Service Function Migration Scheduling based on Encoder-Decoder Recurrent Neural Network

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

***Abstract*—Service function chaining (SFC) enables network operators to flexibly provide diverse services such as Internet-of-Things (IoT) and mobile applications. SFC technologies are required to offer stable and guaranteed quality-of-service (QoS) even in the circumstances of dynamically-changing resource demands and traffic volumes. To meet QoS requirements against time-varying network environment, infrastructure providers need to dynamically adjust the amount of computational resources such as CPU assigned to virtual network functions (VNFs) in each service function chain. If the increased resource demand of a VNF cannot be satisfied from the available resources in the current node, the VNF should be migrated to another node. Related work has been tackling SFC embedding and scheduling issues by applying various techniques such as game theory, integer linear programming and heuristic methods. However, they have limitations that they cannot provide an agile operation of VNF migration as they require a large number of iterations. In order to overcome the limitations, we propose to utilize an encoder-decoder recurrent neural network and train it to solve the VNF migration scheduling problem. Through the simulation, we verify that the proposed method can agilely determine VNF redeployment locations by keeping the frequency of server overload and VNF migration minimum.**

Index terms*—Service function chaining (SFC), Integer linear programing (ILP), Machine learning (ML), Recurrent neural network (RNN), Function migration.

I. Introduction

Service function chaining (SFC) is one of application services providing a framework over network function virtualization (NFV) infrastructure [1-3]. A service function (SF) chain contains a series of virtual network functions (VNFs); for example, it may start from load balancers to contents server across a firewall and an intrusion detection system (IDS). Currently, SFC construction may take about two weeks from receiving a construction request to providing the service to the customers as in the case of network construction [4]. To deal with time-varying network environment and diverse quality-of-service (QoS) requirements in future networks, automation of SFC construction and adjustment is essential to shorten the network configuration and/or reconfiguration time and quickly roll out a new service, avoid human errors in manual configuration, and deal with shortage of skilled human resources. Especially, automation of the deployment and periodical adjustments of computational resources for each network function (NF) would help in realization of efficient and stable service provisioning to customers in time-varying network conditions (e.g., resource utilization and/or failure occurrences).

To avoid shortage of computational resources, prior work [5] has proposed a framework of dynamic resource adjustments of SF chains, which consists of the following three steps: 1) resource arbitration between various functions deployed in a server node, 2) function migration from one node to another by keeping the communication path unchanged, and 3) function migration from one node to another by changing the communication path. The first step achieves more agile operation of resource control, and thus, is prioritized over the subsequent ones. The first and second steps of resource adjustments have discussed and evaluated in [5], but the third step has not been investigated. Therefore, in this paper, we focus on the third step of resource adjustments. The major challenging issue of the third step is how to agilely determine a solution of VNF redeployment while minimizing the number of VNF migration.

VNF placement, allocation and scheduling problems have been studied in related works [6-10]. They apply different approaches, such as solving optimization problem [6,7,10], proposing heuristic methods [8] and game theory based algorithm [9]. However, these existing techniques have limitations in providing an agile operation of VNF migration by keeping the frequency of operation at a minimum level.

For autonomic network management, application of machine learning (ML) techniques has been proposed in related work [11-13]. Mijumbi *et al.* [11] propose methods for network traffic prediction by utilizing graph neural networks and dynamic resource allocation based on the prediction results. Narayanan *et al.* [12] propose to apply an encoder-decoder RNN (ED-RNN) to content cache management for forecasting contents popularity in information centric networks. Similarly, application of reinforcement learning has been proposed for traffic engineering in data center networks [13]. However, the above-mentioned techniques have not considered the situation of dynamic VNF migration for SFC. As the application of ML technologies has a potential for agile operation of resource control compared with existing other techniques such as integer linear programming (ILP), we explore its application for addressing the problem of dynamic VNF migration.

In this paper, to determine a solution of VNF redeployment with minimizing the number of VNF migration, we propose an ML-based autonomic service function migration method. The proposed method uses an attention-based ED-RNN, which is a popular neural network architecture that has been widely used for natural language translation (NLT) [14]. The attention-based ED-RNN is effective, especially when sequential input data is available. Therefore, it is also suitable for the dynamic VNF migration technique by utilizing the time-series data of resource demands of NFs. The proposed method determines a suitable server node to accommodate each NF at each time slot by using the ED-RNN with the attention architecture trained by optimal solutions. It can thus mitigate the occurrences of server overload and VNF migration. Through computer simulation, we verify that the proposed method can prevent the situations of resource shortage and reduce the occurrences of VNF migration.

978-1-5386-9376-6/19/$31.00 ©2019 IEEE

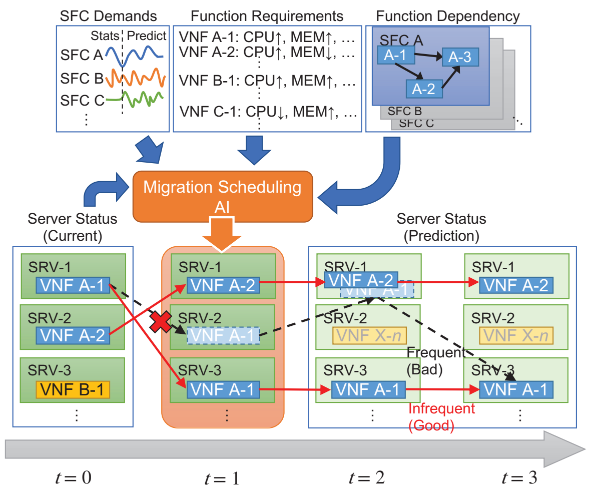

Fig. 1. Service function migration scheduling AI.

This paper is organized as follows. In Sec. II, we introduce our framework of service function migration. The VNF migration problem is formulated in Sec. III. Our ED-RNN architecture is described in Sec. IV, and its performance evaluation is presented in Sec. V. Finally, we conclude this work in Sec. VI.

II. Autonomic Service Funtion Migration Scheme

A. Autonomic Resource Adjustments

In prior work, we proposed a computational resource adjustment method including the process of autonomic resource arbitration among services and VNF migration along the same SF chain in order to achieve the stable operation of SF chains [5]. The proposed method can reduce the number of occurrences of CPU-saturation by more than 90 % in comparison with a static resource allocation method. This method does not consider changing the communication path of the SF chains on the substrate networks. However, if enough computational resources are not available in the nodes located along the given communication path, the above method cannot solve the issue of resource shortage. To address this issue, VNF should be migrated from the current node to another by changing the communication path (i.e., network topology). We apply this approach to the method proposed in this paper.

B. Migration Planning Based on Demand Prediction

The infrastructure provider needs to determine the destination server node from a large number of nodes in the network to migrate a VNF. Solving this problem by ILP in a short time is too hard due to its complexity, and that is the main reason to adopt the ML based approach. Fig. 1 illustrates an advantageous effect of our VNF migration scheduling artificial intelligence (AI). This AI creates VNF migration plans in accordance with the utilization of computational resources at the current state ($t = 0$), QoS requirements of VNFs, position dependency of VNFs within each SF chain, and predicted future resource demands ($t = 1,2,3$). In this figure, VNF A-1 and VNF A-2 are migrated because of resource shortages. If VNF A-1 is migrated to the server node 2 (SRV-2) at $t = 1$, it is forecasted that the VNF is frequently migrated among servers in every time slot ($t = 2,3$) as shown by dashed arrows. Therefore, it would be optimal if VNF A-1 is migrated to SRV-3 at $t = 1$ because the solution can minimize the number of VNF migration.

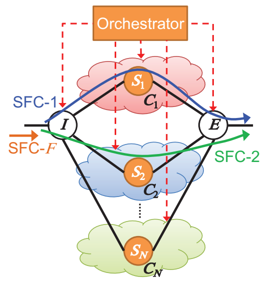

Fig. 2. Illustration of problem statement.

III. Problem Statement and Formuralization

Fig. 2 shows a simple multi-homed network model which we use as the first step to construct the VNF migration scheduling AI and verify its potential. Hereinafter, we explain our VNF migration scheduling AI for this network. At first, we solve the VNF migration scheduling problem by using the ILP even though it takes a long time. Then, we train the ED-RNN with the optimal solution obtained by the above ILP. As a result, the trained ED-RNN generates a VNF migration schedule automatically. In this section, we explain the problem statement for VNF migration problem and its formularization to solve for optimal solutions.

In Fig. 2, the ingress (I) and egress (E) nodes are connected by N paths. Each node can host many servers, depending on their resources. N servers whose capacity is C_i $(i = 1,2,\dots,N)$ are placed at each of paths. The servers provide resources to VNFs included in each SFC. The orchestrator monitors resource utilization of all servers and configures ingress/egress nodes for load distribution to each server. For example, if SFC-1 and SFC-2 require some resources of servers, the orchestrator has only to configure their routes in such a way that they not to go through the same path. However, when many SFCs coexist in the network and their requirements frequently change, it is hard to determine which SF chain should be mapped to which route (communication path) in order to avoid resource shortage.

To solve this problem, we formularize the above VNF migration problem as a combinatorial optimization problem. In this paper, to simplify the problem, we assume that the orchestrator can accurately predict transitions of resource demands in all SF chains at each time slot ($t = t_1, t_2, \dots, t_T$). Note that, we consider that our VNF migration scheduling AI is used for short time scale; namely, the time slot should not be so large (e.g., ranging from a few seconds to a few minutes). Resource demands in SF chains are classified into three classes: high (H), middle (M) and low (L), and the demand in each SF chain dynamically changes. This optimization problem is formularized to seek an optimal resource allocation solution for F VNFs (VNF-$f, f = 1,2,\dots,F$) at each time slot. We define the several variables as follows.

$x_{i,f}(t) \in 0,1$:	Binary variables representing server i hosts the VNF-f at the time t or not.
$d_f(t) \geq 0$:	Demand of the VNF-f at the time t (input parameters, constant real number).
$C_i > 0$:	Capacity of server i (input parameter, constant integer).
$S_i(t) \geq 0$:	Variables representing the amount of resource shortage of server i at the time t

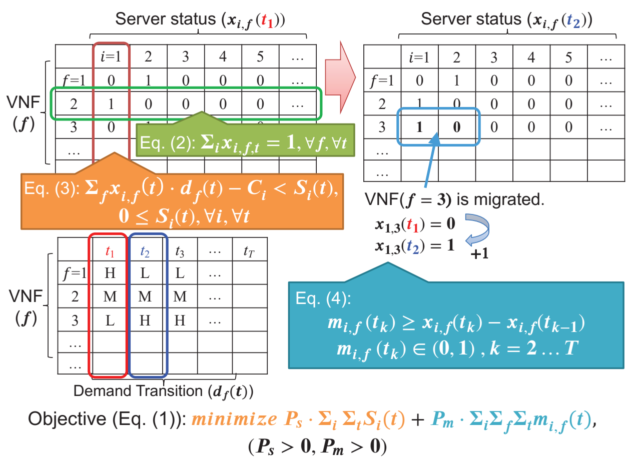
Server status ($x_{i,f}(t_1)$)
Server status ($x_{i,f}(t_2)$)
VNF (f)
Eq. (2): $\Sigma_i x_{i,f,t} = 1, \forall f, \forall t$
Eq. (3): $\Sigma_f x_{i,f}(t) \cdot d_f(t) - C_i < S_i(t)$, $0 \le S_i(t), \forall i, \forall t$
VNF($f = 3$) is migrated.
$x_{1,3}(t_1) = 0$
$x_{1,3}(t_2) = 1$
+1
Eq. (4): $m_{i,f}(t_k) \ge x_{i,f}(t_k) - x_{i,f}(t_{k-1})$, $m_{i,f}(t_k) \in (0,1), k = 2 \dots T$
Demand Transition ($d_f(t)$)
Objective (Eq. (1)): minimize $P_s \cdot \Sigma_i \Sigma_t S_i(t) + P_m \cdot \Sigma_i \Sigma_f \Sigma_t m_{i,f}(t)$, ($P_s > 0, P_m > 0$)

Fig. 3. Formulation of VNF migration scheduling problem.

(real number). $S_i(t) = 0$ means no shortage occurs.

$m_{i,f}(t_k) \in 0,1$: Binary variables representing the VNF-f is migrated at the time t_k.

Then, we formularize the VNF migration scheduling problem as follows.

$$\text{Objective:} \quad \underset{}{\overset{minimize}{P_s \cdot \Sigma_i \Sigma_t S_i(t) + P_m \cdot \Sigma_i \Sigma_f \Sigma_t m_{i,f}(t),}} \quad (P_s > 0, P_m > 0) \tag{1}$$

$$\text{Subject to:} \quad \Sigma_i x_{i,f}(t) = 1, \forall f, \forall t \tag{2}$$

$$\Sigma_f x_{i,f}(t) \times d_f(t) - C_i < S_i(t), \forall i, \forall t \tag{3}$$

$$m_{i,f}(t_k) \ge x_{i,f}(t_k) - x_{i,f}(t_{k-1}), \quad \forall i, \forall f, k \in 2,3, \dots, T \tag{4}$$

Fig. 3 illustrates the relationships between equations. In this figure, the upper two tables represent the values of $x_{i,f}(t)$ at time t_1 and t_2, and thus, denote the server that accommodates the VNF-f at time t_1 (and t_2). And the lower table represents the values of $d_f(t)$, and thus, denotes the resource demands in SF chains at time $t = t_1, t_2, \dots, t_T$. Eq. (2) constrains that each VNF must be allocated on only one server. In Eq. (3), the resource shortage value $S_i(t)$ becomes larger than 0 only when the sum of demands in SF chains located at server i exceeds its capacity C_i at time t. Eq. (4) denotes if a VNF is migrated from a server to another or not, at time t_k. The value of $m_{i,f}(t_k)$ equals 1 when VNF-f is migrated to server i. For example, when VNF-3 is migrated from server 2 to server 1, $x_{i,f}(t_k) - x_{i,f}(t_{k-1}) = 1$ as shown in Fig. 3. If VNF-f is not migrated at time t_k, $m_{i,f}(t_k)$ becomes 0. In the objective function of Eq. (1), P_s and P_m represent the penalty values of resource shortage and migration, respectively. By tuning the ratio of P_s and P_m, we can assign a preference to each of them.

In summary, solving this problem clarifies the best combination of $x_{i,f}(t)$ values, i.e., which server i $(i \in 1,2, \dots, N)$ should accommodate VNF-f $(f \in 1,2, \dots, F)$ at each time slot $(t \in t_1, t_2, \dots, t_T)$. Optimal solutions keep the frequency of resource shortage and VNF migration as low as possible.

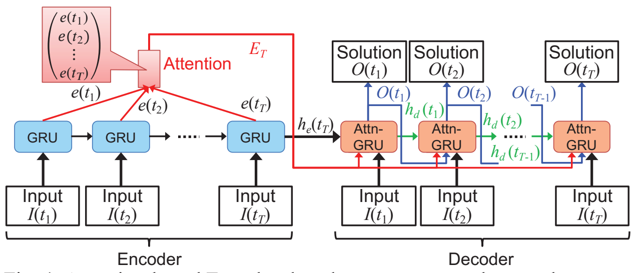

Fig. 4. Attention-based Encoder-decoder recurent neural network.

IV. System Architecture and Training Method

A. ED-RNN Architecture

We train the ED-RNN with the optimal solutions of the combinatorial problem as described in Sec. III. Our VNF migration scheduling problem uses the time series data of resource demands in SF chains. The RNN architecture is suitable for the reduction of the frequency of migration because the past state information is adopted effectively in the scheduling. Note that the past state information includes VNF placement and resource utilization at each time slot.

The ED-RNN architecture [15] is also called sequence-to-sequence RNN as it consists of two stages as shown in Fig. 4: the encoder and the decoder. The encoder reads a certain length of sequential input data entirely. Then the decoder processes the input data step by step with hidden data derived from the encoder. This architecture improves translation accuracy not only by using the past input data, but also by looking the future input data via the encoder. Our VNF migration scheduling problem also uses sequential input data, i.e., time series of SFC demands. So, we use the architecture by focusing on the similarity. We also use the attention architecture [14]. The original ED-RNN [15] only uses the output from the last cell of the encoder, while the attention architecture uses the output data from all cells of the encoder and can further improve the translation accuracy.

In the ED- RNN architecture, we use the Gated Recurrent Unit (GRU) to capture long-term dependency among the sequential input data. It is one of the most popular architecture. Long Short-Term Memory (LSTM) is another technique. Chung *et al.* [16] compared LSTM and GRU. Their results have shown that they have comparable performance, although GRU gives slightly better results. We use GRU because GRU has performance similar to that of LSTM, but requires a smaller number of input gates. In the encoder, the k-th GRU cell outputs two 1-dimensional tensors, i.e., H-length vectors, where $H(> 0)$ is a tunable parameter. The one is the output features of the cell $(e(t_k))$, and the other is the hidden states $(h_e(t_k))$. The GRU cell requires two input tensors, $I(t_k)$ and hidden state of the previous cell $h_e(t_{k-1})$. $I(t_k)$ includes servers' capacity C_i, VNFs' demands at time t_k, and ratio of P_s and P_m, that is, the length of $I(t_k)$ is $N + F + 1$. Output features from all cells $(e(t), t \in t_1, t_2, \dots, t_T)$ are concatenated to a 2-dimensional tensor, i.e., $T \times H$ matrix E_T, and forwarded to the decoder as the Attention matrix. In the decoder, the first Attn-GRU cell receives the last hidden states of encoder $(h_e(t_T))$ and the Attention (E_T), and outputs the VNF placement at time $t_1(O(t_1))$. The second and the following Attn-GRU cells decide the VNF placement at each time t_k sequentially. To determine the server that should host the VNF-f at time t_k, the k-th cell refers the output features

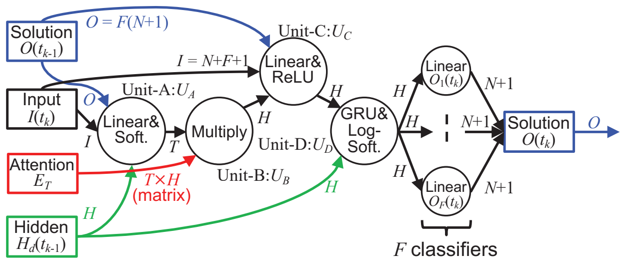

Fig. 5. Attn-GRU architecture.

($O(t_{k-1})$) and hidden states ($h_d(t_{k-1})$) from its previous cell and the Attention matrix (E_T).

Fig. 5 shows the Attn-GRU architecture, which contains four processing units (A, B, C and D) and F classifiers . The four units sequentially convert input tensors according to the following rules.

$$\begin{aligned} U_A^{(1)}(t_k) &\leftarrow Cat(O(t_{k-1}), I(t_k), h_d(t_{k-1})) \\ U_A(t_k) &\leftarrow Softmax(L_A(U_A^{(1)}(t_k))) \end{aligned} \tag{5}$$

$$U_B(t_k) \leftarrow U_A(t_k)E_T \tag{6}$$

$$\begin{aligned} U_C^{(1)}(t_k) &\leftarrow Cat(O(t_{k-1}), I(t_k), U_B(t_k)) \\ U_C(t_k) &\leftarrow ReLU(L_C(U_c^{(1)}(t_k))) \end{aligned} \tag{7}$$

$$\begin{aligned} U_D^{(1)}(t_k) &\leftarrow GRU(U_C(t_k), H_d(t_{k-1})) \\ U_D(t_k) &\leftarrow LogSoftmax(U_D^{(1)}(t_k)) \end{aligned} \tag{8}$$

Here, Cat denotes the simple concatenation of input 1-D tensors, and L_x $(x \in A, C)$ represents the linear transformation to incoming data: $Y = W_x(X) + b_x$, where W_x and b_x are the weight and bias tensors. The details of the other cells, $Softmax$, $ReLU$, GRU, and $LogSoftmax$, are omitted for simplicity.

Units A, B, and C determines which element in the Attention matrix E_T should be focused, by using $I(t_k)$, $O(t_{k-1})$, and $h_d(t_{k-1})$. Unit D (GRU) receives output from unit C ($U_C(t_k)$) and hidden states ($h_d(t_{k-1})$) and calculates the next state. Each of F classifiers determines the server that should accommodate each VNF-f according to the following calculations.

$$\begin{aligned} O_f(t_k) &\leftarrow L_f^O(U_D(t_k)) \\ O(t_k) &\leftarrow Cat(O_1(t_k), O_2(t_k), \dots, O_F(t_k)) \end{aligned} \tag{9}$$

L_f^O $(f \in 1,2,\dots,F)$ outputs $N+1$ length vector: $Y = W_f^O(X) + b_f^O$, where W_f^O and b_f^O are the weight and bias tensors. $O_f(t_k)$ are one-hot vectors that indicate the plausibility of each server corresponding to VNF-f. For example, when $O_f(t_k)$ equals (0, 0.3, 0.7, 0.4), it recognizes that server 2 should accommodate the VNF-f at time t_k because the element indexed by 2 (indices start from 0) shows the largest value. The server index corresponds to $argmax(L_f^O(U_D(t_k)))$. Note that the index starts from 0. The length of $O_f(t_k)$ is $N+1$ because it also includes the element to represent *don't care* (the first element, whose index is 0) to deal with negligibly-small VNFs' demands. Finally, output tensor $O(t_k)$ is the concatenation of $O_f(t_k)$ for $(f \in 1,2,\dots,F)$.

B. Training Encoder-Decoder RNN with Optimizations

To train the ED-RNN, we firstly generated 15,000 patterns of server capacities and time series data of resource demands in SF chains, and then, solved the VNF migration scheduling plans for each of them with the optimization problem described in Sec. III. When the values of N, F and T are large, it is hard to solve the problem because the number of combinations of $x_{i,f}(t)$ equals 2^{N+F+T}. Thus, we kept the parameters small. The ranges of N and F as [2, 5] and [5, 15], respectively. In each pattern, the values of N and F are selected randomly within the above ranges. QoS classes of VNFs at each time $d_f(t)$ are also determined randomly from 0.1 (low), 0.5 (middle) and 1 (high). Meanwhile, to simplify the evaluation, we fix $T = 10$, $P_s = 1$, and $P_m = 0.01$. The values of P_s and P_m were set empirically. That is, we place more importance on the strength of avoiding resource shortage than on avoiding frequent function migration. The length of input tensor is constant for the largest numbers of servers and SF chains, i.e., 5+15+1. If N (or F) is smaller than 5 (or 15), the gaps are filled by zero-value elements. We solved the above optimization by CBC [17].

In the training data, the output tensor $O^t(t_k)$ is consisted of the concatenation of F one-hot vectors $O_f^t(t_k)$, where $O^t(t_k)$ and $O_f^t(t_k)$ denote the training data. For example, if one of the training data indicates that server 3 should host VNF-5 at time t_4, $O_5^t(t_4)$ becomes (0, 0, 0, 1, 0). Note that here $N = 5$. Meanwhile, if VNF-6's demand is constantly 0 (i.e., $d_f(t)$ is always 0), we set $O_6^t(t_k)$ to (1, 0, 0, 0, 0) which means *don't care*. We define the loss function as the sum of cross-entropy loss values of all combinations of f at every time t_k, which is expressed as

$$\begin{gathered} -\Sigma_{t_k}\Sigma_f O_f^t(t_k) \log\left(O_f(t_k)\right), \\ t_k \in t_1, t_2, \dots, t_T, f \in F. \end{gathered} \tag{10}$$

To minimize the above loss function, W_x, b_x, W_f^o, b_f^o and parameters of GRU cells in the encoder and decoder are optimized by the back-propagation method based on Amsgrad. Our ED-RNN is implemented by using PyTorch [18]. We set the learning rate 0.001 and set the number of input/output size from hidden layers (H) 200. The value shows the best performance among $H = 100, 200, \dots, 500$.

V. Evaluation

Fig. 6 shows the training, validation and test results. 95% confidence intervals are also plotted in Fig. 6(b-c). The X-axis represents the number of epochs. All 15,000 patterns of training data are used in a random order at each epoch. We repeatedly trained the ED-RNN in 100 epochs. It took about two hours with Intel Core i9 7900X and Nvidia GeForce GTX 1060. To validate the learning results, we generated 1,000 patterns of server capacity and time series of demands in the same manner as the previous subsection. With the ED-RNN, computing the solution of 1,000 patterns took about ten minutes, i.e., each pattern was solved within a second in average. Fig. 6(a) shows the transition of the values of loss function (Eq. (10)) in the proposed method with and without the attention. The values of loss function were decreasing along with the progress of learning process in both cases due to the learning effect of ED-RNN.

Fig. 6(b) and (c) show transitions of the two factors of objective function: the frequency of resource shortage (the mean value of $\Sigma_i\Sigma_t S_i(t)/T$) and that of migration (the mean value of $\Sigma_i\Sigma_f\Sigma_t m_{i,f}(t)$). For comparison, we also evaluated the optimal solution and random migration method as well as the proposed method utilizing the ED-RNN without the attention. In the random migration method, VNFs are placed at servers randomly-chosen out of active ones at each time. The results in Fig. 6(b) indicate that the ED-RNN with the attention suppresses the frequency of shortage more effectively than the random migration, unlike the case without the attention. Furthermore, the results in Fig. 6(c) indicate that the ED-RNN with the attention keeps the frequency of migration far smaller than that of the case without the attention. The former always keeps over 10 times smaller than the latter. By using the previous output $O(t_{k-1})$ to decide the next output $O(t_k)$, the migration frequency is reduced. In addition, ED-RNN with the attention decides more effective migration schedule because it avoids the Gradient loss problem. In other words, the attention matrix (E_T) exhibits the relationship between inputs apart from each other (e.g. between $I(t_1)$ and $I(t_{10})$). As shown in Fig. 6(d), we can see that the value of the objective function $\left(\Sigma_i \Sigma_t S_i(t) + 0.01 \cdot \Sigma_i \Sigma_f \Sigma_t m_{i,f}(t)\right)$ in the proposed method with attention is always smaller than those in the random migration method and proposed method without the attention after the 30th epoch.

In summary, the ED-RNN with the attention architecture can reduce the mean value of resource shortage by invoking very low frequency of function migrations. Of course, it cannot guarantee that the solution is close to optimal. However, ED-RNN is expected to be complementary to the case when an optimization problem cannot be solved immediately, because ED-RNN solves the NF schedule within a short time. Since the training phase was not so long time (two hours), and it was easy to retrain even if patterns of demands of SFCs drastically changes from the past.

VI. Conclusion

The dynamic adjustment of computational resource assigned to virtual network functions plays a key role in the deployment of management of future SFC infrastructure. In this paper, we investigated the problem of VNF migration scheduling that may stipulate changing the communication path or route and proposed the method by utilizing the ED-RNN. We showed through simulation that, by training with the optimal solutions, the ED-RNN with the attention architecture has the high potential for preventing resource shortage and suppressing the occurrences of VNF migration. In future work, we plan to improve the performances of proposed ED-RNN architecture so that it can be adapted to more complicated topologies of large-scale real networks.

Acknowledgement

This work was conducted as part of the project entitled "Research and development for innovative AI network integrated infrastructure technologies" supported by the Ministry of Internal Affairs and Communications, Japan.

References

[1] IETF RFC 7665, "Service Function Chaining (SFC) Architecture," Oct. 2015.

[2] IETF RFC 8300, "Network Service Header (NSH)," Jan. 2018.

[3] A. M. Medhat, *et al.*, "Service Function Chaining in Next Generation Networks: State of the Art and Reserch Challenges," *IEEE ComMag*, vol. 55, pp. 216-223, Feb. 2017.

[4] K. Katsuura, *et al.,* "IaaS Automated Operations Management Solitions That Improve Virtual Environment Efficiency," *NEC Technical Journal*, vol. 8, pp. 29-32, Apr. 2014.

[5] T. Miyazawa, M. Jibiki, V. P. Kafle, and H. Harai, "Autonomic Resource Arbitration and Service-Continuable Network Function Migration along Service Function Chains," *IEEE/IFIP NOMS*, Apr. 2018.

[6] S. Mehraghdam, M. Keller, and H. Karl, "Specifying and Placing Chains of Virtual Network Functions," *IEEE CloudNet*, Oct. 2014.

[7] A. F. Ocampo, *et al.*, "Optimal Service Function Chain Composition in Network Functions Virtualization," *IFIP AIMS*, July 2017.

[8] M. Ghaznavi, N. Shahriar, S. Kamali, R. Ahmed, and R. Boutaba, "Distributed Service Function Chaining," *IEEE Journal on Selected Areas in Communications*, vol. 35, pp. 2479-2489, Nov. 2017.

[9] 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.

[10] T.-W. Kuo, B.-H. Liou, K. C.-J. Lin, and M.-J. Tsai, "Deploying Chains of Virtual Network Functions: on the Relation Between Link and Server Usage," *IEEE/ACM ToN*, vol. 26, pp. 1562-1576, Aug. 2018.

[11] R. Mijumbi, *et al.*, "Topology-Aware Prediction of Virtual Network Function Resource Requirements," *IEEE TNSM*, vol. 14, pp. 106-120, Mar. 2017.

[12] A. Narayanan, S. Verma, E. Ramadan, P. Babaie, and Z.-L. Zhang, "DeepCache: A deep Learning Based Framework for Content Caching," *ACM SIGCOMM WS (NetAI)*, pp. 48-53, Aug. 2018.

[13] L. Chen, J. Lingys, K. Chen, and F. Liu, "AuTO: Scaling Deep Reinforcement Learning for Datacenter-Scale Automatic Traffic Optimization," *ACM SIGCOMM*, pp. 191-205, Aug. 2018.

[14] M.-T. Luong, H. Pham, and C. D. Manning, "Effective Approaches to Attention-based Neural Machine Translation," *arXiv:1406.1078*, June 2014.

[15] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to Sequence Learning with Neural Networks," *NIPS*, pp. 3104-3112, Dec. 2014.

[16] J. Chung, C. Gulcehre, K. H. Cho, and Y. Bengio, "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling," *arXiv:1412.3555*, Dec. 2014.

[17] COIN-OR Branch-and-Cut MIP Solver. [Online]. https://projects.coin-or.org/Cbc

[18] PyTorch. [Online]. https://pytorch.org/

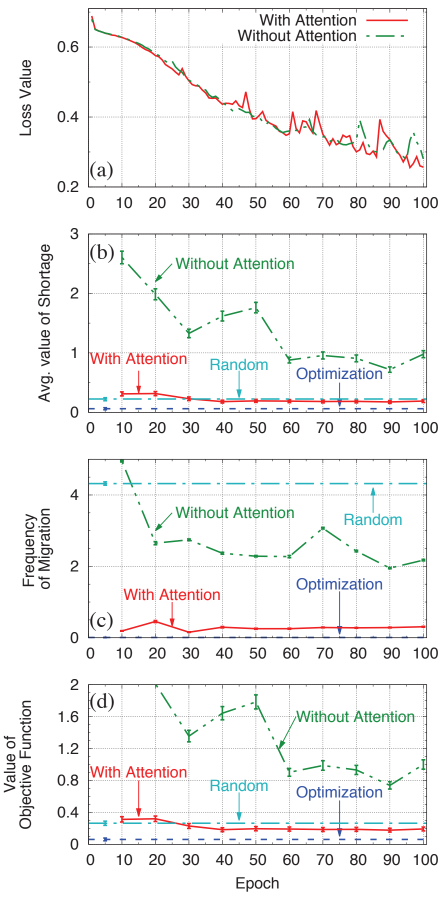

Fig. 6. Training and validation results. (a) Training: Time series of the values of loss function. (b) Validation: Frequency of the resource shortage. (c) Validation: Frequency of the SFC migration. (d) Validation: Value of the objective function.