Scaling Geo-Distributed Network Function Chains: A Prediction and Learning Framework

Ziyue Luo, *Student Member, IEEE*, Chuan Wu, *Senior Member, IEEE*, Zongpeng Li, *Senior Member, IEEE*, and Wei Zhou

***Abstract*— Geo-distributed virtual network function (VNF) chaining has been useful, such as in network slicing in 5G networks and for network traffic processing in the WAN. Agile scaling of the VNF chains according to real-time traffic rates is the key in network function virtualization. Designing efficient scaling algorithms is challenging, especially for geo-distributed chains, where bandwidth costs and latencies incurred by the WAN traffic are important but difficult to handle in making scaling decisions. Existing studies have largely resorted to optimization algorithms in scaling design. Aiming at better decisions empowered by in-depth learning from experiences, this paper proposes a deep learning-based framework for scaling of the geo-distributed VNF chains, exploring inherent pattern of traffic variation and good deployment strategies over time. We novelly combine a recurrent neural network as the traffic model for predicting upcoming flow rates and a deep reinforcement learning (DRL) agent for making chain placement decisions. We adopt the experience replay technique based on the actor–critic DRL algorithm to optimize the learning results. Trace-driven simulation shows that with limited offline training, our learning framework adapts quickly to traffic dynamics online and achieves lower system costs, compared to the existing representative algorithms.**

***Index Terms*— Network function virtualization, service function chain, reinforcement learning, deep learning.**

I. Introduction

THE network function virtualization (NFV) paradigm advocates deploying virtualized network functions (VNFs) on standard virtualization platforms, *e.g.*, in cloud data centers, for network traffic processing [1]. The VNFs are typically chained together as VNF service chains to provide network services, *e.g.*, “Web Application Firewall→IDS→Load Balancer” for access control to a Web service. Many chains are deployed over the WAN with VNFs located in different data centers, to process network flows between geo-dispersed sources and destinations. For example, the upcoming 5G mobile networks are envisioned to support various services such as machine-type communications (MTC) and Internet of Things (IoT) through network slicing, and a network slice primarily consists of chain(s) of 5G network functions (NFs) [2]. For WAN optimization, deduplication or compression functions are deployed close to sources of flows and traffic shaping can happen anywhere along the routes from sources to destinations [3]. For a control-plane service chain “P-CSCF→S-CSCF” in an IMS [4], instances of Proxy-Call Session Control Function (P-CSCF), which a user contacts for call registration, should be placed close to the callers, while Serving-Call Session Control Function (S-CSCF) for session control can be deployed in the middle between a caller and a callee.

The goal of NFV is to achieve significant cost reduction, as compared to traditional NF provisioning with dedicated hardware. On-demand deployment and agile scaling of VNF instances play a key role in cost conservation: VNF instances are easily added when flow demand increases, and removed when idling; geographical placement of VNFs can be flexibly adjusted according to varying geo-distribution of flow sources and destinations. For the example of network slicing in 5G networks, resources can be dynamically allocated to network slices containing different VNF chains, guaranteeing service latency and cost according to the demand [5].

A number of recent studies have designed offline or online optimization algorithms for placement or scaling of VNF chains in one data center [6], [7]. Deployment of service chains over the WAN is less thoroughly investigated, owing mainly to the many more factors to evaluate, such as various resource costs and different delays when flows are routed through different data centers, to obtain an efficient solution. The difficulty escalates when time-varying flow demand is considered, and migration of VNFs over the WAN is needed. A few studies [8]–[11] adopt online optimization techniques to design online scaling algorithms with or without worst-case performance guarantees. Differently, we seek better scaling decisions in expectation through in-depth learning from experiences, using deep reinforcement learning techniques.

Deep reinforcement learning (DRL) has exhibited its power in various applications such as game play (*e.g.*, Go [12],

Manuscript received December 15, 2018; revised June 27, 2019; accepted June 29, 2019. Date of publication July 8, 2019; date of current version August 6, 2019. This work was supported in part by the Huawei Innovation Research Program (HIRP) under Grant HO2016050002BE, in part by the Hong Kong Research Grants Council (RGC) under Contract HKU 17204715, Contract 17225516, and Contract C7036-15G (CRF), in part by the NSFC under Grant 61502504 and Grant 61628209, in part by the Nature Science Foundation of Hubei Province under Grant 2016CFA030, in part by the Technological Innovation Major Projects of Hubei Province under Grant 2017AAA125, and in part by the Science and Technology Program of Wuhan City under Grant 2018010401011288. *(Corresponding author: Chuan Wu.)*

Z. Luo and C. Wu are with the Department of Computer Science, The University of Hong Kong, Hong Kong (e-mail: zyluo@cs.hku.hk; cwu@cs.hku.hk).

Z. Li is with the School of Computer Science, Wuhan University, Wuhan 430072, China (e-mail: zongpeng@whu.edu.cn).

W. Zhou is with the Wireless Network Research Department, Huawei Technologies, Shanghai 201206, China (e-mail: will.zhou@huawei.com).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JSAC.2019.2927068

Atari [13]) and robot control [14]. Recently, DRL has also been used to handle a number of scheduling problems in networking systems, including resource allocation in cloud [15] and big data systems [16], path selection in Internet routing [17], bitrate adjustment in adaptive video streaming [18], and traffic scheduling in cellular networks [19]. DRL is promising for learning service chain scaling strategies too: the policy of a DRL agent evolves over time as it interacts with the environment, allowing quick adaptation to the changing environment, much needed in a WAN-based NFV system with traffic variation and numerous randomness; besides, DRL adopts a deep neural network (NN), capable of modeling the complex inner connections among the many factors in VNF chain scaling decision making, and to learn a good strategy over time.

Nonetheless, the VNF chain scaling problem, when treated with DRL, faces challenges different from the existing scheduling problems handled by reinforcement learning (RL) so far: (1) the flow demand variation is irrelevant to DRL output actions, which, if naively fed into the NN in the input state, does not render a Markov decision process (MDP), making many RL techniques non-applicable; (2) given the many candidate data centers for deploying each VNF and the multiple VNFs on a service chain, the action space is very large if producing deployment of VNFs in the entire chain altogether, leading to slow convergence for training the DRL model [20].

This paper addresses the above challenges by proposing a novel deep learning framework which combines a recurrent neural network (RNN) as the traffic model for predicting upcoming flow rates, and a DRL model for making VNF chain placement decisions across a geographic span. The RNN takes flow rates in a historical time-window as input, and predicts the flow rate in the next time slot. Based on the current deployment of the VNF chain and the predicted flow rate, the DRL NN produces VNF deployment decisions to handle network traffic in the upcoming time slot. Our main technical contributions are as follows:

▷ We train the RNN as a separate network using supervised learning with observations of actual flow rates, instead of including it as part of the DRL NN to be trained altogether using environmental rewards (overall cost incurred by the chain in our model, which indirectly includes actual flow rates), for more accurate prediction of upcoming flow demand.

▷ The DRL NN takes predicted flow rate among the input state. We define each state based on previous DRL action and output of the RNN (based on input of historical flow rates), which ensures the transition to next state is only determined by current input state and action performed. Consequently, the entire process renders an MDP which serves as the foundation for RL algorithm. The input state to the DRL NN is not related to history, allowing experience replay technique, which encourages the learning agent to fully search in a large action space for better training results, and avoids poor decisions due to lack of exploration.

▷ We design a DRL framework based on the asynchronous actor-critic learning method. Due to the large state space in the VNF chain scaling problem, we combine experience replay and ϵ-greedy techniques to allow our agent to explore more actions. Instead of producing entire chain deployment decisions altogether, we allow placement of the VNFs to be decided one by one in sequence to form the chain, to expedite model convergence.

We carry out trace-driven simulation, and compare our DRL scheduler with representative baselines. The results show that with limited offline training, our learning framework adapts quickly to traffic dynamics online and achieves lower system costs, *i.e.*, 10%–42% reduction as compared to the baselines.

The rest of the paper is organized as follow. We discuss related work in Sec. II, and introduce the system model in Sec. III. Sec. IV presents our DRL design. Sec. V gives evaluation results, and Sec. VI concludes the paper.

II. Background and Related Work

A. VNF Chain Placement and Scaling

Ghaznavi *et al.* [8] study the problem of geo-distributed VNF chain placement and propose a local search heuristic to minimize consumption of physical resources. Cohen *et al.* [9] decouple the geo-distributed VNF chain placement problem into a facility location problem, which places VNFs on physical nodes, and a generalized assignment problem, which assigns service requests to VNFs. An approximation algorithm is proposed with theoretically proven performance. Sang *et al.* [21] design two heuristic methods for single-type NFV placement with a constant approximation ratio proven, and one optimal greedy placement algorithm for tree network topologies, assuming a fixed path for each flow. Nadig *et al.* [22] formulate the problem of Service Function Chain (SFC) mapping in geo-distributed data centers as an ILP and propose an application-aware flow reduction algorithm to obtain a simplified ILP that is directly solvable. These work all assume static systems without flow changes.

Considering dynamic flows, Shi *et al.* [6] model dynamic resource allocation of VNFs as an MDP, use Bayesian learning to predict future resource reliability and design a heuristic algorithm. A simple setting is studied, where VNFs are placed within one data center with a simpler cost structure. Zhang *et al.* [10] implement network coding functions based on the NFV paradigm and design an online deployment and scaling algorithm for the functions in the Internet to maximize the throughput of multicast sessions. Pei *et al.* [23] study the problem of SFC Embedding with dynamic VNF placement. A two-stage method is proposed: the first stage of the method is to place, select and concatenate VNF instances for SFC requests; the second stage of the method is to release redundant VNF instances considering the dynamic network load. Jia *et al.* [11] propose an online geo-distributed VNF scaling algorithm based on the regularization method and dependent rounding technique, and prove its competitive ratio with a worst-case performance guarantee.

Different from the above online optimization techniques or heuristic algorithm, we novelly design a DL-based framework that captures the inherent flow patterns and makes online scaling decisions according to the current system state and predicted flow rates, to achieve better performance under various scenarios.

B. Deep Reinforcement Learning

Reinforcement learning is a category of machine learning approaches that maximize an accumulated reward through repeated interactions with the environment. In each step τ, the agent observes the current state of the environment s_τ and produces an action a_τ, based on the policy π. With action a_τ, the environment transits to the next state $s_{\tau+1}$ and the agent receives a reward signal r_τ. The states $\mathbf{s}$ and actions $\mathbf{a}$ form an MDP, where the probability of each possible value of $s_{\tau+1}$ and r_τ is determined solely by the immediately preceding state and action, s_τ and a_τ [24]. The goal of reinforcement learning is to maximize the expected cumulative discounted reward:

$$R = \mathbb{E}_\pi \left[\sum_{\tau=0}^{\infty} \gamma^\tau r_\tau \right] \tag{1}$$

where $\gamma \in (0, 1]$ is the discount factor.

In deep reinforcement learning, deep neural network (DNN) is used to represent the agent's policy. DNN is a kind of NN with multiple layers of neurons. The multiple layers make DNN capable of learning vast information from the input data. A number of recent studies have applied DRL for various resource scheduling problems. Mao *et al.* [15] design a multi-resource scheduler based on RL that dynamically allocates computational resources to incoming jobs in a single data center. Mirhoseini *et al.* [16] decouple the device placement problem into two sub-problems and use two NNs to learn their own decisions through DRL. Xu *et al.* [17] propose a DRL framework with experience-driven control for optimal routing path selection in a communication network. Chinchali *et al.* [19] design a DRL-based traffic scheduler in a cellular network that quickly adapts to traffic variation. A key challenge in the cellular network system is that networks exhibit non-stationary dynamics, which is non-Markovian for RL algorithm to solve. To address the challenge, the authors include past actions and historical traffic pattern in the state for future forecast. Such technique successfully casts the original non-Markovian problem as a Markov Decision Process for RL method. They design the predictor to forecast future states over multiple time slots leveraging all past states. Unlike their design, we propose a traffic model that only models network traffic and makes prediction for each time slot. Therefore, we ensure a more precise prediction of incoming traffic rate. In addition, the exponentially large discrete action space of VNF chain scaling problem makes the method in [19], Deep Deterministic Policy Gradient (DDPG), inapplicable. Ye and Zhang [25] also adopt DDPG for Small cell Base Station (SBS) activation problem. Again, the difference between our problem and the SBS activation problem lies in the size of the action space. While their strategy only needs to decide if the base station is active or not, our scheduler gives a complete VNF placement in each time slot.

In conclusion, VNF chain scaling problem is different from the existing problems handled by reinforcement learning methods: (1) The incoming flow rate is unknown and independent from DRL output actions. This imposes difficulty to render the problem as an MDP and makes RL method non-applicable; (2) Naively producing the placement for the whole VNF chain leads to an exponentially large action space, making RL methods impractical due to slow convergence. Nevertheless, we believe that DRL is promising for VNF chain scaling problem as well, and are the first to tackle the challenges for doing so, different from those faced by existing DRL-based schedulers.

III. System Model

A. Geo-Distributed NFV System

Consider an NFV provider who can rent resources from M data centers distributed in M distinct geographical regions to provision VNF service chains. The NFV provider runs a DL-based scheduler for placement and scaling of each VNF service chain (different chains are scheduled by different schedulers). Each VNF chain is traversed by many network flows (*e.g.*, TCP connections) between various sources and destinations. For ease of presentation, we group the individual network flows originating from the same geographic region and ending at the same region, and refer to the flow group as *one flow* hereinafter. Our system is to place sufficient instances of each VNF in the chain onto geo-distributed data centers, catering for the need of all flows. Instances of the same VNF can be deployed onto different data centers, and different flows can share the same VNF instance.

We focus on one scheduler handling one service chain when presenting our design. There are N distinct VNFs in the service chain, whose connectivity in the chain is indicated by $e_{n,n'}$, with $e_{n,n'} = 1$ if (n, n') is a hop in the VNF chain and $e_{n,n'} = 0$ otherwise, $\forall n \in [N], n' \in [N]$.[1] There are in total I flows traversing the service chain, corresponding to I different source-destination region pairs. Let $m_s^i \in [M]$ ($m_d^i \in [M]$) be the data center in the region that the source (destination) of flow i belongs, $\forall i \in [I]$. For notation simplicity, we use m_s^i and m_d^i to indicate flow i's source region and destination region as well.

The system works in a time-slotted fashion, over a potentially large span of T time slots. Flow rate of each flow i, $f_i(t), t \in [T]$ (*i.e.*, aggregate rate of individual flows when sent out from the source m_s^i), varies over time, decided by the number of individual network connections the flow includes and the data rates along the connections at each time. $f_i(t) = 0$ indicates that there is no flow from region m_s^i to region m_d^i in t. In addition, we practically consider that the flow rate may change after being processed by a VNF: some VNFs, *e.g.*, those performing tunneling gateway functions such as IPSec/SSL VPN and media gateways, convert flow packets from one format to another, which may increase the packet size for encapsulation or decrease the packet size for decapsulation; some VNFs, which perform security functions (e.g., firewalls and intrusion detection), may drop packets which violate security policies [26]. We use $\lambda_n(t)$ to denote the flow change ratio of VNF n in t, computed by the overall rate of the flows after passing the VNF divided by the overall flow rate arriving at the VNF. Let $\bar{\lambda}_n(t)$ denote the cumulative flow rate change ratio from first VNF to the VNF before VNF n in the chain

[1]We use $[X]$ to represent the set $\{1, 2, \ldots, X\}$ in the paper.

at t: $\bar{\lambda}_n(t) = \lambda_{n_1}\lambda_{n_2}\ldots\lambda_{n^-}$, where $n_1, n_2, \ldots, n^-$ are the VNFs before n in the chain. Hence, the rate of flow i arriving at VNF n in t is $f_i(t)\bar{\lambda}_n(t)$.

Let P_n denote the processing capability of one instance of VNF $n \in [N]$, in terms of the incoming flow rate that it can handle in a time slot. Without loss of generality, we assume that instances of the same VNF used to process one flow are to be placed in the same data center in a time slot; that is, there is one source-to-destination flow path across the data centers for each flow, and there can be multiple instances of the same VNF in a data center to serve the flow, depending on the flow rate. On the other hand, different flows may share the same instance of a VNF n, if they place VNF n in the same data center, to minimize the number of instances deployed in the same data center.

At the beginning of each time slot t, the NFV provider adjusts service chain deployment for the flows, to prepare for upcoming flow changes in the time slot, by making the following decisions: (i) Placement of VNFs for each flow, indicated by $x^i_{m,n}(t)$: $x^i_{m,n}(t) = 1$ if VNF n of flow i is deployed in data center m in t, and $x^i_{m,n}(t) = 0$, otherwise, $\forall m \in [M], n \in [N], i \in [I]$. (ii) The number of instances of each VNF to be deployed in each data center, $z_{m,n}(t)$, which is decided by the processing capacity P_n and total rate of flows traversing VNF n in data center m in t, $\forall m \in [M], n \in [N]$.

B. Cost Structure

The goal of the NFV provider is to minimize the overall cost of running the service chain to serve the flows. We consider four types of costs.

(1) *Operating costs.* Let $O_{m,n}$ be the operating cost for running each instance of VNF n in data center m, for renting a virtual machine or container with required resource configuration for running the instance. The overall operational cost for running all VNFs of the service chain at t is:

$$C_{operate}(t) = \sum_{m\in[M]}\sum_{n\in[N]} O_{m,n} z_{m,n}(t) \tag{2}$$

(2) *Flow transfer costs.* Let E^{in}_m and E^{out}_m denote the transfer-in and transfer-out cost per unit of flow rate at data center m. Binary variable $y^i_{m',n',m,n}$ indicates whether flow i goes from VNF n' deployed in data center m' to VNF n in m in t, or not. We can calculate the overall flow transfer cost in t as:

$$\begin{aligned} C_{trans}(t) = &\sum_{i\in[I]}\sum_{m'\in[M]}\sum_{n'\in[N]}\sum_{m\in[M]} \\ &\times \sum_{n\in[N]} E^{in}_m \bar{\lambda}_n f_i(t) e_{n',n} y^i_{m',n',m,n}(t) \\ &+\sum_{i\in[I]}\sum_{m\in[M]}\sum_{n\in[N]}\sum_{m'\in[M]} \\ &\times \sum_{n'\in[N]} E^{out}_m \bar{\lambda}_{n'} f_i(t) e_{n,n'} y^i_{m,n,m',n'}(t) \end{aligned} \tag{3}$$

(3) *Deployment costs.* We use $D_{m,n}$ to denote the cost for deploying VNF n anew in data center m, while there is no instance of VNF n deployed in m in the previous time slot. The cost is mainly incurred due to the effort of copying the VNF's image to the data center, and launching a VM/container with the image. The cost is typically considered on the order of the operating cost to run a server for a short period [27]. Let binary variable $k_{m,n}(t)$ indicate if any instance of VNF n is deployed in data center m in t or not. The total deployment cost in t can be formulated as:

$$C_{deploy}(t) = \sum_{m\in[M]}\sum_{n\in[N]} D_{m,n} \max\{k_{m,n}(t) - k_{m,n}(t1), 0\} \tag{4}$$

(4) *Delay costs.* Let $l_{m,m'}$ denote the network delay from data center m to data center m', $\forall m, m' \in [M]$. The end-to-end delay of a flow is an import performance metric of geo-distributed service chain deployment. To formulate the end-to-end delay of a flow, we augment the service chain with a virtual first VNF and a virtual last-hop VNF, indicating the source and destination of the flow, with fixed deployment in m^i_s and m^i_d, respectively. Define $[\bar{N}]$ to be the set of VNFs including the two virtual ones. The end-to-end delay of flow i can be computed as $\sum\limits_{m\in[M]}\sum\limits_{n\in[\bar{N}]}\sum\limits_{m'\in[M]}\sum\limits_{n'\in[\bar{N}]} l_{m,m'} e_{n,n'} y^i_{m,n,m',n'}(t)$. We multiple the delay by a cost per unit of delay, L, to convert the delay to a delay cost. The overall delay cost of all flows at t is:

$$\begin{aligned} C_{delay}(t) = L &\sum_{i\in[I]}\sum_{m\in[M]}\sum_{n\in[\bar{N}]}\sum_{m'\in[M]} \\ &\times \sum_{n'\in[\bar{N}]} l_{m,m'} e_{n,n'} y^i_{m,n,m',n'}(t) \end{aligned} \tag{5}$$

The overall cost of the NFV system over the entire time span T is hence:

$$C_{all} = \sum_{t\in[T]} (C_{operate}(t) + C_{deploy}(t) + C_{trans}(t) + C_{delay}(t)) \tag{6}$$

C. DL-Based Scheduler

We propose a deep learning-based scheduler, to decide deployment and scaling of the geo-distributed VNF chain to serve multiple flows traversing the chain. Different from existing DL-based schedulers [15], [16], our scheduler does not rely on pure offline training. It consists of an RNN-based traffic prediction model and a DRL model. The traffic model is trained using limited historical traces. Using upcoming flow rate predicted by the traffic model, the DRL model learns the scheduling policy in an online manner: in each time slot, the scheduler takes system state composed of flow prediction and VNF deployment information and produces VNF placement to serve each flow; it observes system cost as reward signal along with actual flow rates and uses them as feedback to update its DRL NN as well as further update the traffic model. This design alleviates the need for historical traces in implementing our scheduler: we only need some

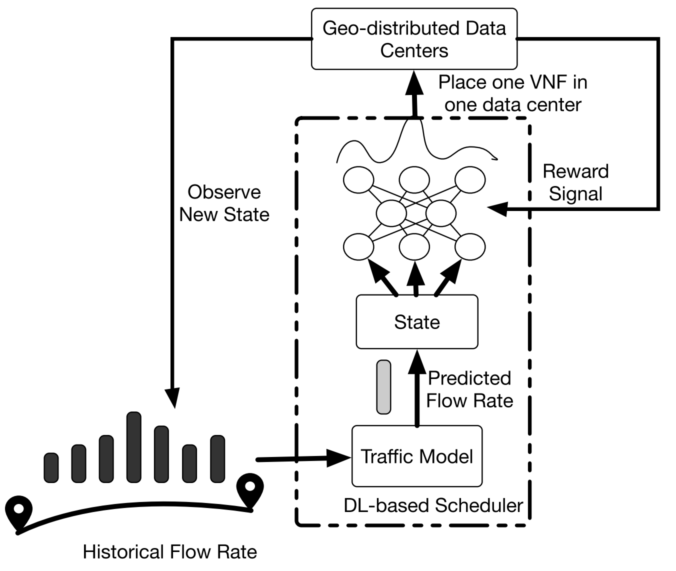

Fig. 1. Overview of DL-based scheduler.

historical traffic rates and user distribution, which are available statistics in many real-world systems, *e.g.*, Websites and other online services; we do not need detailed historical traces on VNF deployment and flow rate change ratios, which are largely unavailable in practice. Our online DRL is able to learn a good deployment strategy by interacting with the real world over time, towards minimizing the overall cost in Eqn. (6). Fig. 1 gives an overview of our scheduler.

IV. Deep Learning Framework for Service Chain Deployment

We present detailed design of our traffic model that learns the inherent pattern of flow rates and the DRL model for deployment decision making.

A. Traffic Model

In view of the strong traffic pattern in practical Internet applications (*e.g.*, daily/weekly access patterns from around the world in a Web application), we propose an NN model to learn the traffic pattern and use it for predicting upcoming flow rates during our online DRL. Combination of an environment model with RL has shown promising performance in tackling complex problems [28], [29]. The predicted flow rate is embedded into the input state of our DRL model. Therefore, we ensure that the next input state of the DRL model is solely dependent on our current input state and the action we select.

1) NN Model: We use a recurrent neural network (RNN) as the traffic model, which is a category of highly expressive models that are capable to learn rich temporal representation of data for future prediction [30]. RNN is a kind of NN where connections between neurons form a directed graph along a temporal sequence. The strong pattern in network traffic is suitable for RNN to learn and predict the future trend. The input to the RNN is a vector $(m_s^i, m_d^i, n, \boldsymbol{f})$. Here, m_s^i and m_d^i are flow source and destination, respectively; n indicates the VNF; $\boldsymbol{f}$ includes rates of the flow arriving at VNF n in

TABLE I
NOTATION

I	# of source-destination pairs for the VNF chain
M	# of data centers
E_m^{in}/E_m^{out}	transfer cost per unit of flow rate for flowing into or out of data center m
T	# of time slots
N	# of VNFs
$O_{m,n}$	operating cost per instance of VNF n in data center m
$D_{m,n}$	deployment cost for deploying VNF n anew in data center m
L	cost per unit of delay
P_n	processing capability per instance of VNF n
$\lambda_n(t)$	flow change ratio of VNF n at t
$\bar{\lambda}_n(t)$	accumulated flow change ratio from the first VNF in chain to the VNF before n at t
m_s^i/m_d^i	source / destination region of flow i
$f_i(t)$	flow rate of flow i
$e_{n,n'}$	(n, n') is a hop in VNF chain or not
$l_{m,m'}$	latency between data center m and m'
$x_{m,n}^i(t)$	deploy VNF n in data center m at t for flow i or not
$y_{i,m,n,m',n'}(t)$	whether flow i goes from VNF n in data center m to n' in m' at t or not
$z_{m,n}(t)$	# of VNF n instances deployed in data center m at t
$k_{m,n}(t)$	deploy VNF n in data center m at t or not

a historical time window, *i.e.*, $\boldsymbol{f} = \{f_i(t-\Delta t)\bar{\lambda}_n(t-\Delta t), f_i(t-\Delta t+1)\bar{\lambda}_n(t-\Delta t+1), \ldots, f_i(t-1)\bar{\lambda}_n(t-1)\}$, where t is the current time slot and Δt decides the window size (48 as in our evaluation). The output produced by the RNN is the predicted upcoming flow rate arriving at VNF n in t, $\widetilde{f_{i,n}}(t)$. In our experiment, the traffic model is a one-layer LSTM [31] with 128 neurons.

2) Model Training: The RNN model is trained offline and updated online, using supervised learning.

For offline training, we can use historical flow traces, describing flow distribution between different source-destination pairs and the flow rates, which are common statistics in real-world applications. We do not rely on any VNF information in the traces, and can just set n in the input to the RNN to be the first VNF in our service chain; essentially, our offline training is to capture flow rate pattern between each pair of source and destination. The training is done using ADAM optimization algorithm [32] to minimize the relative error between the predicted flow rates and real flow rates in the traces [30].

During online DRL training, the traffic model is further refined with real flow data. In each time slot, we observe the flow rate arriving at each NF in the chain, and can collect up to M^2N samples to train the traffic model RNN, for different flows between different source-destination pairs and arriving at different VNFs.

B. Online Deep Reinforcement Learning

We design an online DRL algorithm based on the actor-critic framework [24] for VNF chain placement and scaling. We allow our agent to produce part of the VNF chain placement in each learning step to reduce the action space, and all the decisions in one time slot form the complete VNF chain placement. The information about the previous decisions is embedded into the input state to help the agent learn the VNF chain structure.

1) DRL Model: We detail the DRL model as follows.

a) State: We design the input state to be mainly composed of previous deployment decisions and information about the upcoming flows from the traffic model. The input state to the DRL NN in time slot t should include the following:

(1) VNF deployment in previous time slot $t-1$, indicated by matrix $\mathbf{z}(M,N)$, where $\mathbf{z}(m,n)$ is the number of VNF n instances deployed in data center m in $t-1$.

(2) Source and destination of the current flow i under consideration, encoded by two M-dimensional vectors, $\vec{o}(M)$ and $\vec{d}(M)$, respectively. $\vec{o}(m)=1$ ($\vec{d}(m)=1$) if $m=m_s^i$ ($m=m_d^i$) and $\vec{o}(m)=0$ ($\vec{d}(m)=0$), otherwise.

(3) VNF deployment decisions made in the current time slot t, represented by matrix $\mathbf{r}(M,N)$. $\mathbf{r}(M,N)$ is initialized to all zero at the beginning of time slot t. During the current time slot, $\mathbf{r}(M,N)$ is continuously updated as each VNF placement is decided. With each action produced by the NN (see **Action** below), in the next input state to the NN, $\mathbf{r}(M,N)$ is populated as follows: if a previous action in t indicates placing VNF n serving flow i in m, we increase $\mathbf{r}(m,n)$ by the predicted number of instances of VNF n to serve flow i in t, computed based on the predicted flow rate arriving at VNF n using the traffic model, as follows:

$$\mathbf{r}(m,n) \leftarrow \mathbf{r}(m,n) + \frac{\widetilde{f_{i,n}(t)}}{P_n}. \tag{7}$$

(4) The VNF whose placement is to be decided, encoded in vector $\vec{h}(N)$. While $\mathbf{r}(m,n)$ represents previous decisions, the information of current VNF whose placement is to be decided remains unknown. Therefore, we use $\vec{h}(N)$ to encode the type and the estimated number of instances needed of the VNF that is to be placed. Supposing n is the VNF to be deployed, we have $\vec{h}(i)=0, \forall i \neq n$ and $\vec{h}(n)=\frac{\widetilde{f_{i,n}(t)}}{P_n}$.

(5) Deployment of the previous VNF in the chain to serve flow i in t, indicated by a matrix $\mathbf{g}(M,N)$. This information is included in the state, as current VNF placement is strongly related to the previous VNF's placement in the chain, due to the cost in bandwidth and delay. If the previous action is to place VNF n in data center m, then we have $\mathbf{g}(m,n)=\mathbf{r}(m,n)$ and other components in $\mathbf{g}$ remain 0.

The input state is hence:

$$\mathcal{S} = (\mathbf{z}(M,N), \vec{o}(M), \vec{d}(M), \mathbf{r}(M,N), \vec{h}(N), \mathbf{g}(M,N)).$$

Recall that our scheduler works in a proactive fashion: it adjusts VNF deployment at the beginning of a time slot, preparing for handling the upcoming flow rates in the time slot. As detailed in **Action** below, we allow multiple actions within one time slot. After each action, we will update the state, $\mathcal{S}$, according to the previous action. Therefore, we allow multiple state changes within one time slot. And, we do not have the exact flow rates to handle when making the deployment decisions, but only predicted rates based on the traffic RNN. Besides practicality, this design ensures the basic MDP assumption for our DRL, which requires that transition from current state s to the next state s' is only affected by the chosen action a at s [24]. Suppose the system works in a reactive fashion, and the input state s includes the actual flow rates to handle in a time slot; then the next state s' is not fully decided by s and action a taken on s, as the next-step input flow rates are independent of them. In our system, we use values computed based on predicted flow rates in the input state, which are decided by the traffic NN model according to previous flow rates; Hence with a traffic prediction model, our state transition satisfies the MDP assumption.

b) Action: The DRL NN produces a policy $\pi : \pi(s \mid a; \vec{\theta}) \rightarrow [0,1]$, which is a probability distribution over the action space. Here a represents an action, and $\vec{\theta}$ is the current set of parameters in the NN. Straightforward design of the action is to decide the placement of all VNFs to serve all flows in one action. However, this leads to an exponential action space, containing all possible combinations of VNF placement. A large action space incurs significant training cost and slow convergence [20]. To expedite learning of the NN, we simplify the action definition, and produce one data center as the deployment location for the input VNF (indicated in $\vec{h}(N)$). Hence the action space is of size M:

$$\mathcal{A} = \{a | a \in [M]\}$$

We allow multiple inferences over the NN for producing the complete set of VNF deployment decisions at the beginning of each time slot: we order the I flows randomly, and decide their VNF placement one flow after another; for each flow, we feed the VNFs one after another according to their sequence in the service chain, into the NN in $\vec{h}(N)$. For example, for flow i, we first indicate the first VNF n_1 in $\vec{h}(N)$; after producing one action (n_1's placement), we update state $\mathcal{S}$, indicating the second VNF in $\vec{h}(N)$, and produce the next action. This repeats until we have placed all VNFs for serving flow i, and then we move on to produce actions for placing VNFs to serve another flow.

Each action only gives on which data center to place the respective VNF. The number of instances of the VNF n to deploy in the output data center m is decided based on $\mathbf{r}(m,n)$ in Eqn. (7) using the actual flow rate, as multiple flows can share the same instances. For example, if the scheduler decides to place VNF 1 of flow 1, 2, 3 on the same date center, data center 1, at time slot t, then, the instance number of

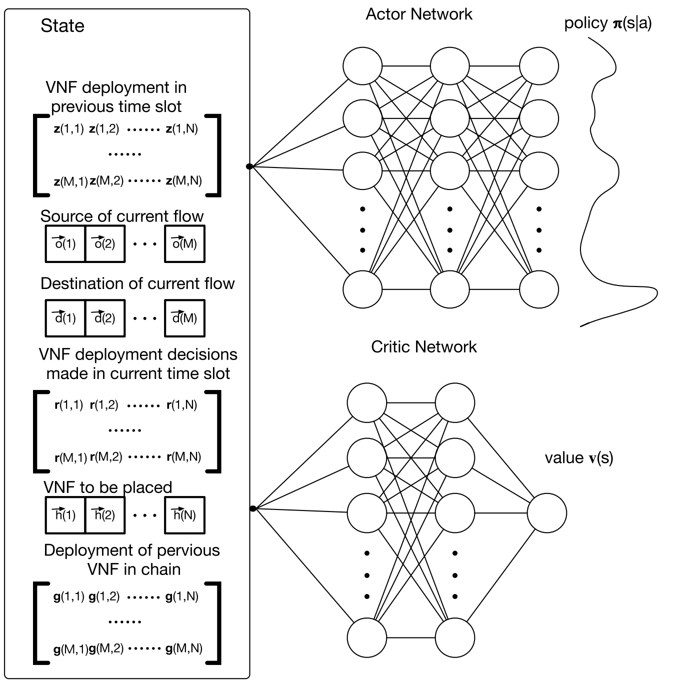
State
VNF deployment in previous time slot
z(1,1) z(1,2) ...... z(1,N)
z(M,1) z(M,2) ...... z(M,N)
Source of current flow
o(1) o(2) ... o(M)
Destination of current flow
d(1) d(2) ... d(M)
VNF deployment decisions made in current time slot
r(1,1) r(1,2) ...... r(1,N)
r(M,1) r(M,2) ...... r(M,N)
VNF to be placed
h(1) h(2) ... h(N)
Deployment of pervious VNF in chain
g(1,1) g(1,2) ...... g(1,N)
g(M,1) g(M,2) ...... g(M,N)
Actor Network
policy π(s|a)
Critic Network
value v(s)

Fig. 2. Actor-Critic framework for VNF scheduler.

VNF 1 to be deployed onto data center 1 is calculated as $\left\lceil \frac{\widetilde{f_{1,1}(t)}+\widetilde{f_{2,1}(t)}+\widetilde{f_{3,1}(t)}}{P_1} \right\rceil$.

c) Reward: At the end of each time slot, the agent receives reward signal r_τ. We define the reward as the additive inverse of the overall cost increment after the action. In this way, the cumulative rewards in Eqn. (1) equals the additive inverse of the overall cost, if the discount factor $\gamma = 1$. By maximizing the cumulative reward, we minimize the overall system cost, C_{all}.

2) Online DRL Training: We adopt a number of training techniques for online DRL.

a) Actor-critic based policy gradient method: To train the DRL agent, we adopt the actor-critic framework [24] as shown in Fig. 2. We maintain two DNNs in our DRL agent: one is the *actor network* that learns the deployment policy, and the other is the *critic network* which estimates the value of the current state. We use θ and θ_v to denote the parameters of the actor network and the critic network, respectively. Both networks take state as input. Output of action network is a probability distribution over total M data centers as policy. And, output of critic network is a real number representing the estimated cumulative discounted reward from input state following current policy.

The actor network is trained through the policy gradient method. The gradient of the expected cumulative discounted reward is calculated as follows [33]:

$$\nabla_\theta \mathbb{E}_{\pi_\theta}\left[\sum_{\tau=0}^{\infty} \gamma^\tau r_\tau\right] = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a|s) A_{\pi_\theta}(s,a)\right] \quad (8)$$

Here $A_{\pi_\theta}(s,a)$ represents the difference between the expected cumulative discounted reward starting from state s when agent chooses action a and follows policy π_θ afterward, and the expected discounted reward from state s following π_θ. $A_{\pi_\theta}(s,a)$ gives an evaluation of the current action. If $A_{\pi_\theta}(s,a)$ is greater than 0, than we know that choosing action a results in a better cumulative discounted reward than following policy π. Hence, the actor network will be updated towards a direction that tends to choose a more often when facing state s than before. Eventually, the actor network will learn a policy that produces an optimal action in each state.

However, unlike offline reinforcement learning, where the agent performs actions till the end state and calculates the cumulative discounted reward for learning, our online DRL agent is not able to acquire the expected cumulative discounted reward when facing state s. Thus, we use the critic network to estimate the cumulative discounted reward of each state following the current actor network's policy, which is also expressed as the value of each state, $v_{\theta_v}(s_t)$. Therefore, θ_v in the critic network is trained to obtain a better estimation of expected cumulative discounted reward for each state following the current policy through the Temporal Difference method [24], where α' is the learning rate of the critic network:

$$\theta_v \leftarrow \theta_v + \alpha' \sum_{\tau \geq 0} \nabla_{\theta_v}(\gamma v_{\theta_v}(s_{\tau+1}) + r_\tau - v_{\theta_v}(s_\tau))^2 \quad (9)$$

We leverage output value estimation from the critic network to calculate the updates of actor network parameters, θ:

$$\theta \leftarrow \theta + \alpha \sum_{\tau} \nabla_\theta \log \pi_\theta(a_\tau|s_\tau)(\gamma v_{\theta_v}(s_{\tau+1}) + r_\tau - v_{\theta_v}(s_\tau)) \quad (10)$$

Here we use $\gamma v_{\theta_v}(s_{\tau+1}) + r_\tau - v_{\theta_v}(s_\tau)$ to represent estimation of $A_{\pi_\theta}(s,a)$; we update θ towards the direction specified by $\log \pi_\theta(a_\tau|s_\tau)$ with a step size of $A_{\pi_\theta}(s,a)$. α denotes the learning rate for actor network.

b) Asynchronous learning with experience replay: The sequence of past experiences, represented by a series of $< s_\tau, a_\tau, r_\tau, s_{\tau+1} >$ transitions, encountered by an online DRL agent is non-stationary and update of agent policy is correlated because we update policy network with gradients computed through a sequence of continuous transitions. This makes online DRL agent unstable and easily fall into local optimum. Hence, we adopt an online asynchronous learning algorithm based on A3C [33], eliminating the update correlation in online DRL. We set up multiple learning agents to interact with its own environment, where each environment consists of the same data center setting and VNF chain structures. Every agent performs scheduling decisions on the same set of flows. After every τ_{update} learning steps, the agent collects past τ_{update} transitions in $trace$ and sends $trace$ to the central agent to update network parameters.

However, we find out that directly updating network parameters based on each agent's experience still results in poor performance. This is because the data correlation is not fully eliminated as experiences from the same agent are always used together to update the actor network and critic network. Thus, we adapt experience replay technique [13] that leverages a memory buffer to store past transitions $< s_\tau, a_\tau, r_\tau, s_{\tau+1} >$. We store the experiences from each agent in a memory buffer and sample transitions from it for network parameter updating.

By doing so, we ensure that every time experiences from multiple agents are used for updating network parameters and correlation is removed.

c) Exploration: In addition, to encourage exploration of our DRL agent in the action space (to produce actions leading to good rewards), we add an entropy regularization term $\beta\nabla_\theta H(\pi_\theta(\cdot|s_\tau))$ [33] to the update of parameters in the actor network. This term is to prevent our agent policy from becoming a deterministic policy.

We find that this commonly used entropy exploration is not sufficient for our RL to find a near-optimal policy (based on our experiments). To further allow good exploration of the action space, we adopt the ϵ-greedy method in our agent policy. During each inference, each learning agent chooses an action according to policy $\pi_\theta(a_\tau|s_\tau)$ with a probability of $1-\epsilon$, or randomly chooses a data center to deploy VNF with a probability of ϵ. This encourages our agent to explore more actions instead of simply following the current policy. Thus, the agent is more likely to find actions that lead to better cumulative reward than current policy.

The algorithms to be carried by each learning agent and the central agent are given in **Alg.** 1 and **Alg.** 2, respectively, where τ represents learning steps (each corresponding to one inference made on our DRL model) and the total number of learning steps, T_{end}, is $T \times I \times N$.

Algorithm 1 Agent Algorithm

```
1: $\tau \leftarrow 0$
2: while True do
3:   $\tau_{start} = \tau$
4:   Observe state $s_\tau$ with traffic model
5:   repeat
6:     Select $a_\tau$ according to $\pi_\theta(a_\tau|s_\tau)$ with the probability of $1-\epsilon$ or perform a random action with the probability of $\epsilon$
7:     Place the current VNF onto data center $a_\tau$
8:     Transfer to new state $s_{\tau+1}$ and corresponding reward is $r_\tau$
9:     $\tau \leftarrow \tau + 1$
10:  until $\tau - \tau_{start} == \tau_{update}$
11:  $trace = []$
12:  for $i \in \{\tau-1, \dots, \tau_{start}\}$ do
13:    $trace$.append($< s_i, a_i, r_i, s_{i+1} >$)
14:  end for
15:  Send $trace$ to central agent
16:  if $\tau == T_{end}$ then
17:    $\tau \leftarrow 0$
18:  end if
19: end while
```

V. Performance Evaluation

A. Simulation Setup

System Settings: We perform trace-driven simulations to evaluate our DRL-based framework. The whole time span is 7 days and each time slot lasts half an hour. Flow data are generated according to real-world Web traffic obtained from

Algorithm 2 Central Agent Algorithm

```
1: Initialize memory buffer $exp$
2: while receive $trace$ from agent do
3:   Add $trace$ to $exp$
4:   Sample $B < s, a, r, s' >$ from $exp$ randomly
5:   for $i = 1 \rightarrow B$ do
6:     Accumulate gradients wrt $\theta$: $d\theta \leftarrow d\theta + \alpha\sum_i \nabla_\theta \log \pi_\theta(a_i|s_i)(\gamma v_{\theta_v}(s'_i) + r - v_{\theta_v}(s_i)) + \beta\nabla_\theta H(\pi_\theta(\cdot|s_i))$
7:     Accumulate gradients wrt $\theta_v$: $d\theta_v \leftarrow d\theta_v + \alpha'\sum_i \nabla_{\theta_v}(\gamma v_{\theta_v}(s'_i) + r_i - v_{\theta_v}(s_i))^2$
8:   end for
9:   $\theta \leftarrow \theta + d\theta$, $\theta_v \leftarrow \theta_v + d\theta_v$
10:  $d\theta \leftarrow 0$, $d\theta_v \leftarrow 0$
11: end while
```

Huawei Inc. The peak flow rate is at most 720Mbps and appears between 9 and 11 PM. The bottom flow rate is at least 160Mbps and appears between 4 and 6 AM. Average flow rate is about 458Mbps. All the flow rates are normalized within $(0, 1]$. We use eight Google data center locations [34] to create our data center network. Latency between two data centers is set proportional to their distance within $(0, 1]$. There are six flows and four types of VNFs in the system. The first two types of VNFs result in an increment of flow rate and the last two reduce flow rate. The flow change ratio for each VNF is within $[0.9, 1.1]$. Data processing capability is set between 600Mbps and 900Mbps based on [35]. Operational cost, migrating cost and transfer cost are set based on Amazon EC2 pricing scheme [36]. Operational cost is between (0,1] and migrating cost is 10% of the operational cost. Transfer cost is set to be within (1, 2]. Weight L in computing the latency cost is set to be 5. We use different flow rate data for DRL training and evaluation.

DNN architecture: We use a one-layer long short-term memory network (LSTM) [31], a typical type of RNN, with 128 neurons as our traffic model. The actor network has three fully-connected hidden layers each with 128 neurons whose activation function is ReLU [31], and an output layer with 8 neurons using *softmax* function as the activation function. The critic network has three fully-connected hidden layers each with 128 neurons whose activation function is ReLU and one linear neuron as output.

Before DRL training, we train our traffic model by 1000 epochs using 3360 flows. The traffic model training can be done within 10 minutes and achieves an average error rate less than 1%.

DRL Settings: During DRL training, the discount factor γ is set to be 0.99. Learning rates for actor network, critic network and traffic model are all 0.0001. We use ADAM optimization for parameter update. Entropy weight β is set to be 5 and ϵ is 0.3. Memory buffer size is set to be 1×10^6 and size of each training batch is 512. Update interval τ_{update} is set to be 512 learning steps. The number of learning agents is set to 16.

During training, we assign each agent to one CPU and the total memory usage is 40G. After the learning converges, only

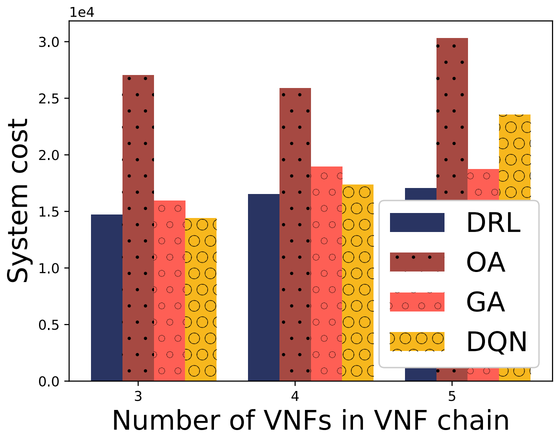

Fig. 3. System cost: different numbers of VNFs in VNF chain.

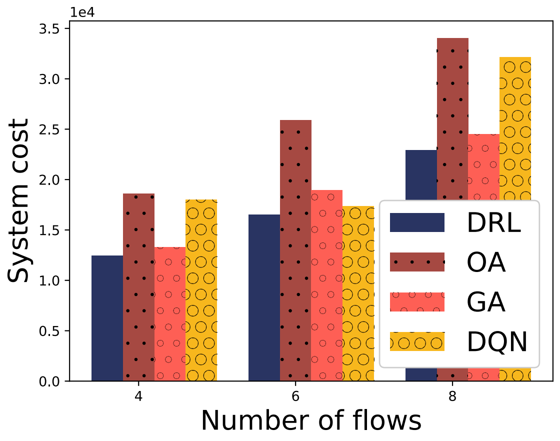

Fig. 4. System cost: different numbers of flows.

one agent is needed to produce placement decisions, whose running cost can be ignored.

Baselines: We compare our DL-based scheduler with the following algorithms:

1) Online Algorithm, OA: We design an online algorithm, *OA*, based on algorithm proposed in [11]. It formulates the original problem as an Integer Linear Program (ILP) and solves the ILP as LP in each time slot respectively by adding a regularizer to the objective. Rounding technique is then applied to obtain the feasible integer solution.

2) Greedy Algorithm, GA: A greedy algorithm, *GA*, is also proposed as a baseline. GA places each VNF serving every flow one by one similarly as our DL-based scheduler does, and assigns every VNF to a data center that minimizes the current increment in system cost.

3) Deep Q-Learning, DQN: A state-of-art deep reinforcement learning system proposed in [13]. The NN in DQN has three fully-connected hidden layers each with 128 neurons whose activation function is ReLU and one output layer with 8 linear neurons. We set the learning rate of DQN to 0.0001. The memory buffer size is 1×10^6 while the training batch size is 512. The initial ϵ is 0.99 and reduced by 1% every learning step.

In our evaluation experiments, our DL-based scheduler and DQN make decisions based on historical flow data and predictions, while OA and GA act based on actual flow rate at each time (i.e., accurate prediction).

B. Performance

Comparison: We compare the performance of our DL-based scheduler with baseline algorithms in terms of system cost in Figs. 3, 4, 5, 6, 7, 8 and 9. We observe that our DL-based scheduler outperforms three baselines in various scenarios.

1) Different Numbers of VNFs: Fig. 3 shows the performance of our scheduler under different numbers of VNFs in the VNF chain. For the case with 5 VNFs, we add an additional VNF with flow change ratio 1 and data process capability 750Mbps to the chain. We see that our DRL approach achieves an average reduction of 10% in system cost compared to GA and 41% compared to OA. DQN achieves comparable performance with 3 and 4 VNFs in the chain.

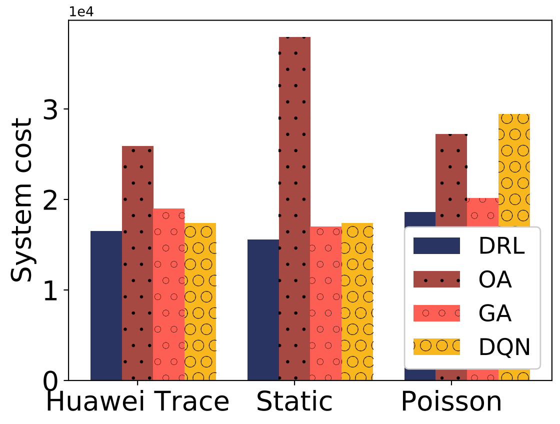

Fig. 5. System cost: different flow patterns.

However, as the chain grows larger, the DQN performs worse compared with our DL-based scheduler. The reason why OA performs significantly worse than our scheduler and GA is that even though it places VNF instances together to reduce operating cost, it incurs large cost in transferring data, which is considerably larger than other costs. On the other hand, our scheduler learns to leverage the chain structures, groups different VNFs together and places them in the same data center to avoid high transfer cost. Hence, we observe a small increase in system cost as the number of VNFs increases since there is no significant change in the transfer cost.

2) Different Numbers of Flows: Fig. 4 illustrates the system cost under different numbers of flows. We see that there is an obvious upward trend in system cost as the number of VNF flows increases due to the increase in all four costs. Our scheduler leads to 9% less system cost than GA and 33% less than OA. The performance of DQN is again unstable, and as the number of flows grows to 8, DQN performs poorly.

3) Flow Patterns: We further investigate how the network flow patterns affect performance in Fig. 5. We evaluate three flow patterns: 1) flow pattern extracted from Huawei Web traffic; 2) a Poisson arrival pattern where the average packet number per second is 60 and the average size per packet is 1000 Bytes; and 3) static flows where the flow rate remains at 458Mbps. We observe that our DRL scheduler significantly

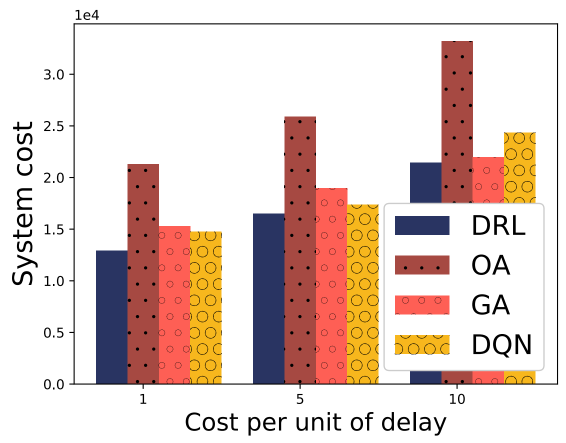

Fig. 6. System cost: different unit delay costs.

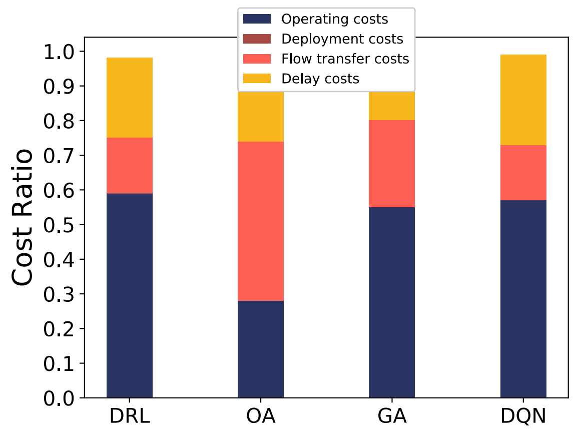

Fig. 7. Cost Composition.

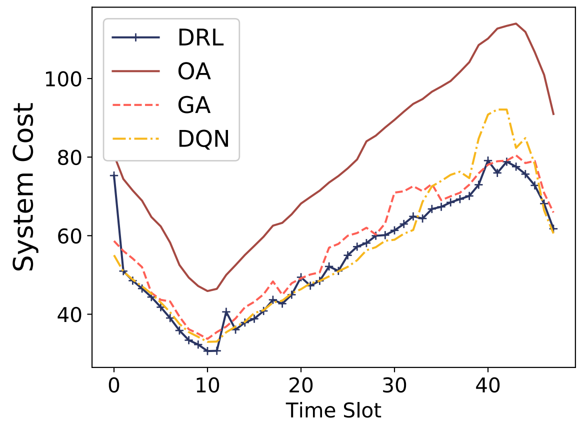

Fig. 8. System cost: evolution over a single day.

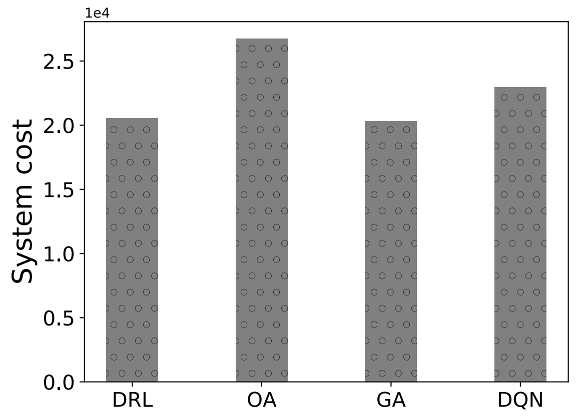

Fig. 9. System cost: network traffic with spikes.

outperforms OA by 42%, GA by 10% and DQN by 17% on average. We observe an even larger gap between our DL-based scheduler and OA in the case of static flows. OA is an approximation algorithm based on convex optimization and random rounding. The randomness introduced by rounding makes the performance of OA unstable. In contrast, DRL is a search-based method that smartly searches the solution space for a better solution. Therefore, our DL-based scheduler outperforms OA.

4) End-to-End Delay: End-to-end delay plays a critical part in the placement of VNF chain. Consequently, we study the performance of our scheduler under different unit delay costs L in Fig. 6. We observe the good performance of our DL-based scheduler under both delay-tolerant and delay-sensitive settings. This shows the strong adaptability of our DRL framework. We can deploy our scheduler to handle different flows for services with diverse requirements on network latency, and the scheduler is capable of learning the system requirements. The DRL approach reduces system cost by 10% compared to GA, 37% compared to OA, and 10% compared to DQN.

5) Cost Composition: Fig. 7 shows the ratio of every cost to the total cost achieved by each method. We observe that all four methods achieve negligible deployment cost. In accordance with previous descriptions, OA incurs a large flow transfer cost resulting in poor performance. On the contrary, our DL-based scheduler incurs a relatively high operating cost in return for less flow transfer and delay costs. Both GA and DQN present similar patterns that operating cost is the largest cost.

6) Distribution of the System Cost Over Time: We present the evolution of system costs over one day's course (48 time slots) in Fig. 8. We observe that the cost evolution is consistent with the flow pattern: large flow rates lead to large system costs during 9 to 11 PM.

7) Network Traffic With Spikes: We evaluate the resilience of our DRL method to network traffic with spikes in Fig. 9. We set the peak-to-mean ratio (PMR) to 1 with a peak flow rate of 720Mbps. The burst of network traffic appears between 9 and 11 PM, while flow rates in other time slots are around 360Mbps. We observe our DL-based scheduler outperforms OA and DQN significantly. Even though the spikes in network traffic make predictions by our traffic model inaccurate, the performance of our DL-based scheduler is comparable with GA which leverages the actual flow rates for scheduling.

a) Traffic Model: To illustrate the impact of the traffic model on our DRL framework, we compare our DRL scheduler with three alternatives, one with a poor traffic model, one without traffic model and one implemented by traditional forecaster Autoregressive Integrated Moving Average (ARIMA) [37]. For the alternative with a poor traffic model,

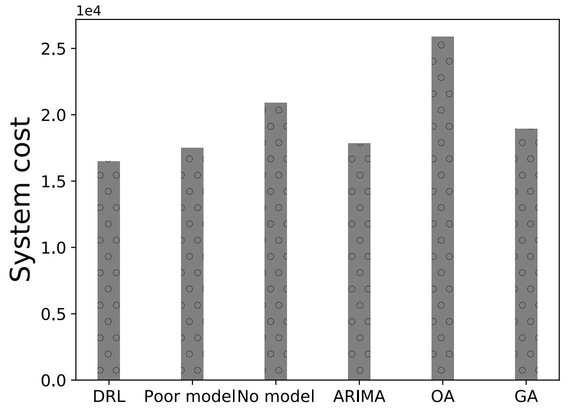

Fig. 10. System cost: good model vs. poor model vs. no model vs. ARIMA.

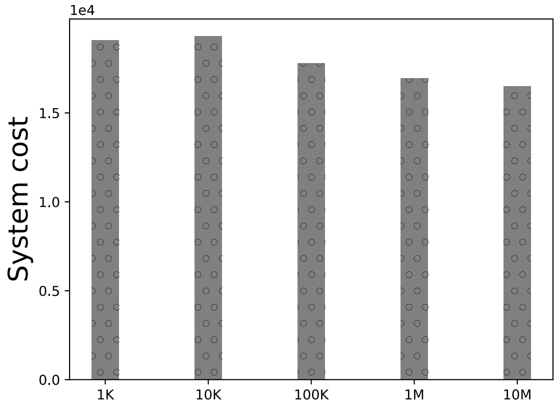

Fig. 11. System cost: different sizes of memory buffer.

we replace the traffic model in our scheduler with a function that always returns the flow rate in the last time slot. For the alternative without traffic model, we replace it with a dummy function that always returns 1. For ARIMA, we set the number of lag observations included in the model to be 12, the number of times that the raw observations are differenced to be 1 and the size of the moving average window to be 0. In Fig. 10, we see that the scheduler without traffic model cannot beat the baselines, because it has no knowledge of the ever-changing flow rate and can only give similar decisions in each time slot. Thus, we show that adding the traffic model can significantly help our agent make better decisions. Furthermore, the relatively small difference between the results of our traffic model and the poor model shows that our DRL agent is quite robust to the prediction results of the traffic model. The poor performance of DRL agent without traffic model proves that the agent indeed leverages the predicted flow rates for decision making. We observe that the agent without traffic model naively places all the VNFs onto one data center. During the evaluation, we find that RNN models traffic fluctuation better than ARIMA due to the frequent traffic fluctuations. ARIMA often makes false predictions when network traffic changes from rising to falling or the opposite. Consequently, scheduler with ARIMA as traffic model shows relatively poor performance compared with scheduler equipped with RNN-based traffic model.

b) Memory Buffer: Size of the memory buffer in DRL is critical to the performance of our scheduler. A larger memory buffer can store more past experiences for DRL agent to learn. Accordingly, a larger memory encourages the DRL agent to explore more instead of sticking to a local optimum. In Fig. 11, We observe that when the memory buffer size is small, our learning agent is not able to store much past experience, and learns only from current decisions; update correlation in online DRL exists, leading to poor performance. When we increase the memory buffer size, the system cost decreases as the update correlation is eliminated.

c) ϵ-Greedy: Fig. 12 illustrates the system cost when our DRL scheduler uses different values of ϵ in exploration. When ϵ equals 0, our scheduler performs solely according to the actions produced by the actor network and quickly converges to poor policy. By increasing ϵ, the system cost drops sharply

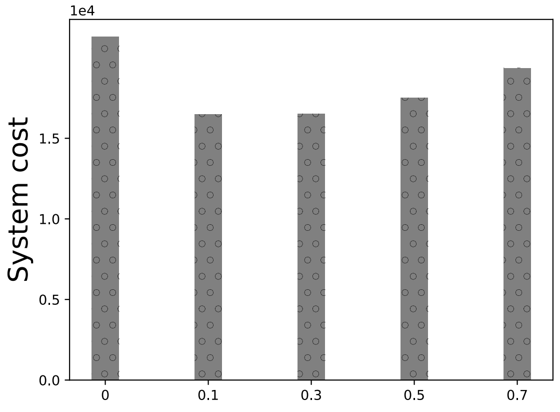

Fig. 12. System cost: different ϵ.

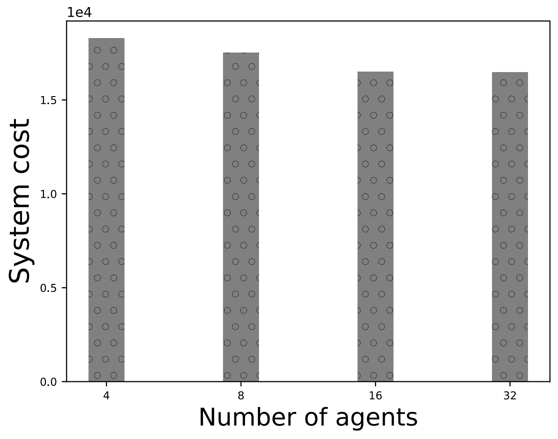

Fig. 13. System cost: different numbers of agents.

and then gradually increases. This is because when ϵ is large, our scheduler's policy is too random for the DNN to learn and hence leads to poor performance.

d) Learning Agents: Fig. 13 shows the system cost with different numbers of asynchronous learning agents. We observe that the scheduler obtains a low system cost when the number of learning agents is large. This is because with more learning agents, our scheduler is able to explore much more states in less training epochs. However, when there are more than 16 learning agents, the performance remains relatively stable. Accordingly, we set the number of learning agents in our scheduler to 16 by default.

VI. Conclusion

This paper presents a deep learning-based scheduler for geo-distributed VNF chain placement and scaling. Our scheduler consists of a traffic model and a deep reinforcement learning agent. The traffic model is trained by offline supervised learning with historical traces to capture the pattern in network traffic fluctuation, and further refined online with real flow data. The DRL agent based on an actor-critic framework is used to make VNF chain placement decisions across geo-distributed data centers and improves its policy through reward signals obtained purely in an online fashion. Compared with representative heuristic, online optimization and DRL algorithms, trace-driven simulation in various scenarios demonstrates that our DL-based scheduler outperforms baselines by 10%-42% regarding overall system cost.

References

[1] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee, "Network function virtualization: Challenges and opportunities for innovations," *IEEE Commun. Mag.*, vol. 53, no. 2, pp. 90–97, Feb. 2015.

[2] *5G White Paper*, NGMN Alliance, Frankfurt, Germany, Feb. 2015.

[3] Y. Zhang, N. Ansari, M. Wu, and H. Yu, "On wide area network optimization," *IEEE Commun. Surveys Tuts.*, vol. 14, no. 4, pp. 1090–1113, 4th Quart., 2012.

[4] G. Camarillo and M.-A. García-Martín, *The 3G IP Multimedia Subsystem (IMS): Merging the Internet and the Cellular Worlds*. Hoboken, NJ, USA: Wiley, 2007.

[5] J. Ordonez-Lucena, P. Ameigeiras, D. Lopez, J. J. Ramos-Munoz, J. Lorca, and J. Folgueira, "Network slicing for 5G with SDN/NFV: Concepts, architectures, and challenges," *IEEE Commun. Mag.*, vol. 55, no. 5, pp. 80–87, May 2017.

[6] R. Shi *et al.*, "MDP and machine learning-based cost-optimization of dynamic resource allocation for network function virtualization," in *Proc. IEEE SCC*, Jun./Jul. 2015, pp. 65–73.

[7] S. Gu, Z. Li, C. Wu, and C. Huang, "An efficient auction mechanism for service chains in the NFV market," in *Proc. IEEE INFOCOM*, Apr. 2016, pp. 1–9.

[8] M. Ghaznavi, N. Shahriar, S. Kamali, R. Ahmed, and R. Boutaba, "Distributed service function chaining," *IEEE J. Sel. Areas Commun.*, vol. 35, no. 11, pp. 2479–2489, Nov. 2017.

[9] R. Cohen, L. Lewin-Eytan, J. S. Naor, and D. Raz, "Near optimal placement of virtual network functions," in *Proc. IEEE INFOCOM*, Apr./May 2015, pp. 1346–1354.

[10] L. Zhang, S. Lai, C. Wu, Z. Li, and C. Guo, "Virtualized network coding functions on the Internet," in *Proc. IEEE ICDCS*, Jun. 2017, pp. 129–139.

[11] Y. Jia, C. Wu, Z. Li, F. Le, and A. Liu, "Online scaling of NFV service chains across geo-distributed datacenters," *IEEE/ACM Trans. Netw.*, vol. 26, no. 2, pp. 699–710, Apr. 2018.

[12] D. Silver *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, p. 354, 2017.

[13] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[14] A. Cully, J. Clune, D. Tarapore, and J.-B. Mouret, "Robots that can adapt like animals," *Nature*, vol. 521, no. 7553, p. 503, 2015.

[15] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning," in *Proc. ACM Workshop Hot Topics Netw.*, 2016, pp. 50–56.

[16] A. Mirhoseini, A. Goldie, H. Pham, B. Steiner, Q. V. Le, and J. Dean, "A hierarchical model for device placement," in *Proc. ICLR*, 2018, pp. 1–11.

[17] Z. Xu *et al.*, "Experience-driven networking: A deep reinforcement learning based approach," in *Proc. IEEE INFOCOM*, Apr. 2018, pp. 1871–1879.

[18] H. Mao, R. Netravali, and M. Alizadeh, "Neural adaptive video streaming with pensieve," in *Proc. ACM SIGCOMM*, 2017, pp. 197–210.

[19] S. Chinchali *et al.*, "Cellular network traffic scheduling with deep reinforcement learning," in *Proc. AAAI*, 2018, pp. 1–9.

[20] G. Dulac-Arnold *et al.*, "Deep reinforcement learning in large discrete action spaces," Dec. 2015, *arXiv:1512.07679*. [Online]. Available: https://arxiv.org/abs/1512.07679

[21] Y. Sang, B. Ji, G. R. Gupta, X. Du, and L. Ye, "Provably efficient algorithms for joint placement and allocation of virtual network functions," in *Proc. IEEE INFOCOM*, May 2017, pp. 1–9.

[22] D. Nadig, B. Ramamurthy, B. Bockelman, and D. Swanson, "Optimized service chain mapping and reduced flow processing with application-awareness," in *Proc. 4th IEEE Conf. Netw. Softwarization Workshops (NetSoft)*, Jun. 2018, pp. 303–307.

[23] J. Pei, P. Hong, K. Xue, and D. Li, "Efficiently embedding service function chains with dynamic virtual network function placement in geo-distributed cloud system," *IEEE Trans. Parallel Distrib. Syst.*, to be published.

[24] R. S. Sutton *et al.*, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 1998.

[25] J. Ye and Y.-J. A. Zhang, "DRAG: Deep reinforcement learning based base station activation in heterogeneous networks," Sep. 2018, *arXiv:1809.02159*. [Online]. Available: https://arxiv.org/abs/1809.02159

[26] J. F. Kurose and K. W. Ross, *Computer Networking: A Top—Down Approach*. London, U.K.: Pearson, 2013.

[27] M. Lin, A. Wierman, L. L. Andrew, and E. Thereska, "Dynamic right-sizing for power-proportional data centers," *IEEE/ACM Trans. Netw.*, vol. 21, no. 5, pp. 1378–1391, Oct. 2013.

[28] D. Ha and J. Schmidhuber, "Recurrent world models facilitate policy evolution," in *Proc. NIPS*, 2018, pp. 1–13.

[29] S. Racanière *et al.*, "Imagination-augmented agents for deep reinforcement learning," in *Proc. NIPS*, 2017, pp. 1–12.

[30] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, p. 436, 2015.

[31] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep Learning*, vol. 1. Cambridge, MA, USA: MIT Press, 2016.

[32] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. ICLR*, 2015, pp. 1–15.

[33] V. Mnih *et al.*, "Asynchronous methods for deep reinforcement learning," in *Proc. ICML*, 2016, pp. 1–10.

[34] *Google DC*. Accessed: Jul. 1, 2018. [Online]. Available: https://www.google.com/about/datacenters/inside/locations

[35] M. F. Bari, S. R. Chowdhury, R. Ahmed, and R. Boutaba, "On orchestrating virtual network functions," in *Proc. CNSM*, 2015, pp. 50–56.

[36] *Amazon EC2 Pricing*. Accessed: Jul. 1, 2018. [Online]. Available: https://aws.amazon.com/ec2/pricing/on-demand

[37] R. J. Hyndman and G. Athanasopoulos, *Forecasting: Principles and Practice*. Melbourne, VIC, Australia: OTexts, 2018.

Ziyue Luo received the B.S. degree from Wuhan University, China, in 2018. He is currently pursuing the Ph.D. degree with the Department of Computer Science, The University of Hong Kong. His research interests include cloud computing, network function virtualization, and distributed machine learning/big data analytics systems.

Chuan Wu received the B.Eng. and M.Eng. degrees from the Department of Computer Science and Technology, Tsinghua University, China, in 2000 and 2002, respectively, and the Ph.D. degree from the Department of Electrical and Computer Engineering, University of Toronto, Canada, in 2008. Since 2008, she has been with the Department of Computer Science, The University of Hong Kong, where she is currently an Associate Professor. Her current research is in the areas of cloud computing, distributed machine learning/big data analytics systems, network function virtualization, and intelligent elderly care technologies. She was a co-recipient of the Best Paper Awards at HotPOST 2012 and ACM e-Energy 2016.

Zongpeng Li received the B.E. degree in computer science from Tsinghua University in 1999 and the Ph.D. degree from the University of Toronto in 2005. He has been with the University of Calgary and Wuhan University. His research interests are in computer networks and cloud computing. He has co-authored papers that received the Best Paper Award at the following conferences: PAM 2008, HotPOST 2012, and ACM e-Energy 2016. He was named an Edward S. Rogers Senior Scholar in 2004. He received the Alberta Ingenuity New Faculty Award in 2007. He was nominated as the Alfred P. Sloan Research Fellow in 2007. He received the Department Excellence Award from the Department of Computer Science, University of Calgary, the Outstanding Young Computer Science Researcher Prize from the Canadian Association of Computer Science, and the Research Excellence Award from the Faculty of Science, University of Calgary.

Wei Zhou received the Ph.D. degree from the Department of Electrical Engineering and Information Systems, University of Science and Technology of China, in 2009. He is currently a Principal Engineer with Huawei Technologies Co., Ltd. His research interests include wireless communication, wireless intelligence, machine learning, and big data in-next generation mobile networks.