst}(i_1,i_2,j,k),\ M_{dst}(i_1,i_2,j,k))\ |foreach(i_1,i_2) = \{\ {}_{i_1}P_{i_2}\ |i_1,i_2 \in \{1,..,I\}\ ,\ i_1 \neq i_2)\}\}$, where ${}_{i_1}P_{i_2}$ is a permutation.

3) Definition of action: The actions $a \in \mathbf{A}$ we address in this paper increase or decrease the number of CPU or memory usage assigned to VNFs or changing the placement of a VNF. The step size of the increase and decrease is constant for each resource defined in advance. The design that defines the increase and decrease of resource allocation size as an action is general because several studies have adopted it [15] [16] [20]. This has the effect of efficiently learning the relationship between state and reward by sequentially searching for similar states (adjacent states) when searching the state space. We have to consider moving a VNF as an action. Therefore, we restrict the movable VNF to a VNF that does not involve the movement of other VNFs, in other words, a VNF deployed at the edge of a POP. This restriction prevents significant configuration change at one action.

4) Definition of reward: We use two different reward functions depending on the learning objective. These two functions correspond to two-step RL which we discuss in III-C. The first function simply returns SFC performance as rewards because the first RL step is aimed at learning the general performance dynamics not depending on a user's environment and requirement. In this paper, we use throughput latency ratio as the performance of the SFC. Hence, the first reward function $r_{step1}(i_1,i_2,j)$ is represented as

$$r_{step1}(i_1,i_2,j) = \sum_{\mathbf{SFCI}} \frac{throughput_{sfci(i_1,i_2,j)}}{latency_{sfci(i_1,i_2,j)}}. \qquad (1)$$

The second function represents the degree of satisfaction of comprehensive user requirements considering resource consumption. To specify the degree of satisfaction, we first define the variable deg representing the degree of satisfaction of a single SFC instance as $\min_{m \in \mathbf{Metric}}\{mperf_m/tperf_m\}$, where $\mathbf{Metric}$ is a set of performance metrics, $mperf_m$ is the measured performance value of metric m, and $tperf$ is target performance value of m. If a small value is more preferable than a large one, the inverse value is used. Since deg takes a minimum value among the performance-achievement degrees of all metrics, it can always reflect the metric to be improved the most. Next, we define a user's utility function regarding the performance of single SFC instance as the combination of the linear function using deg and step function that has an upper and lower limit. That is, the utility function $u(i_1,i_2,j)$ for each SFC instance is shown as

$$u(i_1,i_2,j) = \begin{cases} K & deg > b \\ 0 & deg < d \\ \frac{K}{b-d}(deg-d) & Otherwise \end{cases} \qquad (2)$$

where K is the upper limit and an arbitrary constant. The variables d and b ($d < b$) adjust the range of deg where the utility function linearly varies. Finally, the reward of the second RL step $r_{step2}(i_1,i_2,j)$, involving consideration of resource consumption, is defined as $r_{step2}(i_1,i_2,j) = \sum_{\mathbf{SFCI}} u(i_1,i_2,j)/cost$, where $cost$ is $\sum_{\mathbf{SFCI}}(w_{cpu}C(i_1,i_2,j) + w_{mem}M(i_1,i_2,j))$ and w_{cpu} and w_{mem} are the weights per unit resource for each resource and specified by the user in advance. This reward definition returns the maximum value when the requirements are satisfied with a margin (b) while suppressing resource consumption.

5) Learning algorithm: In this study, we adopted algorithms based on value iteration in which there has been extensive investigation into the IT resource allocation problem as an RL algorithm. Algorithms based on value iteration work on the assumption that the value is expressed by the scalar value for each state. These algorithms learn a policy through improving the state-value function. A state-value function takes a state as input and returns the estimated value of the state. A policy refers to a rule that determines actions that can transition to a higher value state. Algorithms based on value iteration includes Monte Carlo (MC) [21], Q-learning [19], and SARSA [22]. In this study, we used the MC algorithm, which is a simple algorithm of updating the state-value function. When the state $S = \{s_i | i = 1,...,N\}$ and reward $\{r_i | i = 1,...,N\}$ (N is the number of actions) are observed in one series of exploration, which is called episode, the return for state s_i, which represents the discounted present value per episode R_{s_i}, is $R_{s_i} = \sum_{l=i}^{N} r_l \delta^{l-1}$, where δ is a discount rate ($0 < \delta \leq 1$).

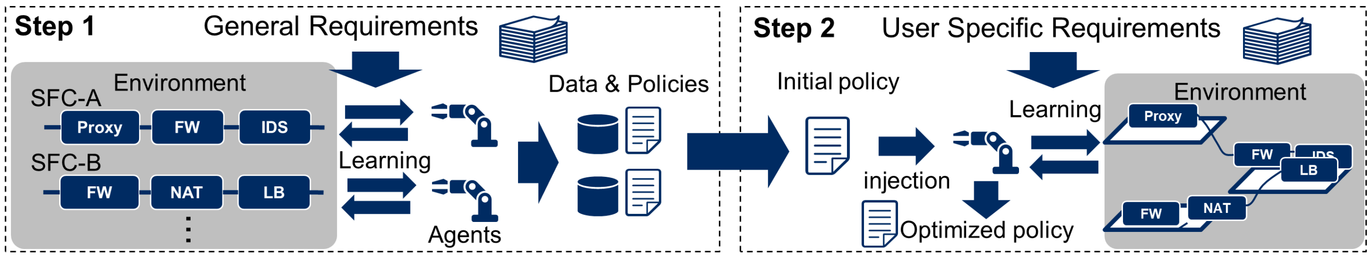

Fig. 3. Overview of 2-step reinforcement learning (RL)

In the MC algorithm, the estimated state-value is the average of R_{s_i} for each state.

The state space shown in III-B2 can be huge. For instance, a user specifies four SFCs. Each SFC has an average of three VNFs, and there are five assignment patterns of each CPU and memory. The size of the state space under this condition is approximately $5^{4\cdot3\cdot2} = 5^{24} \approx 10^{17}$. It is difficult to complete learning with such a large state space size if we execute table-lookup-type learning that records and updates the estimates of the value for each state because the memory area for storing the table becomes enormous or a large number of explorations is required until convergence of the value function. Therefore, we use a learning method called function approximation that approximates the state-value function with an arbitrary prediction function rather than a list of state and estimated value pairs. With function approximation, the state-value function is updated by improving the parameters of an approximation function. With function approximation in MC algorithm, the approximation function is determined by determining the parameter θ of the approximation function $V(s, \theta)$ that minimizes the prediction error using the learning data set $< \mathbf{S}, R_S >$, where $\mathbf{S}$ is the union of S of each episode, and R_S is the set of $R_s (s \in \mathbf{S})$.

6) Policy definition: In this paper, the policy to decide an action $\pi(s, a)$ is expressed as the probability of taking action a on state s. In exploration during learning, $\pi(s, a)$ is calculated using the softmax policy [23] as follows.

$$\pi(s,a) = \frac{exp(\frac{V(S(s,a),\theta)}{\tau})}{\sum_{a\in A} exp(\frac{V(S(s,a),\theta)}{\tau})} \tag{3}$$

where τ is the so-called inverse temperature, and $S(s, a)$ represents the set of states of the transition destination when taking action a in state s. Our proposed method is done by on-policy learning (it means learning target policy and exploration policy is the same) using the above policy.

C. Two-Step Reinforcement learning

1) Overview: The overview of two-step RL we proposed is shown in Figure 3. The first step involves learning the general performance characteristics of each SFC excluding a service user's specific requirements, such as performance and resource cost, on each POP. SFC providers or users can carry out this learning step in advance before a user's service request. Therefore, the execution time of learning does not affect the lead time of the service. This step generates two outputs. The first is the policy that expresses what placement and allocation are preferable when the upper limit of a POP's allocated resources and resource cost are not considered. The second involves learning the log data consisting of a list of state and reward pairs during that state.

In the second step, the RL agent learns the optimal allocation and placement control policy depending on a service user's specific environment. Acceleration of learning at this step leads to shortening the lead time of service delivery. In the first step, the RL agent learns an SFC's behavior, such as performance, without assuming any knowledge about that behavior. Whereas, in the second step, the RL agent improves the policy by diverting the knowledge obtained at the first step. Finally, the policy converged at this step is the optimal policy for a user's specific environment and requirements.

2) Learning at first step: We define the RL tasks for each SFC in the first RL step. SRC and DST are assigned to the location that has enough resources. A user's requirements of performance and resource cost weights are not specified. Hence, the state and action explained in III-B can be applied to this first RL step without any consideration, and the reward function is the throughput latency ratio, the equation (1).

3) Learning at second step: There are two differences between learning at the second step and that at the first step. The first is to initialize the approximate function $V(s, \theta)$ of the state-value function using the learning result of the first step before the start of learning (the initial policy in Figure 3 is generated from this $V(s, \theta)$). The second is to update $V(s, \theta)$ repeatedly during learning in consideration of the learning data from the first step. The details are given below.

a) Initialization of approximation function: In the first step learning, RL tasks are defined for each SFC, and learning is executed. Therefore, as many approximation functions are generated as the number of SFCs after learning. We define $V_j(s, \theta)$ as an approximation function of SFC_j. The initial state of the second RL step is decided by $V_j(s, \theta)$. That is, we predict $V(s, \theta)$ with $\sum_{\mathbf{SFCI}} \frac{V_j(s',\theta)}{cost}$ and decide the initial state s_0 as $argmax_{s\in S} V(s, \theta)$ (if it is difficult to obtain the maximum value analytically based on the property of the approximate function, it is replaced with an approximate solution). The s' is a state vector for $V_j(s, \theta)$ that is replaced with SRC and DST of the second step task with the location corresponding to the first step.

Next, we generate a multidimensional normal distribution whose dimension is $dims$ and center is s_0. Based on this probability density function, we sample an arbitrary number of states. Sampled states are defined as $T = \{t_i | i = 1, .., N\}$ (An actual state vector must be discrete. Therefore, it will be discretized as appropriate). The initial policy is generated from $V_{init}(s, \theta)$ obtained from supervised-learning with data D1 $< T, R_T >$. The R_T is the set of prediction values from $V(t_i, \theta)$.

b) Update of approximation function: The RL agent updates $V_{init}(s, \theta)$ with the training data set D2 $< \mathbf{S}, R_{\mathbf{S}} >$ obtained in the second step then, updates the updated $V(s, \theta)$ repeatedly with the newly obtained training data for every episode. While updating, we basically emphasize data D2 for predicting the unsearched state but D1 is also used as required. To use the above idea, we take into account the weight of the data for prediction (since learning of regression models using weighted data is common, this is an updating method that does not specialize in a specific approximation function). We introduce the weight vector w for D1 as follows.

$$w_n = \prod_{m=1}^{M} (1 - exp(-|s_{1n} - s_{2m}|)) \tag{4}$$

where learning data $D1 = \{(s_{1n}, R_{s_{1n}}) | n = 1, .., N\}$,$D2 = \{(s_{2m}, R_{s_{2m}}) | m = 1, .., M\}$, and $|s_{1n} - s_{2m}|$ is the Euclidean distance between s_{1n} and s_{2m}. To prevent being affected by the scale of state-vector element value, each value is normalized with the average and variance of the set of s_{1n}. As an intuitive explanation of the above equation, this means that the distant data do not change the weight (as 1), and the weight becomes close to 0 (replaced with D2) for closer data. The approximation function is updated with the weight vector that combines w_n with the weight vector for D2, all elements of which are one $(1, .., 1)$.

IV. Evaluation

A. Environment and scenario

We evaluated the proposed method and conventional method through three network environments with the mixture of multiple kinds of traffic. Three network environments have different spec to compare the environment-adaptability. Furthermore, to make the environment closer to reality, each traffic has different network path, SFC and performance requirement.

We first give an overview of the evaluation environment. The hardware we used consisted of 18 rack-mounted servers (Intel (R) Xeon (R) CPU E5504 @2.00 GHz×8/16 GB of RAM/1-Gbps Ethernet) and a network switch that connects these servers. We installed OpenStack [24] on them and built simulated network environments, i.e., WAN, Internet and local network, at each location by using Neutron, which is a component of OpenStack. We used VyOS [25], which is open-source software router, to connect between these networks. VyOS has a latency emulation function for network verification where it gives an arbitrary delay to packets output from specific network interfaces. In the evaluation, this function was used to emulate the delay of the WAN by a setting a 10 msec delay on the WAN-side interface of the router installed at the boundary between the WAN and location and 20-msec delay at the boundary interface with the virtual Internet environment. The bandwidth of the WAN was 1 Gbps (the delay of WAN changed by varying the scenarios in the second step presented in IV-C). We also set up five locations, i.e., a head office, three branch offices, and DC, because we assumed a medium-sized enterprise network environment. The maximum amount of CPU and memory usage (the number of CPU cores/memory (GB)) allocated to the POP at each location is head office (4 cores/8 GB), branch office 1 (2 cores/2 GB), branch office 2 (1 core/2 GB), and branch office 3 (0.5 core/1 GB). We assumed the DC had unlimited resources regarding both CPU and memory.

VNFs were built as a Docker [26] container that can allocate resources rapidly and flexibly. The step size of resource-allocation change was 0.05 cores on the CPU and 100 MByte on the memory. The details of VNFs and SFCs used in the evaluation are shown in Table II and Table III, respectively. The generated traffic in the evaluation was intended for a Web application placed on the DC and for data-transfer service hosted via the Internet. At the end of each RL action, we measured the performance metrics (throughput and latency) with Apache JMeter [29] and calculated the rewards.

TABLE II
Virtualized network functions (VNFs)

No.	name	description
1	Proxy	Cache server built with Nginx [27].
2	FW	Built with Snort [28]. Defined by about 100 drop rules and start up with in-line (IPS) mode.
3	IDS	Built with Snort using published registered rule.

TABLE III
SFCs

No.	VNF chain	description
1	Proxy-FW-IDS	For in-house website (almost static) connection.
2	FW-IDS	For internal and external website connection.
3	IDS-FW	For public cloud service connection.

We evaluated environment adaptability of the proposed method on three scenarios (scenarios 1, 2 and 3) in the second RL step. These scenarios differed in substrate network specification of the WAN as shown in Table IV. The list of SFC instances and cost factor are shown in Table V and VI, respectively. The size of the exploration spaces of these scenarios was approximately 10^{20}. We implemented a python [30] module that controls executions of RL running on a VM deployed on the OpenStack environment. This module also controls the agents, updates the approximation function, and calculates rewards in the manner we presented in the previous section.

B. First RL step

In the first RL step, we evaluated the learning degree of performance characteristics by the comparing a learning model

TABLE IV
WAN SPECIFICATION FOR EACH SCENARIO IN SECOND STEP

scenario	1	2	3
bandwidth (Gbps)	1.0	1.0	0.5
latency (round trip, msec)	10	20	40

TABLE V
SFC INSTANCES AND PERFORMANCE REQUIREMENTS IN SECOND STEP

SFC instance	SFC	SRC	DST	throughput (req/sec)	latency (msec)
1	1	Head office	DC	11.0	200
2	1	Branch1	DC	10.5	250
3	1	Branch2	DC	9.5	250
4	1	Branch3	DC	9.0	250
5	2	Head office	DC	9.5	400
6	2	Branch1	DC	10.5	400
7	2	Internet	DC	5.0	600
8	3	DC	Internet	0.3	4000

using three approximation functions, i.e., linear approximation based on tile coding [31] (CMAC), gradient boosting decision tree (GBDT) [32] [33], and multi-layer perceptron (MLP) [34]. CMAC is one of the most basic state coding method used with the linear function approximation. GBDT is a decision tree algorithm with high accuracy [13], and MLP is a neural network model. Basically, the meta-parameters of these approximation functions were determined from cross-validation. The validation range of the main parameters for each approximation function is as follows: tile size (CMAC) is 0.25 (value of each element of a feature vector is normalized with $[0, 1]$); number of layers (CMAC) is 4, offset (CMAC) is 0.2; max tree depth (GBDT) is chosen from $[3, 5, 8, 12]$; number of hidden layers (MLP) is chosen from $[3, 4, 5]$; and unit size (MLP) is 50 or 100. The reward function shown in III-B4, which is based on the performance of each SFC, was used for this step. The iteration length of an episode was fixed as 20 actions. The inverse temperature parameter τ while exploring was 1.0, and the δ of the discount rate for calculating returns was set to 0.2. The meta-parameters of each learning algorithm were determined using the evaluation results of preliminary experiments conducted under the same conditions as those in first RL step. Figure 4 illustrates the results from the first step for each SFC (performance metrics are plotted by moving average of past five episodes for smoothing the graph). As the episode advanced, the learning with GBDT indicated throughput increase and latency decrease for all SFCs, although there was some stochastic fluctuation. To evaluate prediction accuracy of approximation functions, Table VII shows the mean coefficients of determination R2s in cross validation. The R2 is defined as $1 - \frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{N}(y_i - \bar{y})^2}$, where N is sample size, y_i is the observed value, $\hat{y}_i$ is the predicted value, and $\bar{y}$ is the average of y_i. The R2 is 1.0 when all observed values are predicted completely and 0 when all prediction values are average values. The cross-validation method was k-fold [35], and the number of folds was 5. We combined the data obtained from learning with each approximate function, and used the data as evaluation data. These results indicate that GBDT approximation had the highest R2. Since the other approximation functions, CMAC and MLP, could not predict SFC performance, their R2s were negative value (it means it is worse than predicting all value as the average). Therefore, we confirmed that the learning algorithm using GBDT as an approximation function could acquire well-learned policy reflecting SFC performance characteristics. This is because the real state-value function of this task that contains dynamically moving bottle necks is fitted to the GBDT suitable for approximating nonlinear and noncontiguous functions. On the other hand, since the other approximation functions required more data to learn SFC dynamics, it seemed that learning did not proceed. In fact, we observed that the learning using CMAC progressed after more than 500 episodes on RL task of SFC 1 in the preliminary experiment. Although we could not observe a progress with MLP, it seems that more explorations are required for convergence of many parameters. In response to this, we conducted an evaluation for the second step by using GBDT as the approximate function.

TABLE VI
COST FACTORS IN SECOND STEP

location	head office	branch1	branch2	branch3	DC
CPU/core:w_{cpu}	1.1	1.2	1.2	1.2	1.0
MEM/GB:w_{mem}	1.1	1.2	1.2	1.2	1.0

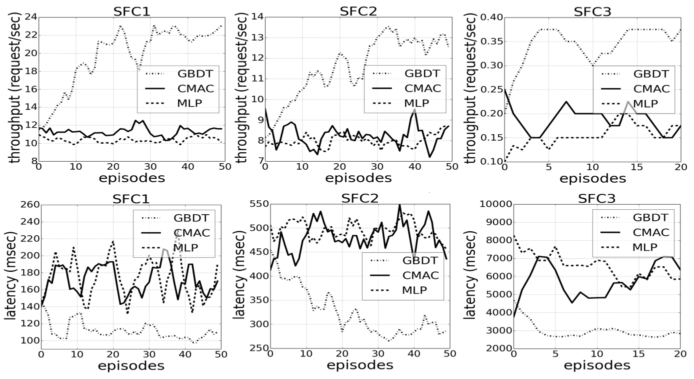

Fig. 4. Comparison of different approximation functions in first step

TABLE VII
MEAN COEFFICIENTS OF DETERMINATION (R2) IN CROSS VALIDATION

SFC	GBDT	CMAC	MLP
SFC1	0.6213	-3.760×10^{28}	-0.8192
SFC2	0.7476	-1.938×10^{30}	-4.695
SFC3	0.9740	-1.308×10^{31}	-3.306

C. Second RL step

In the evaluation of the second RL step we evaluated three scenarios (scenarios 1, 2 and 3). These scenarios differed in substrate network specifications. Additionally, along with comparing learning speed, we evaluated the RL policy learned only

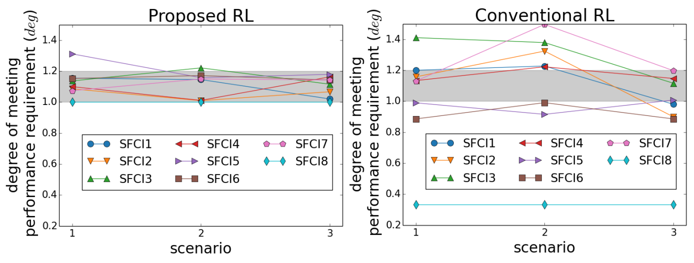

Fig. 5. Distribution of degree of meeting requirements after 50 episodes.

from D2 (data obtained in second step). This task involved a conventional RL without prior knowledge from the first step. The learning meta-parameters were the same as in the first step, but we used the user requirement-based reward function discussed in III-B4 ($K = 100$, $d = 0.8$, $b = 1.1$). Figure 5 shows the degree of meeting the performance requirement (deg) that we discussed in III-B4 after 50 episodes (50 episode corresponds to 1,000 explorations and about 17 hrs if one exploration takes 30 sec). The area colored with gray in Figure 5 corresponds to the desirable range of deg. That is, deg is desired to be not only greater than 1.0 but also less than 1.2, and if deg is too large, it means over allocation of computing resources. Though the upper of desirable deg should be determined based on probabilistic fluctuation range of deg, we set it to 1.2 from the preliminary experiments (therefore, reward function parameter b is set to 1.1 as the center of the range). With the proposed RL method, all deg values are greater than 1.0. It means that all SFC instances met the performance requirements shown in Table V in all scenarios. Furthermore, almost all deg are less than 1.2 except SFC instance 5 in scenario 1 and SFC instance 3 in scenario 2. Whereas, there was no SFC instance which deg is within the desirable range through all scenarios in the conventional RL. This result shows that the proposed method could learn the proper configuration environmental adaptively.

Figure 6 illustrates how the RL progressed. The tile color corresponding to the SFC instance, and the episode indicates deg (the lighter the color, the higher the deg, and if deg is greater than 1.0, it is regarded as 1.0). The proposed RL method learned to control placements and resource allocations to meet the given performance requirements as an episode progressed. Whereas, the conventional RL without using the knowledge from the first step could not progress learning within 50 episodes. This is because that since the proposed RL uses the knowledge from the first step, the initial state is relatively near the optimal configuration, additionally, it can explore efficiently. We continued using the conventional RL until episodes 500 (10,000 explorations) for each scenario, however, learning hardly progressed. This suggests that with the conventional RL method, it is difficult to learn the proper configuration on a real network in which the number of explorations is limited.

Figure 7 shows trends in resource cost defined in III-B4 regarding computing resource consumption. Despite the fact that

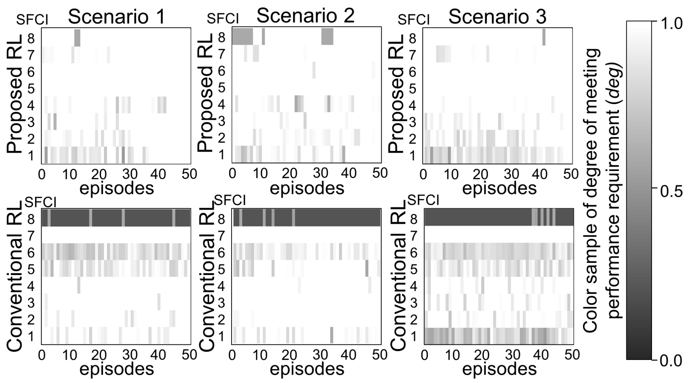

Fig. 6. Trends in degree of meeting requirements in second step

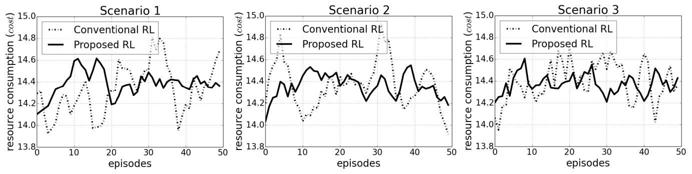

Fig. 7. Trends in amount of resource consumption in second step

additional resources were necessary for some SFC instances to satisfy the requirements, the proposed method did not always increase the amount of consumption in all scenarios. This means that our method not only adds resources but also reduces them if possible. Hence, our method can also learn a cost-efficient control policy.

V. Conclusion

We proposed a method for solving VNF resource sizing and placement problems using accelerated RL divided with two-steps depending on the generality of the learning target. With the proposed method, VNF the resource sizing and placement problem of SFCs was formulated as a RL task. We also presented an effective technique that uses the knowledge about the performance dynamics of SFCs to learn a practical RL task with individual environment and requirement by weighting and mixing learning data.

We showed that the learning using GBDT as a state-value approximation function converges faster with the proposed method. Moreover, for three scenarios with different network environments, we showed that the proposed RL method can learn the cost-efficient configuration that meets the performance requirements within only 1,000 explorations, whereas, general RL cannot even if it takes 10,000 explorations. This is because that the first step learning reduce actual exploration space in the second step in advance. Additionally, the method requires all SFCs available to be learned in the first step.

The major future work is prospecting the environment-adaptability of the proposed method more precisely and improve it. For that, we will need additional evaluation to measure how much the method can handle fluctuations of environment and requirements.

References

[1] ETSI ISG for NFV Home Page, https://www.etsi.org/technologies-clusters/technologies/nfv

[2] Open Source MANO, https://osm.etsi.org/

[3] Cloudify, https://cloudify.co/

[4] Enterprise Service Automation (ESA), https://www.cisco.com/c/en/us/products/collateral/cloud-systems-management/application-policy-infrastructure-controller-enterprise-module/datasheet-c78-736830.html

[5] Sutton, Richard S., and Andrew G. Barto. Reinforcement learning: An introduction. Vol. 1. No. 1. Cambridge: MIT press, 1998.

[6] Behringer, Michael, et al. Autonomic networking: Definitions and design goals. No. RFC 7575. 2015.

[7] Craven, Robert, et al. ”Policy refinement: Decomposition and operationalization for dynamic domains.” Proceedings of the 7th International Conference on Network and Services Management. International Federation for Information Processing, 2011.

[8] A. Fischer, J. F. Botero, M. T. Beck, H. de Meer and X. Hesselbach, ”Virtual Network Embedding: A Survey,” in IEEE Communications Surveys & Tutorials, vol. 15, no. 4, pp. 1888-1906, Fourth Quarter 2013.

[9] J. Gil Herrera and J. F. Botero, ”Resource Allocation in NFV: A Comprehensive Survey,” in IEEE Transactions on Network and Service Management, vol. 13, no. 3, pp. 518-532, Sept. 2016.

[10] M. C. Luizelli, L. R. Bays, L. S. Buriol, M. P. Barcellos and L. P. Gaspary, ”Piecing together the NFV provisioning puzzle: Efficient placement and chaining of virtual network functions,” 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM), Ottawa, ON, 2015, pp. 98-106.

[11] N. Tastevin, M. Obadia and M. Bouet, ”A graph approach to placement of Service Functions Chains,” 2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM), Lisbon, 2017, pp. 134-141.

[12] G. A. Gallardo, B. Baynat and T. Begin, ”Performance Modeling of Virtual Switching Systems,” 2016 IEEE 24th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS), London, 2016, pp. 125-134.

[13] T. Lei, Y. Hsu, I. Wang and C. H. -. Wen, ”Deploying QoS-assured service function chains with stochastic prediction models on VNF latency,” 2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), Berlin, 2017, pp. 1-6.

[14] L. Gupta, M. Samaka, R. Jain, A. Erbad, D. Bhamare and C. Metz, ”COLAP: A predictive framework for service function chain placement in a multi-cloud environment,” 2017 IEEE 7th Annual Computing and Communication Workshop and Conference (CCWC), Las Vegas, NV, 2017, pp. 1-9.

[15] J. Rao, X. Bu, C. Z. Xu and K. Wang, ”A Distributed Self-Learning Approach for Elastic Provisioning of Virtualized Cloud Resources,” 2011 IEEE 19th Annual International Symposium on Modelling, Analysis, and Simulation of Computer and Tele-communication Systems, Singapore, 2011, pp. 45-54.

[16] Rao, Jia, et al. ”VCONF: a reinforcement learning approach to virtual machines auto-configuration.” Proceedings of the 6th international conference on Autonomic computing. ACM, 2009.

[17] P. Tang, F. Li, W. Zhou, W. Hu and L. Yang, ”Efficient Auto-Scaling Approach in the Telco Cloud Using Self-Learning Algorithm,” 2015 IEEE Global Communications Conference (GLOBECOM), San Diego, CA, 2015, pp. 1-6.

[18] S. I. Kim and H. S. Kim, ”A research on dynamic service function chaining based on reinforcement learning using resource usage,” 2017 Ninth International Conference on Ubiquitous and Future Networks (ICUFN), Milan, 2017, pp. 582-586.

[19] Watkins, Christopher JCH, and Peter Dayan. ”Q-learning.” Machine learning 8.3-4 (1992): 279-292.

[20] R. Mijumbi, J. L. Gorricho, J. Serrat, M. Claeys, F. De Turck and S. Latré, ”Design and evaluation of learning algorithms for dynamic resource management in virtual networks,” 2014 IEEE Network Operations and Management Symposium (NOMS), Krakow, 2014, pp. 1-9.

[21] Doucet, Arnaud, Nando De Freitas, and Neil Gordon. ”An introduction to sequential Monte Carlo methods.” Sequential Monte Carlo methods in practice. Springer, New York, NY, 2001. 3-14.

[22] Rummery, Gavin A., and Mahesan Niranjan. On-line Q-learning using connectionist systems. Vol. 37. University of Cambridge, Department of Engineering, 1994.

[23] Kaelbling, Leslie Pack, Michael L. Littman, and Andrew W. Moore. ”Reinforcement learning: A survey.” Journal of artificial intelligence research 4 (1996): 237-285.

[24] OpenStack, https://www.openstack.org/

[25] VyOS, https://vyos.io/

[26] Docker, https://www.docker.com

[27] Nginx, https://www.nginx.com/

[28] Snort, https://www.snort.org/

[29] Apache JMeter, https://jmeter.apache.org/

[30] Python, https://www.python.org/

[31] Albus, James S. ”A new approach to manipulator control: The cerebellar model articulation controller (CMAC).” Journal of Dynamic Systems, Measurement, and Control 97.3 (1975): 220-227.

[32] Friedman, Jerome H. ”Greedy function approximation: a gradient boosting machine.” Annals of statistics (2001): 1189-1232.

[33] Chen, Tianqi, and Carlos Guestrin. ”Xgboost: A scalable tree boosting system.” Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining. ACM, 2016.

[34] Lippmann, Richard. ”An introduction to computing with neural nets.” IEEE Assp magazine 4.2 (1987): 4-22.

[35] Stone, Mervyn. ”Cross-validatory choice and assessment of statistical predictions.” Journal of the royal statistical society. Series B (Methodological) (1974): 111-147.