Energy-Efficient and Interference-Aware VNF Placement with Deep Reinforcement Learning

Yanyan Mu, Lei Wang, Jin Zhao
School of Computer Science, Fudan University
{yymu19,leiwang19,jzhao}@fudan.edu.cn

***Abstract*—By decoupling network functions from the underlying dedicated hardware, network function virtualization (NFV) has become a promising paradigm to reduce network operating expenses. NFV can provide elastic placement of Virtual Network Functions (VNFs) in the underlying data centers. However, the co-located VNFs on the same server may suffer from performance interference due to computing-resource and memory-resource sharing. This article focuses on how to ensure the performance of each VNF while minimizing the total energy consumption of the data center. By showing that the bin-packing problem is polynomial-time reducible to our model, we prove that the offline version of this problem is NP-complete. Then, for a homogeneous environment where all servers are of the same type, we design First-Fit Heuristic (FFH) algorithm and analyze the approximation performance of it by proving the lower bound value. For the heterogeneous environments, we propose an efficient solution based on deep reinforcement learning (DRL) named DDAP (Deep Deterministic Automatic Placement). Our experiments show that DDAP can quickly respond to each request and achieve better performance. In particular, DDAP can reduce energy consumption by 7.6% and running time cost by 63.2% on average compared to state-of-the-art methods.**

***Index Terms*—Network Function Virtualization, Deep Reinforcement Learning, Performance Interference.**

I. INTRODUCTION

Unlike traditional hardware-based middleboxes, network function virtualization (NFV), which has been a recent trend, can decouple the hardware and network functions and implement them in the form of software. NFV can deploy virtual network functions (VNFs) on commodity servers, which can reduce Capital Expenditure (CAPEX) and Operation Expense (OPEX) [1]. Encouraged by those advantages, some companies have begun to use VNFs instead of traditional middleboxes. Recently, the concept of Network-as-a-Service enables rapid network service commercialization [2]. Under this trend, NFV providers, who own or rent resources to build VNF instances to serve customers, have emerged as a very important role in the NFV eco-system.

Energy consumption has been a critical concern for data centers. With the increase of network traffic, the energy consumption of the infrastructure also occupies a high cost for the NFV providers who own the data center. Therefore, from the perspective of cost control and environmental protection, reducing energy consumption is very important. According to previous work [3], the energy consumption of enterprise servers accounts for 60% of the total energy consumption of the data center. Naturally, we can consolidate VNFs in fewer servers to maximize the utility of the servers and reduce the overall energy consumption. For example, Sun *et al.* [4] built a controller for flow migration to merge some underloaded VNFs to save energy. Gu *et al.* [5] designed a system for coordinating VNFs to maximize network utility to control energy consumption.

However, some researchers have proved that for VNFs running on the same server, hardware competition is very common which may lead to varying degrees of performance degradation [6]. Even in a multi-core CPU environment, co-located VNFs still compete for resources because of memory sharing and cache consistency. According to a measurement study [7], the performance degradation, which is measured relative to the performance achieved when each NF is run individually, can be as high as 35%. Co-located VNFs may compete with each other to obtain resources such as CPU and memory. For example, VNFs with encryption or compression operations placed on the same server may compete for CPU resources for computationally intensive tasks.

To reduce performance interference, each server should host a reasonable combination of VNFs. Clearly, it is impossible to place each VNF on each server individually because the energy consumption of the data center and the cost of the server should also be considered. For NFV providers, there is a tradeoff between energy consumption and performance interference when placing VNFs.

Therefore, we study how to place VNFs efficiently by jointly considering energy consumption and performance interference. Compared with previous work that usually ignores the performance interference between VNFs, we explicitly consider the energy consumption of each server and the performance interference of each network function in the VNF placement model. In our model, the performance interference of each function depends on the VNFs running on the same server. Therefore, when combined with various VNFs, the performance value of each VNF may be different. For example, as shown in the Fig. 1, if a new VNF is placed on a server that already has a VNF, the existing VNF may suffer from varying degrees of performance interference, such as case (a) or (b). However, if we choose to start a new server like case (c), the new server will contribute to more energy consumption.

In this paper, we propose a method that explicitly considers energy consumption and performance interference when placing VNFs. We adopt the supply and demand model proposed in the previous paper [7] to quantify the performance of

ISBN 978-3-903176-39-3© 2021 IFIP

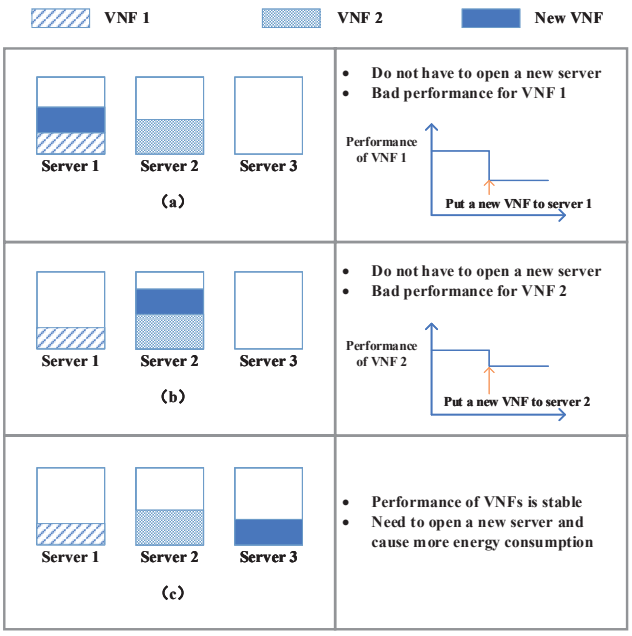

Fig. 1. Server selection cases: An example of three cases to place VNF

each VNF in the same server. According to a research [8], reinforcement learning is proven to achieve a near-optimal solution in the combinatorial optimization problem. Hence, unlike previous work that uses heuristic methods to optimize this problem, we propose a framework based on deep reinforcement learning (DRL) to process complex large-scale network state spaces in real-time.

We summarize our main contributions as follows:

- We aim to minimize the energy consumption of data center servers and formulate the VNFs placement model in real-time with lower performance interference. To illustrate the complexity of this problem, we prove the offline version of this problem is NP-complete by showing the bin-packing problem is polynomial-time reducible to it.
- For a homogeneous case where all servers are of the same type, we prove that when the performance of each VNF is greater than the upper limit L, First-Fit Heuristic (FFH) can be used as a solution and has a lower bound. For a more general case, we introduce the DDQN technique for VNF placement and name it DDAP (Deep Deterministic Automatic Placement), which is based on DRL.
- The experiment shows the DDAP surpasses the state-of-art method by 7.6% lower energy consumption and 63.2% lower running time cost on average.

The rest of the paper is organized as follows. In Section II, we review some of the related work on VNF placement DRL in NFV. Then we propose the problem definition and model formulation and prove this problem is NP-complete in section III. We also analyze the approximation performance of FFH by proving the lower bound value in a homogeneous environment in section IV. In section V, we introduce the detail of algorithm designing for the heterogeneous case. The experimental results of our approach and baseline are given in Section VI. Finally, Section VII concludes this paper.

II. Related work

A. VNF Placement

There have been many studies on the VNF placement problem from different objectives such as energy consumption and performance interference. Those problems are often investigated independently. For example, to reduce energy consumption, Sun *et al.* [4] design a flow controller to realize NFV elasticity control and merge underloaded VNFs for saving energy. Jia *et al.* [9] study on the dynamic deployment of VNF service chains across geo-distributed data centers for operational cost minimization including energy consumption. Kim *et al.* [10] propose an approach based genetic algorithm to minimize energy consumption.

The above studies are all about the optimization model of energy consumption and do not consider the performance interference of co-located VNFs. Recently, some studies have focused on how to place VNFs to avoid severe resource contention. For instance, Xu *et al.* [11] consider interference betweens co-located VMs, and proposed a lightweight interference-aware VM live migration strategy, using a demand-supply model to formulate interference. Zeng *et al.* [12] measured the throughput of VNFs as an indicator of performance among different types of co-located VNFs and prove that resource contends between co-located VNFs are very common. Zhang *et al.* [7] proposed AIA for the VNF placement problem which jointly considers interference between co-located VNFs, using a heuristic approach to solve the problem. Manousis *et al.* [6] developed a framework named SLOMO for multivariable performance prediction of VNFs. However, most of these algorithms are based on the heuristic method and none of them consider energy consumption and performance interference jointly.

B. DRL in NFV

In dynamic and complex real networks, DRL can solve these problems more intelligently than traditional heuristic algorithms. Therefore, there have been many recent studies applying DRL to make decisions in the NFV environment, *e.g.* [13]–[16]. Xiao *et al.* [13] proposed NFVdeep which is an online DRL to automatically deploy the VNF service chain for minimizing the operation cost of NFV providers and maximizing the total throughput of requests. Khezri *et al.* [14] also combined the DRL with NFV placement problem considering the reliability requirement of the services to make a decision. Gu *et al.* [15] designed a DRL framework for VNF orchestration for network utility maximization. Mao *et al.* [16] proposed a framework named DDQP based on DRL to place VNF service chains considering fault tolerance. Solozabal *et al.* [17] proposed a DRL agent to learn placement decision to minimize the overall power consumption. Nevertheless, co-located VNFs also suffered performance interference in those work, but they didn't consider it.

TABLE I
KEY NOTATION

Symbol	Description
F	Set of VNFs
S	Set of servers in the data center
F_j	Set of VNFs which have been placed in a server $j \in S$
M_j	The memory capacity of server $j \in S$
C_j	The CPU capacity of server $j \in S$
m_i	The memory requirement of VNF $i \in F$
c_i	The CPU requirement of VNF $i \in F$
$M_j^o(t)$	The occupied memory resource of server $j \in S$ at time slot t
$C_j^o(t)$	The occupied CPU resource of server $j \in S$ at time slot t
$P_i^j(t)$	Performance value of VNF i in server j at time slot t, $i \in F$, $j \in S$
L	Lower bound of performance
e_j	The idle energy consumption of server $j \in S$
e_j^c	energy consumption of each CPU in $j \in S$
a_i^t	1 if VNF $i \in F$ is in service at time slot t, 0 otherwise
x_{ij}	1 if VNF $i \in F$ is assigned to server $j \in S$, 0 otherwise
y_j^t	1 if any VNF runs on server $j \in S$ at time slot t, 0 otherwise

III. Model And Problem Definition

In this section, we first introduce the model and problem definition. Then we formulate the VNF placement problem. Finally, we prove that the offline version of this problem is NP-complete. The key notations are listed in Table I.

A. Model

In this paper, we mainly consider such a scenario that requests for deploying a VNF instance arrive from time to time, and we need to make a placement decision. We use F and S to represent a set of VNFs and a set of servers respectively, then $|F|$ and $|S|$ represent the number of VNFs and servers respectively. A request of building VNF instance can be denoted as (c_i, m_i), where c_i and m_i represent the CPU and memory requirements for VNF $i \in F$ respectively.

To adapt to the fluctuation of network traffic, we use time slot t to denote a constant period and formulate this problem in an online way. Every server has CPU resource and memory resource, which is denoted as C_j and M_j for each server $j \in S$. As the degree of performance interference may change in different periods, the performance value of VNF $i \in F$ on server $j \in S$ at time slot t is considered as $P_i^j(t)$, we use a demand-supply model to quantify it according to [7]. Then the equation is as below:

$$P_i^j(t) = k_0 + k_1 * \frac{c_i}{C_j^o(t)} + k_2 * \frac{m_i}{M_j^o(t)} \tag{1}$$

Where k_0, k_1 and k_2 are the coefficients, $C_j^o(t)$ and $M_j^o(t)$ represents the occupied CPU and memory resource of server $j \in S$ at time slot t respectively. We assume that the occupied resource of a server is constant during a time slot. This equation means that the more resource the VNF occupy, the more competitive it is in a co-located environment.

Let x_{ij} represents a binary decision variable. x_{ij} is set to one if VNF $i \in F$ is assigned to server $j \in S$, zero otherwise. Besides, we use a_i^t to indicate whether VNF $i \in F$ is in service at time slot t. Then the occupied CPU and memory resource of server $j \in S$ at time slot t can be represented as below:

$$\begin{aligned} C_j^o(t) &= \sum_{i=1}^{|F|} x_{ij} c_i a_i^t \\ M_j^o(t) &= \sum_{i=1}^{|F|} x_{ij} m_i a_i^t \end{aligned} \tag{2}$$

Since each server will consume energy once it starts to work, let y_j^t set to one if any VNF runs on server $j \in S$ at time slot t, zero otherwise. The main reason for server energy consumption is CPU, and it is linearly related to CPU utilization [18]. According to the energy consumption equation in [19], we use e_j, e_j^c to represent the idle energy consumption and energy consumption of each CPU unit in a time slot for server $j \in S$ respectively. Those parameters can be known in advance from the profile of each server. Then the energy consumed by server j at time t is calculated by:

$$E_j(t) = (e_j + e_j^c C_j^o(t)) y_j^t \tag{3}$$

So the overall energy consumption of the data center servers is expressed by:

$$E = \sum_{t=1}^{|T|} \sum_{j=1}^{|S|} E_j(t) \tag{4}$$

B. Formulation of Problem

The problem of VNF placement for minimizing energy consumption with a performance guarantee is defined as follows:

Problem 1: Requests of VNF deployment arrive from time to time, given a set of servers and threshold value L, both CPU and memory capacity of servers, the parameters about the energy consumption of servers, how to place each VNF on servers to minimize the consumption and guarantee the performance of VNFs not below the threshold value L?

Based on the problem defined, we can formulate the VNF placement model as follow. Among them, constraints (6) and (7) model the CPU and memory resource constraint of each server, respectively. In particular, we use F_j to denote the set of VNFs that have been placed in server $j \in S$. Equation (8) guarantees every VNF has and only has one server to place. Equation (9) can check if server $j \in S$ run any VNF. Since this model has a performance guarantee, constraint (10) requires that the performance of each VNF is no less than L, which represents a lower bound of performance value. Constraints (11) to (13) restrict the ranges of decision variable to 0 and 1.

$$\min \quad E \tag{5}$$

$$\text{s.t.} \quad \sum_{i=1}^{|F|} x_{ij} m_i a_i^t \le M_j, \qquad \forall j \in S, \forall t \tag{6}$$

$$\sum_{i=1}^{|F|} x_{ij} c_i a_i^t \le C_j, \qquad \forall j \in S, \forall t \tag{7}$$

$$\sum_{j=1}^{|S|} x_{ij} = 1, \qquad \forall i \in F \tag{8}$$

$$y_j^t = sgn(\sum_{i=1}^{|F|} x_{ij} a_i^t), \quad \forall j \in S, \forall t \tag{9}$$

$$P_i^j(t) \ge L, \qquad \forall i \in F_j, \forall t \tag{10}$$

$$a_i^t \in \{0,1\} \qquad \forall i \in F, \forall t \tag{11}$$

$$x_{ij} \in \{0,1\} \qquad \forall j \in S, \forall i \in F \tag{12}$$

$$y_j^t \in \{0,1\} \qquad \forall j \in S, \forall t \tag{13}$$

We study an online problem about how to place VNF in real-time. To show the complexity of this problem, we prove that the offline version of this problem is NF-complete.

C. NP-completeness

We define the offline version of this problem as follow: Given a set of servers, VNFs and threshold value L, both CPU capacity and memory of servers, parameters about the energy consumption of each server, how to place each VNF on servers to minimize the consumption and guarantee the performance of VNFs not below the threshold value L?

Theorem 1: The offline version of this problem is NP-complete.

Proof: First, we prove this problem is NP by certifying a given instance of which is feasible in a polynomial time. Therefore, we have to check whether it conforms to the constraints (6) to (10). For constraint (10), the computational time complexity is $\mathcal{O}(|F|^2|S|)$, which is greater than computing other constraints. In consequence, we can verify any instance of this problem is feasible in a polynomial-time of $\mathcal{O}(|F|^2|S|)$. So the offline version of this problem is NP.

Then, we prove the bin-packing problem can be reducible to this problem. We simplify the settings of servers to homogeneousness by setting CPU, memory resource, and parameters of energy consumption of each server to constant such as $e_j = e, \forall j \in S$ and $L = 0$. The whole energy consumption of servers can be expressed as:

$$\begin{aligned} &\sum_{j=1}^{|S|} (e_j + e_j^c C_j^o) y_j \\ =&e \sum_{j=1}^{|S|} y_j + e^c \sum_{j=1}^{|S|} C_j^o y_j \\ =&e \sum_{j=1}^{|S|} y_j + e^c \sum_{i=1}^{|F|} c_i \end{aligned} \tag{14}$$

The equation above shows that minimizing energy consumption is equal to minimize the number of the working server. Hence, This problem is as same as the bin-packing problem. According to proof in [20], bin-packing problem is np-complete. This means that the bin-packing problem is a subset of our original problem. So we can prove the offline version of the VNF placement problem is also NP-complete due to the NP-completeness of the bin-packing problem.

IV. A Solution in Homogeneous Case

As discussed in section III, when the given server is homogeneous, which means all servers have the same CPU and memory resource, this problem is also NP-complete. Therefore, we can use FFH algorithm to solve it. In FFH algorithm, it attempts to place VNF in the first server that can accommodate these constraints, otherwise, it starts a new server. The next theorem describes the performance lower bound of FFH compared to the optimal solution under certain circumstances.

Theorem 2: Given a set of servers with the same type, then the variable of each server is the same. In this situation, for all VNF instance, if $\min(k_0 + k_1 * \frac{c_i}{C} + k_2 * \frac{m_i}{M}) \ge L$, we use FFH as a solution and $FFH(E)$ to denote its energy consumption, $OPT(Y)$ is the value of $\sum_{t=1}^{|T|} \sum_{j=1}^{|S|} y_j^t$ in optimal solution. Then we have:

$$FFH(E) \le e\lfloor 1.7OPT(Y) \rfloor + e^c \sum_{t=1}^{|T|} \sum_{i=1}^{|F|} c_i a_i^t \tag{15}$$

Proof: First, we prove that the online version of energy consumption is similar to the bin-packing problem. The process of proof is similar to equation (14) above, from which we have:

$$E = e \sum_{t=1}^{|T|} \sum_{j=1}^{|S|} y_j^t + e^c \sum_{t=1}^{|T|} \sum_{i=1}^{|F|} c_i a_i^t \tag{16}$$

According to [21], the absolute approximation ratio for FFH to solve the bin-packing problem is $\lfloor 1.7OPT \rfloor$. Then we use mathematical induction to prove that bin-packing problem with duration and slot also conforms with this ratio. We use $FFH_T(Y)$ to denote the value of $\sum_{t=1}^{|T|} \sum_{j=1}^{|S|} y_j^t$ when FFH is the solution.

When slot $t = 1$, we have $FFH_1(Y) \le \lfloor 1.7OPT_1(Y) \rfloor$ since it is basically the offline situation. When slot $t = k > 1$, we assume that $FFH_k(Y) \le \lfloor 1.7OPT_k(Y) \rfloor$. And when slot $t = k + 1$, we have:

$$\begin{aligned} &\lfloor 1.7OPT_{k+1}(Y) \rfloor \\ =&\lfloor 1.7OPT_k(Y) + 1.7\frac{\sum_{i=1}^{|F|} c_i a_i^{k+1}}{C} \rfloor \\ \ge&\lfloor 1.7OPT_k(Y) \rfloor + \lfloor 1.7\frac{\sum_{i=1}^{|F|} c_i a_i^{k+1}}{C} \rfloor \\ \ge&FFH_k(Y) + \lfloor 1.7\frac{\sum_{i=1}^{|F|} c_i a_i^{k+1}}{C} \rfloor \\ \ge&FFH_{k+1}(Y) \end{aligned} \tag{17}$$

Hence we conclude that the bin-packing problem with duration and slot is also conformed with this value. Then put this conclusion above into equation (15), we have:

$$\begin{aligned} FFH(E) &= eFFH(Y) + e^c \sum_{t=1}^{|T|} \sum_{i=1}^{|F|} c_i a_i^t \\ &\leq e\lfloor 1.7OPT(Y) \rfloor + e^c \sum_{t=1}^{|T|} \sum_{i=1}^{|F|} c_i a_i^t \end{aligned} \tag{18}$$

V. General Algorithm in Heterogeneous Case

In complex network environments with different configurations of servers, traditional heuristic algorithms often can not work well. To make sure the algorithm more reliable, we decide to use DRL to solve this problem. In this section, we first present the Markov decision process (MDP) model, which is a mathematical model of sequential decision making and used to simulate the random strategy and rewards in an environment where the system state has Markov properties. Then we elaborate on the detail of this algorithm.

A. State, Action and Reward Design

With formulations above, we can present the MDP model as $< \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma >$, where $\mathcal{S}$ present the discrete states, $\mathcal{A}$ is the discrete actions, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S}$ is the transition probability distribution, $\mathcal{R} : \mathcal{S} \times \mathcal{A}$ is the reward function, finally, γ is a discount factor for future reward which ranges from 0 to 1.

State Definition. Each $s_i \in \mathcal{S}$ is defined as a vector (S_i, F_i). $S_i = (S_i^1, S_i^2, ..., S_i^{|S|})$ represents the current state of all servers. For example, $\forall j \in S$, we have:

$$\begin{aligned} S_i^j = (&S_{cpu}^{i,j}, S_{mem}^{i,j}, S_{Eidle}^{i,j}, S_{Ecpu}^{i,j}, S_{cpuutil}^{i,j}, \\ &S_{memutil}^{i,j}, S_{work}^{i,j}, S_{TTL}^{i,j}, S_{min}^{i,j}) \end{aligned} \tag{19}$$

which indicates the properties of each server, where the first six items present CPU resource, memory resource, idle energy consumption, CPU consumption, occupied CPU resource, occupied memory resource of server j respectively at state s_i. $S_{work}^{i,j}$ presents if server j is working and $S_{TTL}^{i,j}$ is the working duration of it. Since the performance of every original VNF instance on a server will decrease when a new VNF instance place on the same server. To check whether the performance of old VNFs are below L and ensure the length of states is fixed, we choose the characteristics of VNF which has the smallest performance value when the new VNF run on server j, and name it $S_{min}^{i,j}$. Besides, $F_i = (F_{cpu}^i, F_{mem}^i, F_{TTL}^i, L)$ reveals the characteristics of current VNF creating request, where F_{cpu}^i is the CPU demand, F_{mem}^i is the memory demand, F_{TTL}^i is the time-to-live of this VNF and L is the lower bound of performance respectively.

Action Definition. First, we label each server node as an integral index $j = 1, 2, ..., |S|$. Then we define our action space as $\mathcal{A} = \{1, 2, 3, ..., |S|\}$, and action $a \in \mathcal{A}$.

Reward Function. To minimize overall energy consumption, we define the reward function based on the selected server. When a server is chosen, the agent should consider whether: (1) the chosen server can satisfy capacity and performance constraints when placing the new VNF, (2) the chosen server is working before, and (3) the rest duration of the chosen server is longer than the required TTL of the new VNF.

For the purpose of making the agent convergence quickly, we remove the actions that do not meet the condition (1) before letting the agent chosen. For the rest conditions, we define the reward function as the negative value of extra consuming energy by choosing this server. Thus, we have:

$$r(s, a) = -(e_j^c c_i F_{TTL}^i + e_j t_i) \tag{20}$$

Where F_{TTL}^i is the time-to-live of VNF i. As for t_i, we use $s_{restTTL,j}$ to denote rest time-to-live of server j and have:

$$t_i = \begin{cases} F_{TTL}^i, & \text{when fail to meet condition (2)} \\ F_{TTL}^i - S_{restTTL,j} & \text{when fail to meet condition (3)} \\ 0, & \text{otherwise} \end{cases}$$

State Transition. We define the MDP state transition as (s_t, a_t, r_t, s_{t+1}), where s_t presents the current state and a_t presents the taken action for handling the placement decision for new VNF. Then reward r_t is as feed back, and s_{t+1} is new state.

B. Architecture Design

Based on the MDP model above, we can dynamically describe the variations of server instances and VNF instances. Then we need to use DRL to build an efficient VNF placement algorithm. For achieving a high reward, this algorithm should automatically decide suited actions for each state. Therefore, we propose an algorithm named DDAP, which is based on the DRL approach and placing VNFs in an online way with a performance guarantee.

There are three major types of DRL approach, which are value-based, policy-based, and model-based approaches respectively. In our situation, we need to find appropriate servers to place each VNF and the action space is discrete. Therefore, we adopt a value-based DRL approach. There are also many methods in the value-based DRL approach, such as Q-learning [22], which builds a look-up table to store action-values $Q(s, a)$ and select the action to get the maximum reward according to this table. But Q-learning needs to store all state-action values to look up which will take up a lot of time and memory. Therefore, Mnih *et al.* [23] design Deep Q-Networks (DQN) to tackle these problems. DQN uses Convolutional Neural Networks (CNN) to evaluate Q-function, and experience replay to improve the learning ability of CNN. But DQN also suffers from substantial overestimations in a large-scale environment. Hence, we use Double DQN (DDQN) [24] approach. Comparing with former algorithms, DDQN decouples the selection and calculation of the target Q value, which can reduce the overestimations of Q value and lead to much better performance.

The architecture of DDAP is illustrated in Fig. 2. Environment transmits the current states to the agent, who uses CNN to evaluate Q value of each action in the current state. Then

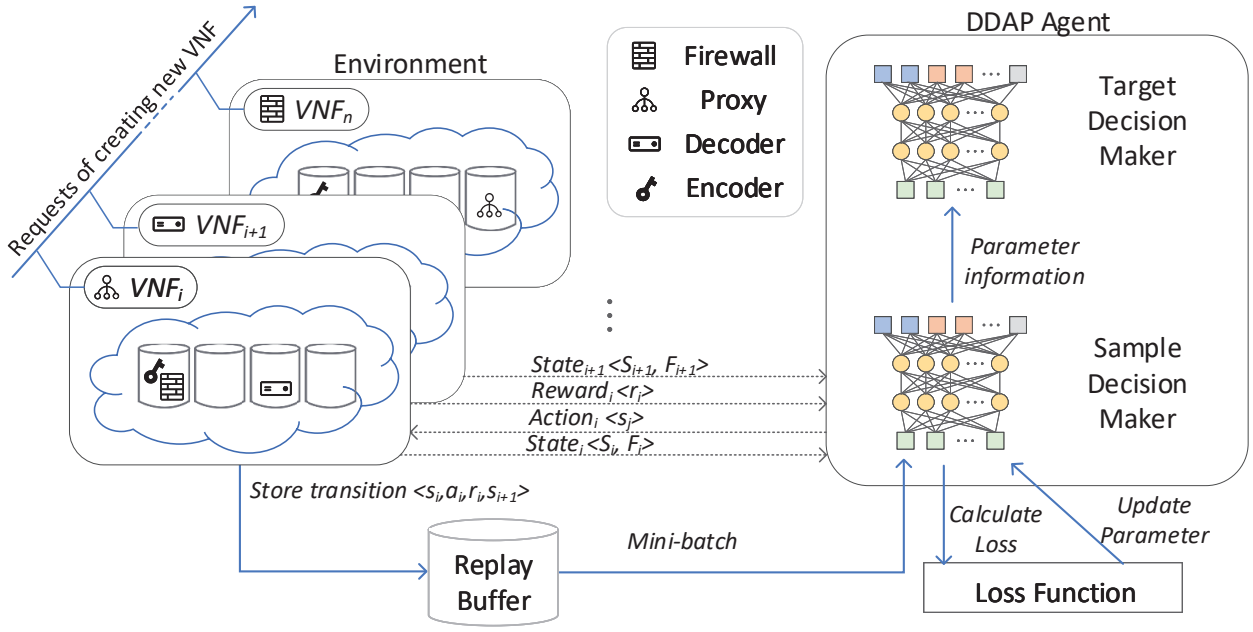

Fig. 2. The DDAP design structure

the agent decides an action according to Q value to maximize the expected return. Next, the environment will feedback a reward according to the result of executing the action. Since we adopt DDQN to train approach to learn the policy, there is a replay buffer to store experienced data and extract a portion of the data from it for updating parameter of CNN. The reason to use replay buffer is to broke the dependencies between experienced data so as to make Q function more reliable. Moreover, there should be two CNNs in the DDAP agent, sample network randomly chooses data from the replay buffer to train and use lose function to update parameters. And at every C step, the parameters of the target network are updated through the sample network.

The CNN design of Q function is shown in Fig. 3. The input layer is the state vector which includes the server instances and new VNF, and the output layer is the state-action value at the current state. As for the hidden layer, we use leaky ReLU [25] as the activation function.

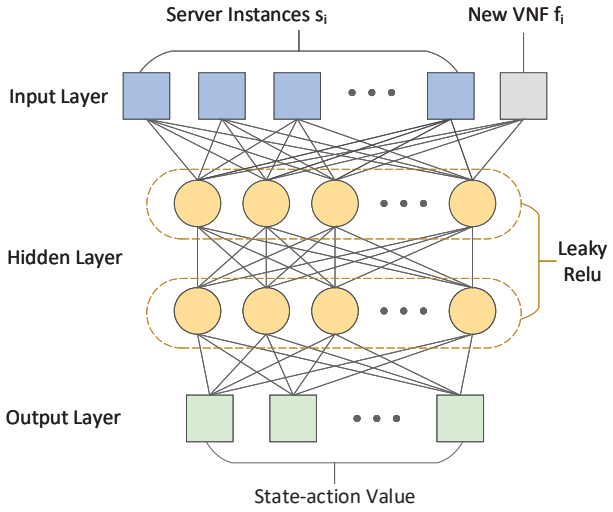

Fig. 3. The neural network design.

C. Training procedure

In order to effectively handle VNF placement online, we introduce the time slot as defined before. The whole placement procedure is listed in Algorithm 1. First, We choose a time slot that has at least one request to place VNF (lines 1 to 4). Then, we select the first request to place VNF i at the current time slot according to arriving time (line 5). For each step, we initiate state s_t (line 8). To make the agent converge faster, we re-define action space $\mathcal{A}$ before selecting an action. We check each action and remove those do not conform the equation (6), (7), (10) (lines 9 to 13). Next, we select an action a_t and execute it to get a reward r_t. The state s_t is transferred to s_{t+1} consequently (line 16). When there is no request left in the current time slot, we remove those expired servers and move to the next time slot (lines 17 to 20). Finally, we choose a new VNF and kick off another new step (lines 21 to 23).

Incorporating the above design, we summarize our training procedure based on DDQN in Algorithm 2. We first initialize the Q function network with random weights $Q(s, a; \theta)$ (line 1), and the target Q function network $\hat{Q}(s, a; \hat{\theta})$ is cloned

Algorithm 1 Placement approach

1: **Begin:** Initiate time slot $\tau \leftarrow 1$
2: **while** None of the request to place VNF at time slot τ **do**
3: $\tau \leftarrow \tau + 1$
4: **end while**
5: Initiate $i \leftarrow 1$
6: select a request to place VNF $i \in F$ from time slot τ based on arriving time.
7: **for** state $t \leftarrow 1, T$ **do**
8: Initiate state s_t.
9: **for** action $a \leftarrow \mathcal{A}$ **do**
10: **if** a doesn't conform equation (6), (7), (10) **then**
11: Removing action a from $\mathcal{A}$
12: **end if**
13: **end for**
14: Taking action a_t from $\mathcal{A}$ to place VNF
15: Calculate the reward r_t based on equation (20)
16: $s_t \leftarrow s_{t+1}$
17: **if** There is no request left in time slot τ **then**
18: Rescanning all servers and removing expired servers.
19: $\tau \leftarrow \tau + 1$
20: **end if**
21: $i \leftarrow i + 1$
22: select request of VNF i
23: **end for**

Algorithm 2 DDQN Based Training Procedure

1: **Begin:** Initialize Q-function Q with random weights θ
2: Initialize target Q-function $\hat{Q}$ with weights $\hat{\theta} = \theta$
3: Initialize replay buffer R to capacity N
4: **for** $episode \leftarrow 1, M$ **do**
5: Initialize the NFV environment and get the vector of state s_1.
6: **for** $t \leftarrow 1, T$ **do**
7: Select an action a_t randomly with the probability ϵ, otherwise select action based equation (21).
8: Execute action a_t and observe reward r_t based equation (20).
9: Transfer the state to s_{t+1}.
10: $< s_t, a_t, r_t, s_{t+1} > \rightarrow$ buffer R.
11: Sample random mini-batch of B transitions
12: **for** $j = 1$ to B **do**
13: $< s_j, a_j, r_j, s_{j+1} >$ from B transitions.
14: **if** episode terminates at step $j + 1$ **then**
15: Set $y_j = r_j$
16: **else**
17: Set $a' = \arg\max_{a'} Q(s_{j+1}, a, \theta)$
18: Set $y_j = r_j + \gamma \hat{Q}(s_{j+1}, a'; \hat{\theta})$
19: **end if**
20: **end for**
21: Update Q-function by minimizing the loss: $L = \frac{1}{B} \sum_{j=1}^{B} (y_j - Q(\phi_j, a_j; \theta))^2$.
22: Every C steps reset $\hat{Q} = Q$.
23: **end for**
24: **end for**

from the former (line 2). To break the data dependencies, we construct the replay buffer as R in line 3.

For each episode, we start the agent training with the initial environment and state s_1 (line 5). Then, in each step, we select an action based on ϵ-greedy method which can balance the exploitation of policy and the exploration of the environment. ϵ-greedy method means to select a_t randomly with a probability of ϵ, otherwise use equation (21) to choose the action which is the maximum Q value in the current state (line 7). Next, the action is executed to get a reward r_t based the reward equation (20) (line 8). Accordingly, the state will transfer to s_{t+1} and the transition sample is stored in the replay buffer R in lines 9 and 10.

$$a_t = \max_a Q(s_t, a; \theta) \tag{21}$$

After gathering enough experienced samples in buffer R, we sample a mini-batch of B transitions from it to train the Q function network (line 11). For the j-th transition sample $< s_j, a_j, r_j, s_{j+1} >$, we calculate the target value y_j from the target Q function network (line 12 to 20). This is the most critical step which is different from DQN: First, we choose an action that corresponds to the maximum Q value in the current Q network (line 17). Then, we use this action to calculate the target value in the target Q network (line 18). The target value is defined as:

$$Y = r + \gamma Q(s', \arg\max_a Q(s', a; \theta); \theta^-) \tag{22}$$

TABLE II
CONFIGURATION OF DATA CENTERS

Scale	#Servers	#VNFs	Duration	#Slots
Small	50	(100,150)	(1,5)	500
Middle	200	(400,500)	(1,10)	4000
Large	500	(1200,1500)	(1,30)	6000

Then, the parameter of the Q network can be updated by minimizing the loss (line 21). Finally, the parameter of the target Q network will be reset after C steps (line 22).

VI. EVALUATION AND ANALYSIS

In our numerical analysis, we compare our approach with other methods. Since the homogeneous environment is not universal and the result of it has been proved by theory, the experiment is based on the heterogeneous environment. In section VI-A, we introduce the specific parameters of the experiment including data, baselines, and simulation platform. Next, we show the experimental result and analyze the specific reasons in section VI-B.

A. Simulation Setup

Server instances: In our experiment, three scales that correspond to different numbers of servers and VNF instances. Details are shown in table II. Each server has CPU and

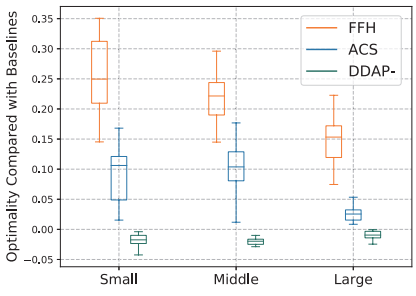

Fig. 4. The percentage improvement of DDAP compared to other algorithms.

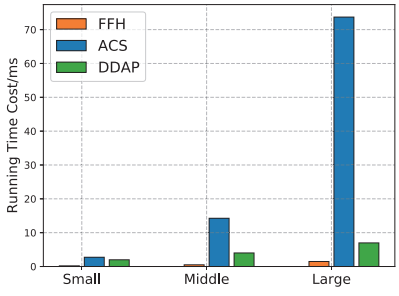

Fig. 5. Running time cost between different algorithms.

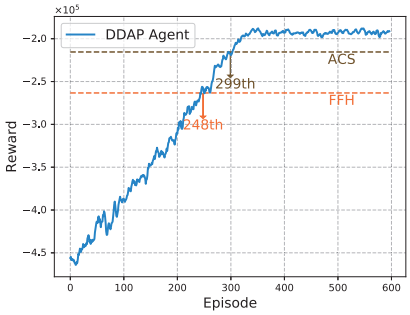

Fig. 6. Learning history under small scale.

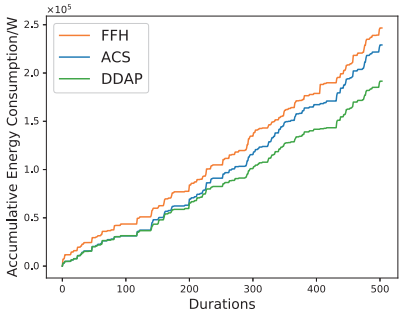

Fig. 7. Accumulative energy consumption under small scale where number of VNFs is 150.

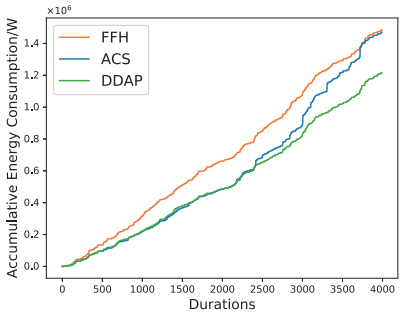

Fig. 8. Accumulative energy consumption under middle scale where number of VNFs is 500.

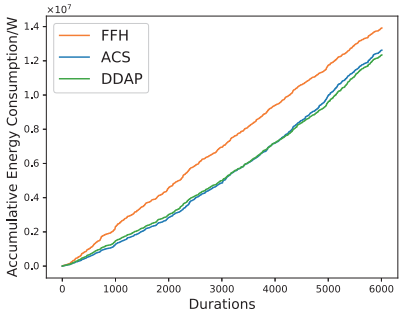

Fig. 9. Accumulative energy consumption under large scale where number of VNFs is 1500.

memory resources, idle energy consumption, and CPU consumption. According to [7], we set the CPU resource of each server ranges from 20 unit to 200 unit, the memory resource of each server ranges from 16 unit to 64 unit. As for the energy consumption of servers, we refer to [26] and randomly select the idle energy consumption of the server from [10, 30], the CPU energy consumption of each server from [50, 150]. The lower bound L is 0.9.

Requests of creating new VNFs: We construct requests for creating new VNF according to [27]. Each VNF has CPU and memory demand, the CPU demand is randomly selected from [2, 10], the memory demand is randomly selected from [1, 4].

Baseline and schemes compared: Firstly, we compare with FFH [28] algorithm and Ant Colony System (ACS) [19]. As we prove in section IV, FFH is a very competitive solution in a homogeneous case. As for the ACS algorithm, it is based on Ant-Colony System which is embedded with new heuristics. The ACS algorithm is presented for an energy-efficient solution to the optimization problem according to [19].

Simulation platform: We use a Python-based simulation framework, PyTorch library to construct the neural network of our architecture. All experiments are based on a workstation with 128GB RAM and an Intel (R) Core (TM) i7-8700 CPU with 6 cores 12 threads.

B. Performance Evaluation

To validate our approach, we compare the results with the FFH and ACS algorithms. Our approach fully considers the performance interference of each server, which may cause higher energy consumption. Therefore, we also use DRL to solve energy consumption without the performance guarantee and name it DDAP-.

Energy consumption: We divide model settings into three types according to the scale of the data center, and the detailed settings of the environments are listed in Table II. We run each set of experiments several times under different scales and the percentage improvement of DDAP compared to other algorithms is reflected in Fig.4. DDAP- only consumes 1.9%, 2.0%, 0.9% less energy averagely than DDAP under three scales. Considering the loss caused by performance interference such as throughput drop and SLA violation, the excess energy consumption can be ignored. Compared with the FFH algorithm, the improvement of DDAP is 25%, 22%, 15% averagely under three scales. DDAP also improved the energy consumption by 9.9%, 10.3%, 2.5% averagely compared with the ACS algorithm. Thus, we can conclude that DDAP has improved the energy consumption by 20.67% on average compared to the FFH and 7.6% to the ACS. The accumulative energy consumption under each scale is listed in Fig.7, 8, 9.

Online running cost: As the scale increases, the gap of energy consumption between DDAP and ACS gradually decreases. But we compare the running costs of the DDAP

and ACS in three scales in Fig.5, the DDAP spends much less time than ACS does. Although the DDAP has to train the neural network before deployment, it only needs to use the modeled NN to calculate and get actions after that. Hence the computation complexity of DDAP is $O(n)$, and so is FFH, where n denotes the number of servers. The computation complexity of ACS is $O(nmk)$, where m and k represent the number of iterations and ants respectively. Therefore, DDAP saves about 63.2% of running time overhead on average compared to the ACS algorithm. The running cost of DDAP is slightly higher than the FFH since DDAP needs to make wise decisions globally which spends more calculations than FFH does.

Training efficiency: To show our training efficiency, we demonstrate the training process on small scale in Fig 6. DDAP goes through a period of rapid ascent from the beginning, and then climbs slowly and converges eventually at around the 330th episode. It also can be seen that when DDAP is trained to the 248th episode, its energy consumption is lower than the FFH algorithm. When DDAP is trained to the 299th episode, its energy consumption is lower than the ACS algorithm.

VII. Conclusions

In this paper, we aim to minimize the energy consumption of the data center while considering the performance interference of co-located VNFs in NFV. First, we prove the complexity of this problem. Then, for a homogeneous environment, we prove that FFH has a performance lower bound. Furthermore, for heterogeneous environments, we design a DRL-based online framework to automatically place VNFs. Through our experiments, we have proved that our method performs well in diverse scenarios. In short, our method outperforms the state-of-art methods by lower energy consumption and running time cost on average.

Acknowledgments

The work was supported by the National Natural Science Foundation of China under Grant No. 61972101. Jin Zhao is the corresponding author.

References

[1] B. Yi, X. Wang, K. Li, S. K. Das, and M. Huang, "A comprehensive survey of network function virtualization," *Comput. Networks*, vol. 133, pp. 212–262, 2018.

[2] X. Fei, F. Liu, H. Xu, and H. Jin, "Adaptive vnf scaling and flow routing with proactive demand prediction," in *Proc. IEEE INFOCOM*, 2018, pp. 486–494.

[3] P. Arroba, J. M. Moya, J. L. Ayala, and R. Buyya, "Dynamic voltage and frequency scaling-aware dynamic consolidation of virtual machines for energy efficient cloud data centers," *Concurr. Comput. Pract. Exp.*, vol. 29, no. 10, p. e4067, 2017.

[4] C. Sun, J. Bi, Z. Meng, T. Yang, X. Zhang, and H. Hu, "Enabling nfv elasticity control with optimized flow migration," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 10, pp. 2288–2303, 2018.

[5] L. Gu, D. Zeng, S. Tao, S. Guo, H. Jin, A. Y. Zomaya, and W. Zhuang, "Fairness-aware dynamic rate control and flow scheduling for network utility maximization in network service chain," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 5, pp. 1059–1071, 2019.

[6] A. Manousis, R. A. Sharma, V. Sekar, and J. Sherry, "Contention-aware performance prediction for virtualized network functions," in *Proc. ACM Conf. Special Interest Group Data Commun. (SIGCOMM)*, 2020, pp. 270–282.

[7] Q. Zhang, F. Liu, and C. Zeng, "Adaptive interference-aware vnf placement for service-customized 5g network slices," in *Proc. IEEE INFOCOM*, 2019, pp. 2449–2457.

[8] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, "Neural combinatorial optimization with reinforcement learning," *arXiv preprint arXiv:1611.09940*, 2016.

[9] 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, 2018.

[10] S. Kim, S. Park, Y. Kim, S. Kim, and K. Lee, "VNF-EQ: dynamic placement of virtual network functions for energy efficiency and qos guarantee in NFV," *Clust. Comput.*, vol. 20, no. 3, pp. 2107–2117, 2017.

[11] F. Xu, F. Liu, L. Liu, H. Jin, B. Li, and B. Li, "iaware: Making live migration of virtual machines interference-aware in the cloud," *IEEE Trans. Comput.*, vol. 63, no. 12, pp. 3012–3025, 2014.

[12] C. Zeng, F. Liu, S. Chen, W. Jiang, and M. Li, "Demystifying the performance interference of co-located virtual network functions," in *Proc. IEEE INFOCOM*, 2018, pp. 765–773.

[13] Y. Xiao, Q. Zhang, F. Liu, J. Wang, M. Zhao, Z. Zhang, and J. Zhang, "Nfvdeep: Adaptive online service function chain deployment with deep reinforcement learning," in *Proc. ACM IWQoS*, 2019, pp. 1–10.

[14] H. R. Khezri, P. A. Moghadam, M. K. Farshbafan, V. Shah-Mansouri, H. Kebriaei, and D. Niyato, "Deep reinforcement learning for dynamic reliability aware nfv-based service provisioning," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, 2019, pp. 1–6.

[15] L. Gu, D. Zeng, W. Li, S. Guo, A. Y. Zomaya, and H. Jin, "Intelligent vnf orchestration and flow scheduling via model-assisted deep reinforcement learning," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 2, pp. 279–291, 2020.

[16] W. Mao, L. Wang, J. Zhao, and Y. Xu, "Online fault-tolerant vnf chain placement: A deep reinforcement learning approach," in *Networking*, 2020, pp. 163–171.

[17] R. Solozabal, J. Ceberio, A. Sanchoyerto, L. Zabala, B. Blanco, and F. Liberal, "Virtual network function placement optimization with deep reinforcement learning," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 2, pp. 292–303, 2020.

[18] A. Beloglazov, J. H. Abawajy, and R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing," *Future Gener. Comput. Syst.*, vol. 28, no. 5, pp. 755–768, 2012.

[19] F. Alharbi, Y.-C. Tian, M. Tang, W.-Z. Zhang, C. Peng, and M. Fei, "An ant colony system for energy-efficient dynamic virtual machine placement in data centers," *Expert Syst. Appl.*, vol. 120, pp. 228–238, 2019.

[20] M. R. Garey and D. S. Johnson, *Computers and intractability*. freeman San Francisco, 1979, vol. 174.

[21] G. Dósa and J. Sgall, "First fit bin packing: A tight analysis," in *STACS 2013*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2013.

[22] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[23] V. Mnih, K. Kavukcuoglu, D. Silver, and A. A. Rusu, "Human-level control through deep reinforcement learning," *Nat.*, vol. 518, no. 7540, pp. 529–533, 2015.

[24] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," *arXiv preprint arXiv:1509.06461*, 2015.

[25] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[26] M. Pedram and I. Hwang, "Power and performance modeling in a virtualized server system," in *ICPP Workshops 2010*, 2010, pp. 520–526.

[27] X. Zhang, C. Wu, Z. Li, and F. C. M. Lau, "Proactive vnf provisioning with multi-timescale cloud resources: Fusing online learning and online optimization," in *Proc. IEEE INFOCOM*, 2017, pp. 1–9.

[28] S. Kumaraswamy and M. K. Nair, "Bin packing algorithms for virtual machine placement in cloud computing: a review," *Int. J. Elect. Comput. Eng.*, vol. 9, no. 1, p. 512, 2019.