QoS-aware SFC Migration Scheduling Based on Encoder-Decoder RNN for Cloud-Native Platform

Takahiro Hirayama, Masahiro Jibiki, Takaya Miyazawa, and Ved P. Kafle
National Institute of Information and Communications Technology
{hirayama, jibiki, takaya, kafle}@nict.go.jp

***Abstract*—Service function chaining (SFC) provides the platform for flexible resource management by dynamically allocating resources to virtual and/or container network functions (VNFs/CNFs). To meet the quality of service (QoS) requirements while facing increasing resource demands, the system will require the migration of the VNFs/CNFs from the current server to the others that offer sufficient resources. In this study, we formulate an integer linear programming (ILP) based optimization model to solve the function migration scheduling problem so that it meets QoS requirements of each service function (SF) chain. The remarkable points of this work are the following two points. The one is that we consider latency between VNFs/CNFs belonging to an SF chain, avoiding overhead due to their unnecessary migration and resource shortage. And the other is that we consider the case in which each VNF/CNF must be to be deployed strictly to a designated virtual machine (or container). To reduce complexity, we apply an encoder-decoder recurrent neural network (ED-RNN) as a machine learning model to the function migration scheduling problem. Performance evaluations show that the ED-RNN based approach achieves a similar performance as the ILP, while adding the benefits of very low complexity.**

Index terms*—Service function chaining (SFC), Integer linear programming (ILP), Machine learning (ML), Recurrent neural network (RNN), Cloud-native platform.

I. Introduction

The 5th generation (5G) or beyond 5G (B5G) mobile communication systems are expected to provide many types of application services such as virtual reality and autonomous vehicles over a single network infrastructure. To enable network operators to effectively provide diverse services over the same network, the systems must be reconfigurable by software. For this purpose, service function chaining (SFC), software defined networking (SDN), and network function virtualization (NFV) are promising platform technologies [1].

A service function (SF) chain contains a series of network functions (NFs) such as load balancers, and firewalls, and they can be deployed in virtual machines (VMs) as virtual NFs (VNFs) or in smaller containers as container NFs (CNFs). On a cloud-native network service platform, services can easily be scaled up or down with demand. However, implementing NFs with smaller granularity and deploying them in many microcontainers makes the system architecture more complex to operate and monitor. Furthermore, operators should determine NF placement with considering QoS requirement of each SF chain and the constraints of the cases when SF chains are deployed on the cloud-native platform. To maintain the diverse QoS of SF chains, network operators should consider multiple factors those impact to performance of all NFs belonging to a chain, for example, latency between NFs in the same chain, overhead due to NF migration, and resource competition among NFs on the same server.

The NF placement, resource allocation and migration scheduling problems for the monolithic deployment of NFs have been studied recently [2,3], but the techniques applied are limited in providing agile operations of NF migration as they require a significant amount of time to complete an operation cycle. Meanwhile, machine learning (ML) techniques are expected to be capable of meeting the diverse QoS requirements by autonomous and proactive resource control and management predicting time-varying traffic demands. Application of ML techniques has also been presented in several prior studies. The authors in [4,5] proposed SF chaining using multiple distributed CNFs, and discussed traffic steering such as load balancing. However, they did not address the issues of proactive and sophisticated resource adjustments using ML technologies.

To adapt to the situation of dynamic NF migration in SFC, the prior study has applied the encoder-decoder recurrent neural network (ED-RNN) model to tackle the migration scheduling problem [6]. However, it is limited in that it does not considered the prominent feature of cloud-native platform features, such as the coexistence of VNFs and CNFs and the diversity of QoS requirements of SF chains. Therefore, in the work presented in this paper, we propose an NF migration scheduling solution suited for cloud-native platforms where services are deployed and provided in containers created on VMs meeting the desired QoS requirement of each service.

The main contributions of this paper are as follows:

- We formulate an NF migration scheduling problem for cloud-native platform as an integer linear programming (ILP) with the objective to meet QoS requirements of every SF chain. We determine its objective function as to minimize the uncomfortability of SF chains, by avoiding unnecessarily frequent NF migration while guaranteeing low latency. Furthermore, our formulation includes the NF placement constraint that each VNF or CNF is designated to be allocated to a VM or a container, respectively.
- We show that the ED-RNN architecture has the possibility to be used for NF migration scheduling in a cloud-native platform. Through computer simulation, we verify that the proposed method can reduce the occurrences of unnecessary NF migration and that it has much lower computation complexity.

This paper is organized as follows. In Sec. II, we introduce our SF migration framework. The NF migration problem is discussed in Sec. III and the ED-RNN model architecture is described in Sec. IV. Performance evaluations are presented in Sec. V. Finally, the conclusions are presented in Sec. VI.

978-3-903176-51-5 © 2022 IFIP

II. Autonomic Service Funtion Migration Scheme

Our assumptions regarding the architecture of the network service platform are as follows. Infrastructure providers (IPs) construct their networks by connecting servers (nodes) that can deploy several NFs in VMs. Application service providers (ASPs) borrow an appropriate number of NFs from the IPs according to their service types and scales. When an IP receives a request for the construction of a network service from an ASP, it constructs and manages the SF chains of the desired service. If sufficient computational resources are exhausted due to increased demand, the NF should be migrated from the current node to another. The IP must determine the destination server node from many available ones. Solving this problem using ILP in a short time is prohibitively difficult because of its complexity, which is the main reason for adopting an ML based approach for this type of problem.

The advantage of our NF migration scheduling leveraging ML techniques is as follows. In the given network infrastructure, the ML model autonomously creates NF migration plans in accordance with the utilization of computational resources in the current state, QoS requirements of the NFs, position dependencies of the NFs per SF chain, and predicted future resource demands. If a certain NF is migrated from one node to another too often, the service deployed on the NF will experience frequent disruptions. Therefore, it would be optimal if a certain NF once migrated to a node is not migrated again to other nodes for a while. Our previous work revealed the efficacy of the ED-RNN in NF migration scheduling on a simple multihomed topology [6]. However, it is still necessary to consider more complex situations of cloud-native platform, where each server can accommodate NFs as service applications deployed on VMs and/or containers (within another VM). Notably, if an NF does not strictly require a specific operating system distribution, the NF can be deployed as an application on any container.

In this study, we validate that the ED-RNN, which was introduced in our previous work for NF migration scheduling, is also effective for the NF migration scheduling in the cloud-native platform. Related studies have applied ML-based approaches to the SFC placement problem before, but they mainly use deep reinforcement learning (DRL) [7] to find the optimal solutions to maximize the reward function. However, DRL requires a very long time to train complicated models, and is probably incapable of dealing with the rapidly time-varying resource utilization. Our work differs from their approaches by applying a simple neural network trained by the solutions obtained by the ILP.

III. Problem Statement and Formulation

Fig. 1 displays a leaf-spine topology with $N = 4$ servers to set the stage for the problem formulation and its subsequent evaluation. In this section, we elucidate the VNF scheduling problem for SF chains deployed on a cloud-native platform and its ILP solution. We describe the ED-RNN training process using the ILP solution in Sec. IV.

In Fig. 1, an SF chain (SFC1) is composed of three NFs (NF1-1, NF1-2, and NF1-3) deployed on three servers. NF1-1 and NF1-3 are deployed on VMs (as VNFs), and NF1-2 is installed on a container (as a CNF). In our example, because

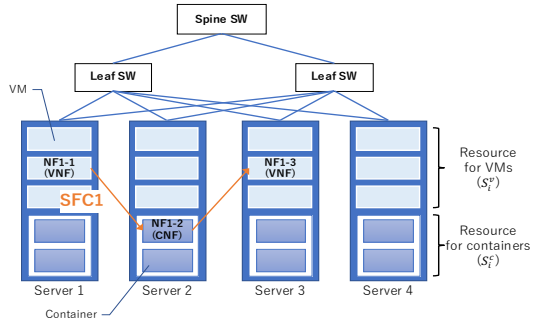

Fig. 1 Network topology (N=4 servers).

all functions are installed on different servers, the packet transmission cost induced by their encapsulation and decapsulation overhead and propagation delays between servers must not be ignored. The architecture for application working on a container is more complex than the cases of applications working on VMs. Therefore, the transmission cost of any SF chain increases when it includes many NFs deployed as CNFs. The propagation delay is proportional to the hop count between servers.

To formulate the problem, we denote the computational resources (such as CPU) of server i $(i = 1,2,\dots,N)$ allocated to VNFs as S_i^v and to CNFs as S_i^c. There are F SF chains, and each SF chain c_f $(f = 1,2,\dots,F)$ consists of several NFs. The h-th NF of SF chain c_f is denoted as $v_{f,h}$, where $h \in 1,2,\dots,H_f$ and H_f denotes the hop count of chain c_f. The demands of NF computing resources dynamically change at each time slot $(t = t_1, t_2, \dots, t_T)$. Thus, to simplify the problem, we assume that the orchestrator can accurately predict the changes in the resource demands of all SF chains during each time slot. Therefore, the appropriate placement positions of the NFs can change significantly. Some NFs are required to be migrated from one server to another when the total resource demand of NFs deployed on the server exceeds the server's total capacity. Notably, the QoS requirements of each SF chain differ from those of the others. When severe resource competition occurs, it is better to migrate the NFs of a disruption-insensitive service.

Each SF chain has unique parametric requirements. The uncomfortability of c_f at time t, $U(c_f, t)$, is defined as:

$$\boldsymbol{U(c_f,t) = \alpha_f G(c_f,t) + \beta_f M(c_f,t) + \gamma_f L(c_f,t)} \quad (1)$$

where $G(c_f,t)$, $M(c_f,t)$, and $L(c_f,t)$ represent the performance degradations induced by the resource shortages of servers (i.e., due to the gap between capacity and demand), the temporal service disruptions induced by NF migration, and the transmission costs (latency) of NF packet forwarding, respectively. Their concise definitions of $G(c_f,t)$, $M(c_f,t)$ and $L(c_f,t)$ are provided later. The parameters α_f, β_f and γ_f are coefficients exhibiting the importance of these metrics in the SF chain c_f. We formulate the optimization problem of minimizing the sum of the uncomfortability values.

To formulate the NF scheduling problem as an ILP problem, we define several variables, as follows:

S: Set of servers. $|S| = N \times 2$ because server i includes S_i^v and S_i^c.

V: Set of NFs. $|V| = \Sigma_{f \in F} H_f$.

$x(s,v,t) \in 0,1$: Binary variable indicating whether server s hosts NF-v at time t. $s \in S, v \in V$.

$d_v(t) \geq 0$: Demand level of NF-v at time t (input parameters, constant real number). To simplify, values of $d_v(t)$ have only three values, high (H), middle (M), and low (L).

$C_s > 0$: Capacity of server $s \in S$ (input parameter, constant integer).

$g(s,t) \geq 0$: Variable representing the resource shortage amount in server i at time t (real number). $g(s,t) = 0$ indicates no shortage.

$m(s,v,t) \in 0,1$: Binary variable indicating whether NF-v is migrated at time t or not.

$l(s,s') \geq 0$: Variable representing the latency between a pair of servers (s,s') (input parameter, constant real number). $s, s' \in S$.

$r(v,v') \in 0,1$: Binary variable indicating whether a pair of adjacent NFs (v,v') is deployed in the path of SF chain c_f or not. $v, v' \in V$.

$p(v) \in 0,1$ ($\neg p(v) \in 0,1$): Binary variable indicating whether NF-v must be deployed as VNFs (CNFs) (input parameter). Details are described later.

We formulate the NF migration scheduling problem as follows.

Objective:

$$\boldsymbol{minimize} \sum_{t \in t_1,\ldots,t_T} \sum_{f \in F} \boldsymbol{U}(\boldsymbol{c_f}, \boldsymbol{t}) \tag{2}$$

subject to:

$$\sum_{s \in S} x(s,v,t) = 1, \forall v \in V, \forall t, \tag{3}$$

$$\sum_{v \in V} x(s,v,t) \cdot d_v(t) - C_s < g(s,t), \forall s \in S, \forall t, \tag{4}$$

$$m(s,v,t_k) \geq x(s,v,t_k) - x(s,v,t_{k-1}), \quad \forall s \in S, \forall v \in V, \forall k \in 2,3,\ldots,T. \tag{5}$$

Fig. 2 illustrates the relationship between these constraints. In this figure, the upper two tables represent the values of $x(s,v,t)$ at times t_1 and t_2. Thus, it denotes the server that accommodates NF-v at time t_1 (and t_2). Similarly, the lower table represents the values of $d_v(t)$, which denote the resource demands of SF chains at time $t = t_1, t_2, \ldots, t_T$. The constraint of Eq. (3) indicates that each NF must be allocated to only one server. In Eq. (4), the resource shortage value, $g(s,t)$, becomes larger than zero only when the sum of the demands in the SF chains on server s exceeds its capacity, C_s, at time t. Eq. (5) denotes whether an NF is migrated from one server to another or not, at time t_k. The value of $m(s,v,t_k)$ equals one when NF-v is migrated to server s. For example, when NF-$v_{1,2}$ is migrated from server S_1^v to server S_2^v, $x(S_2^v, v_{1,2}, t_2) - x(S_2^v, v_{1,2}, t_1) = 1$, as shown in Fig. 2. If NF-v is not migrated at time t_k, $m(s,v,t_k)$ remains zero.

As described, the objective function (Eq. (2)) includes three factors, $G(c_f,t)$, $M(c_f,t)$ and $L(c_f,t)$. The metric of the resource shortage affecting the chain c_f, $G(c_f,t)$, is defined as

$$\boldsymbol{G}(\boldsymbol{c_f}, \boldsymbol{t}) = \sum_{s \in S} \sum_{v \in c_f} \boldsymbol{g}(\boldsymbol{s}, \boldsymbol{t}) \cdot \boldsymbol{x}(\boldsymbol{s}, \boldsymbol{v}, \boldsymbol{t}). \tag{6}$$

The value of $G(c_f,t)$ increases if NF-v is deployed on server s at time t, and the amount of resource demand exceeds the capacity of server s. $M(c_f,t)$ represents the frequency of migration of VNFs in c_f at time t. The metric of the resource shortage affecting the chain c_f, $G(c_f,t)$, is defined as

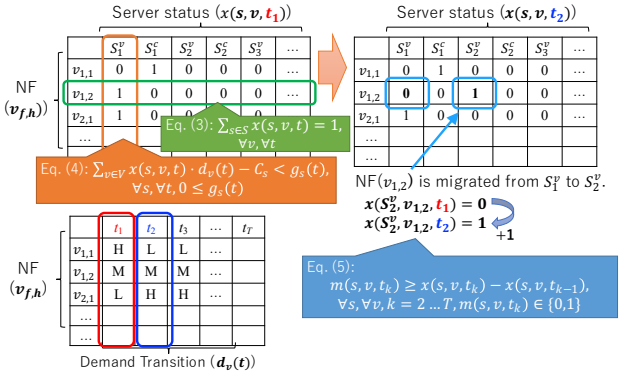

Fig. 2 Illustration of constraints of the VNF(CNF) scheduling problem.

$$\boldsymbol{M}(\boldsymbol{c_f}, \boldsymbol{t}) = \sum_{s \in S} \sum_{v \in c_f} \boldsymbol{m}(\boldsymbol{s}, \boldsymbol{v}, \boldsymbol{t}). \tag{7}$$

Finally, $L(c_f,t)$ indicates the total transmission latency through the NFs comprising the SF chain c_f at time t. It is defined by

$$\boldsymbol{L}(\boldsymbol{c_f}, \boldsymbol{t}) = \sum_{s_a \in S} \sum_{s_b \in S} \sum_{v_{f,g} \in c_f} \sum_{v_{f,g'} \in c_f} \boldsymbol{L'}, \quad L' = l(s_a, s_b) \cdot x(s_a, v_{f,h}.t) \cdot x(s_b, v_{f,h'}, t) \cdot r(v_{f,g}, v_{f,h'}). \tag{8}$$

In Eq. (8), the transmission cost from server s_a to s_b is included in the value of $L(c_f,t)$ if NF-$v_{f,h}$ and NF-$v_{f,h'}$ are adjacent pairs of VNFs comprising the SF chain c_f and NF-$v_{f,h}(v_{f,h'})$ is deployed on server $s_a(s_b)$.

As mentioned, NFs should be deployed in VMs or containers according to their requirements when considering the features of the cloud-native platform. Thus, if an NF-v is marked for deployment on a VM ($p(v) = 1$), it should be installed in a VM. We replace Eq. (3) using the following detailed constraints:

$$\begin{aligned} &\sum_{i \in N} x(S_i^v, v, t) + \sum_{i \in N} x(S_i^c, v, t) = 1, \\ &\sum_{i \in N} x(s_i^v, v, t) = p(v), \\ &\sum_{i \in N} x(s_i^c, v, t) = \neg p(v), \\ &p(v) + (\neg p(v)) = 1, \forall v, \forall t. \end{aligned} \tag{9}$$

This set of equations denotes the following constraints: First, NF-v must be deployed on at least one server. Second, NF-v should be installed in a VNF if $p(v) = 1$. Third, NF-v should be installed in a CNF if $\neg p(v) = 1$. Finally, $p(v)$ and $\neg p(v)$ exhibit an exclusive relationship.

In summary, solving this problem clarifies the best combination of $x(s,v,t)$ values. Thus, server $s \in S$ should accommodate NF-v ($v \in V$) at each time slot ($t \in t_1, t_2, \ldots, t_T$). The optimal solutions maintain the sum of the uncomfortability metrics of SF chains as low as possible.

IV. ED-RNN Architecture

We trained the ED-RNN model with the solutions of the optimization problem as described in Sec. III. Our NF migration scheduling problem uses the time-series data on the resource demands of SF chains. As clarified by a prior study [6], the ED-RNN architecture is suitable for reducing the frequency of migration because the past state's information is adopted effectively during scheduling. Note that the past state information includes the NF placement and resource utilization at each time slot.

Fig. 3 illustrates the architecture of the ED-RNN model, which is composed of three parts: encoder, decoder, and attention. The encoder reads a certain length of sequential

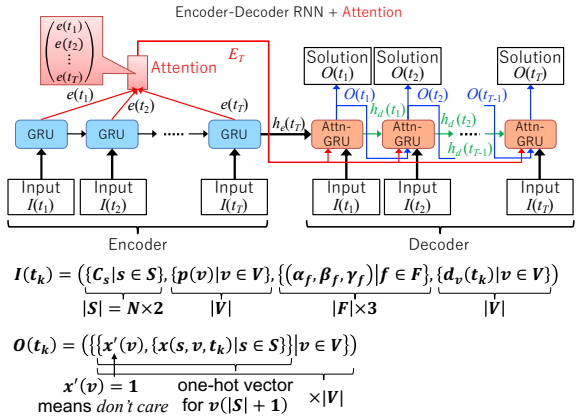

Fig. 3 Encoder-Decorder RNN (ED-RNN) and input/output tensors.

input data. The decoder then processes the input data step-by-step alongside the hidden state retrieved from the gated recurrent unit (GRU) cell, which is used for an RNN to memorize the long- and short-term behaviors of the encoder part. This architecture improves scheduling performance by using both past and future input data. Our VNF/CNF migration scheduling problem also uses sequential input data that include the time series of SFC demands. The encoder reads the input data, $I(t_k)(k = 1,2,\dots,T)$, and the attention part integrates the output data from the encoder at each occasion. The output data from the encoder at each time are gathered at the attention part, which memorizes the $T \times H$ matrix. This part supports the decoder in identifying parameters that have a significant impact on NF scheduling. Our previous work [6] showed that the attention part plays a key role in improving NF placement performance because it handles dynamically changing conditions, i.e., fluctuations in resource demands.

To more effectively use the ED-RNN model architecture, we apply the QoS parameters of each chain and the placement constraint. The input tensor to the encoder at time t_k, $I(t_k)$, is shown in Fig. 3. The input data include the capacity of servers $(C_s, s \in S)$, constraints of NFs deployed as a VNF $(p(v), v \in V)$, QoS requirements of SF chains $(\alpha_f, \beta_f, \gamma_f)$ $(f = 1,2,\dots,F)$, and predicted resource demands of NFs at time t $(d_v(t), v \in V)$. We assume that the demand for NFs can be met by some prediction techniques; thus, we did not focus on these in this study. The size of the input data at time t, is $|S| + |V| + 3F + |V|$. The size of the output data, $e(t)$, is the same as the output size of the hidden layer in GRU H.

The decoder sequentially decides on the placement of NFs at each time slot using the output data from the encoder and the attention. The decoder consists of combined linear and GRU cells. As shown in Fig. 4, at time t_k, the Attn-GRU reads the output data from the encoder $(h_d(t_{k-1}))$, attention part (E_T), and previous slot t_{k-1} $(O(t_{k-1}))$. The group of $|V|$ classifiers produces an NF placement matrix at the time t_k (i.e., $\{x(s,v,t_k)|s \in S\}$). Each classifier thus corresponds to the NF- v placements $(v \in V)$. The NF placement is represented by one-hot vector in which only one element is arranged as one, and the others are forced to be 0. As shown in Fig. 3, the size of each one-hot vector becomes $|S| + 1$ because it also includes the element $x'(v)$, which represents *don't care* (the first element, whose index is 0). This is used to handle cases in which the resource demands of the

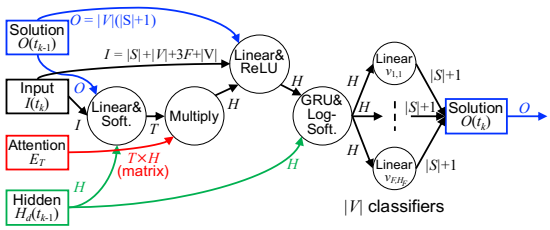

Fig. 4 Attn-GRU architecture.

corresponding NFs are 0. Consequently, the output data comprise a vector of size $|V| \times (|S| + 1)$.

Each one-hot vector output from the classifier exhibits a server that should accommodate the VNF. For example, if the v-th one-hot vector is (0, 0.3, 0.7, 0.4) at time t, then NF-v should be deployed on the second server because the element indexed by 2 (starting from 0) shows the largest value. In some cases, a solution obtained from the ED-RNN is not suitable for meeting the NF requirements, that is, the cases when the server type mismatches its requirements (e.g., S_i^c is selected despite $p(v) = 1$). In these cases, the NF-v is deployed on the server corresponding to the element having the second-largest value in the one-hot vector.

V. Evaluation and Discussion

To train the ED-RNN model, we first generated 10,000 patterns of QoS requirements and time-series data as resource demands in SF chains. We then solved the NF migration scheduling plans for each pattern as ILPs. The number of combinations $x(s,v,t)$ is $2^{|S|+F+T}$. The ranges of the number of active chains, F, and the SF chain length (max: H_f) were set to [3, 8] and [1, 3], respectively. Thus, the maximum value of $|V|$ was $8 \times 3 = 24$. The capacities of S_i^v and S_i^c were 3 and 2, respectively. There were $N = 4$ servers in the network, thus, $|S| = 4 \times 2 = 8$. The number of time slots, T, was 10. The values of $p(v)$ were randomly chosen, and the resource demands of VNFs at each time, t, $(d_v(t))$ were randomly chosen from 0.1 (low), 0.5 (middle), and 1 (high). When the number of active chains and active NFs in the chains was smaller than 8 and 3, respectively, the gap was filled with padding zeros. $l(s_a, s_b)$ was defined as the sum of hop counts and connection overheads. Hop counts are defined by the shortest paths from s_a to s_b in the topology shown in Fig. 1. The connection overheads are 0, 0.4, and 0.6, which correspond to cases in which both of s_a and s_b are in VMs, when one is in a VM and the other is in a container, and when both are in containers, respectively. The QoS requirements of SF chains $(\alpha_f, \beta_f, \gamma_f)$ were randomly selected from ((0.5, 0.2, 0.1), (0.15, 0.05, 0.05), and (0.05, 0.01, 0.01)). These represent expensive, moderate, and low-price services, respectively. We solved the above optimization problem using MATLAB. If the computation time exceeds a 5-min threshold, the solution- seeking process is disrupted, and one of the feasible solutions is chosen as a provisional solution.

In the training data, the output data were converted to a one-hot vector, as described in the previous section. For example, if the NF-v is deployed to server 5, the output vector becomes (0, 0, 0, 0, 0, 1, 0, 0, 0). Note that, the output one-hot vector becomes (1, 0, 0, 0, 0, 0, 0, 0, 0), if the demand of NF-v is 0. The case in which the first element equals 1 means *don't care*. We defined the loss function as the sum of the cross-

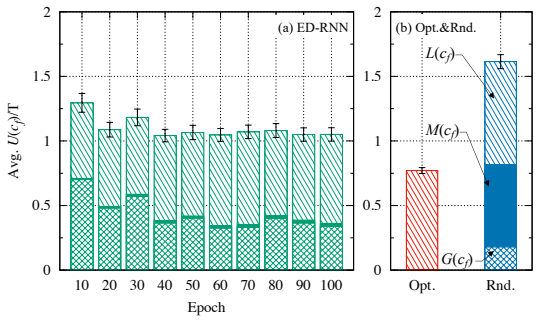

Fig. 5 Average uncomfortability ($U(c_f)$) value comparison of the ED-RNN, the optimal solution and the random placement.

entropy loss values, and the ED-RNN model was implemented using PyTorch. We set the number of input/output sizes from the hidden layers (H) in the GRU cells of the encoder and decoder as H=1,000, which gave the best performance among the other values in [800, 2000].

All 10,000 patterns of training data were used in random order for each epoch. To validate the learning results, we generated another set of 1,000 patterns and their optimal solutions in the same manner. We repeatedly trained and validated the ED-RNN model for 100 epochs. In total, it took approximately 30 minutes on an Intel Core i9-10850K and Nvidia GeForce RTX 3090. To evaluate the performance, we generated an additional 1,000 data patterns.

Fig. 5(a) shows the evaluation results of the scheduling solution obtained from the trained ED-RNN. For comparison, the results obtained by the optimization (Opt.) and random NF placement (Rnd.) are plotted in Fig. 5(b), which shows the average uncomfortability ($U(c_f)$) values. The average values of 1,000 trials and the error bars represent the 95% confidence interval of the average $U(c_f)$. $U(c_f)$ is the sum of $G(c_f)$, $M(c_f)$, and $L(c_f)$. Therefore, this figure represents a breakdown. The values of $G(c_f)$ and $M(c_f)$ are zero in the optimization results; thus, the solution provided by solving the optimization perfectly avoids resource shortages and unnecessary NF migration. Our ED-RNN exhibited the best performance at the 60th epoch as it is effective in avoiding unnecessary NF migrations at the cost of a slightly increased number of resource shortage cases. The migration frequency was lower than that of random placement. As a result, the ED-RNN shows about 30% better performance than random placement when we focus on the sum of metrics, $U(c_f)$.

Fig. 6 shows the average uncomfortability values (i.e., expensive, moderate, and low) of each service type. The values found by the ED-RNN at the 60th epoch were approximately 28%, 30% and 40% larger than those of the optimal solutions, respectively. The results became worse than those of the optimization when we focused on the performance of expensive services. However, the ED-RNN showed remarkable improvement in the cases of expensive services when we compared the results to those of the random placement. The average uncomfortability value was about 40% smaller than that of the random placement.

A performance gap remains between ED-RNN and the optimal results. However, the derivation of optimal solutions takes a longer time, in the worst case, it takes more than an hour. Therefore, to keep pace with dynamically changing network conditions in a short period, it is difficult to optimize

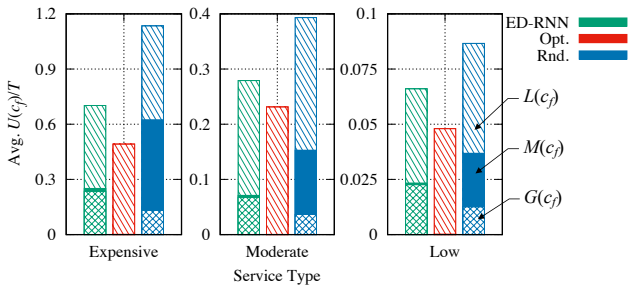

Fig. 6 Average Uncomfortability value by each service type. (The values of "ED-RNN" are obtained by the AI trained at the 60th epoch).

at all. However, the ED-RNN requires only a few seconds to find the solutions, and the evaluation results reflect sufficient generalizability for SFC management in a short period. Therefore, our ED-RNN based approach overcomes the critical issue of the optimization methods. Moreover, the ED-RNN is a practical method of remaining the uncomfortability of SF chains that do not exceed 30% larger than that of the optimization method.

VI. Conclusion

The dynamic adjustment of the computational resources assigned to NFs in a cloud-native platform is essential for the 5G/B5G systems. In this study, we investigated the problem of NF migration scheduling to satisfy the diverse QoS requirements of SF chains against dynamically changing resource demands. We first formulated the NF scheduling problem as an ILP problem, considering cases in which each SF chain has different QoS requirements and when each NF is deployed to designated nodes. We showed that the ED-RNN model trained with the optimal solutions obtained by ILP has high potential to prevent unnecessary NF migrations while guaranteeing low latency, and avoiding resource shortages. In a future work, we will improve the ED-RNN model architecture by applying it to larger topologies and cases with more diverse requirements.

Acknowledgement

This work was conducted as a part of the project entitled “Research and development of infrastructure technologies for innovative virtualization network (JPMI00316),” supported by the Ministry of Internal Affairs and Communications, Japan.

References

[1] A. M. Medhat et al., "Service Function Chaining in Next Generation Networks: State of the Art and Research Challenges," *IEEE Com. Mag.*, vol. 55, pp. 216-223, Feb. 2017.
[2] A. Leivadeas, G. Kesidis, M. Falkner, and I. Lambadaris, "A Graph Partitioning Game Theoretical Approach for the VNF Service Chaining Problem," *IEEE TNSM*, vol. 14, pp. 890-903, Nov. 2017.
[3] T.-W. Kuo et al., "Deploying Chains of Virtual Network Functions: on the Relation Between Link and Server Usage," *IEEE/ACM ToN*, vol. 26, pp. 1562-1576, Aug. 2018.
[4] A. Bouridah et al., "Optimized Scalable SFC Traffic Steering Scheme for Cloud Native based Applications," *IEEE CCNC*, Jan. 2021.
[5] S. D. L. Shah et al., "Cloud-Native Network Slicing Using Software Defined Networking Based Multi-Access Edge Computing: A Survey"," *IEEE Access*, vol. 9, pp. 10903-10924, Jan. 2021.
[6] T. Hirayama et al., "Service Function Migration Scheduling based on Encoder-Decoder Recurrent Neural Network," *IEEE Netsoft*, June 2019.
[7] Y. Bi et al., "Multi-Objective Deep Reinforcement Learning Assisted Service Function Chains Placement," *IEEE TNSM*, vol. 18, pp. 4134-4150, Dec. 2021.