Deployment of Virtual Network Functions for Load Balancing and Cost Efficiency in NFV Systems

You-Chiun Wang and Cheng-Ying Tsai
Department of Computer Science and Engineering,
National Sun Yat-sen University, Kaohsiung, Taiwan
Email: ycwang@cse.nsysu.edu.tw; bernietw123456@gmail.com

***Abstract*—The emerging technique of *network function virtualization (NFV)* facilitates resource management by decoupling network functions from dedicated hardware devices and moving them to software appliances known as *virtual network functions (VNFs)*. Network services are carried out by *service function chains (SFCs)*, each with multiple VNFs. Given SFCs and *physical machines (PMs)*, this article formulates a VNF deployment problem that asks how to assign each SFC's VNFs to PMs with three objectives: maximizing the service ratio, balancing loads of PMs, and minimizing the communication cost to run SFCs. We then propose the *cost-aware and load-balancing VNF scheduling (CLVS)* scheme by assessing the suitability of selecting a PM to serve a VNF through a utility function. To increase the chance of successful deployment of VNFs and distribute workloads evenly among PMs, CLVS differentiates the deployment of the leading VNF from others in each SFC. When a PM is overloaded, some VNFs can be adaptively offloaded to light-load PMs. Simulation results show that our CLVS scheme attains a high service ratio, balances PMs' loads, and reduces the total communication cost.**

***Index Terms*—cost, deployment, load balance, network function virtualization, virtual network function.**

I. Introduction

In large networks, many network services are provided by *middleboxes*, purpose-built hardware devices. They manipulate and transform network traffic based on packet content. Service orchestration relies on steering data to pass selected middleboxes to process in order, referred to as *service function chains (SFCs)*. This leads network functions to couple with middleboxes tightly and thus brings challenges [1]. First, middleboxes are expensive and require domain knowledge to maintain. Second, integrating customized network functionalities may be time-consuming. Third, changing middleboxes to adjust SFCs comes at a high cost after deployment. Resource management becomes rigid as a result.

Network function virtualization (NFV) leverages the virtualization technology to replace traditional middleboxes with software appliances called *virtual network functions (VNFs)*. VNFs can be easily installed and run on commodity, general-purpose *physical machines (PMs)*, thereby decoupling network functions from underlying hardware. They are capable of moving between PMs. Moreover, users can grow or downsize SFCs by dynamically adding or removing VNFs. NFV provides great scalability, simplifies resource management, and reduces capital as well as operating expenditures. It plays an indispensable role in beyond 5G systems [2].

The assignment of VNFs to PMs for execution, known as the *VNF deployment issue*, plays a critical role in determining the performance of NFV systems [3]. Many methods consider maximizing the resource utilization of PMs. Doing so helps increase the *service ratio*, the ratio of the number of VNFs successfully deployed to the number of VNFs in all SFCs. These methods tend to use fewer PMs to handle VNFs, but they could result in uneven loads between PMs. Some PMs handle many VNFs, keeping them busy and impairing performance, while others remain idle. To achieve load balance, a few studies shift VNFs from heavy-load PMs to light-load PMs. They do not, however, account for communication costs between PMs in which VNFs of an SFC are deployed. The communication cost to complete the SFC may significantly increase, as some of its VNFs are dispersed over far-off PMs.

In light of this, the article proposes the *cost-aware and load-balancing VNF scheduling (CLVS)* scheme for the VNF deployment problem with three goals: 1) maximizing the service ratio, 2) balancing loads among PMs, and 3) decreasing the total communication cost used to complete SFCs. A utility function is used to measure the suitability of assigning one VNF to a PM. Moreover, the deployment of the leading VNF in each SFC is distinguished from that of other VNFs to boost the service ratio and balance loads between PMs. If a PM is overloaded, CLVS picks out some VNFs and moves them to nearby PMs (i.e., VNF offloading). CLVS considers reducing an SFC's communication cost when deploying and offloading VNFs. Using simulations, we show that the CLVS scheme can keep a high service ratio, achieve load balance for PMs, and reduce the total communication cost of SFCs.

The rest of this article is organized as follows: Section II discusses related work, and Section III describes the system model. In Section IV, we detail the CLVS scheme, followed by performance evaluation in Section V. Finally, Section VI draws a conclusion and presents future work.

II. Related Work

Various VNF deployment issues have been discussed. The work [4] employs network slicing for VNF deployment in a hybrid cloud with a central cloud and multiple edge clouds. By associating each SFC with a deadline, the study [5] routes traffic flows along that SFC as well as places its VNFs to meet the deadline. Given SFCs composed of VNFs and physical network functions, the deployment problem is formulated in [6] using mixed integer linear programming to minimize both cost and latency. The work [7] proposes a scaling method to adjust resources given to an SFC based on resource demands of VNFs. Obviously, they have different objectives from ours.

Many studies aim to raise the resource utilization of PMs. In [8], an integer linear program is used in VNF deployment,

whose objective is to increase resource usage and provider revenue. To reduce the complexity, only a few PMs are picked to deploy VNFs. Based on the norm-based criteria adopted in virtual machine placement, Pham et al. [9] select a PM with suitable resources to meet each SFC's demand. The study [10] applies deep reinforcement learning to VNF deployment, which improves the service ratio and decreases working PMs. Both [11] and [12] consider optimizing resource consumption of PMs while ensuring service latency when deploying VNFs. In [13], each SFC is given a weight to reflect importance. The first-fit, best-fit, and segment-based methods are designed to raise the total weight of deployed SFCs. In [14], an offline method finds the minimum number of PMs used to deploy VNFs. Then, potential mispredictions in incoming workloads of PMs are corrected by an online method. However, when PMs possess more resources, these methods may still make VNFs concentrate on some PMs to improve resource utilization. Doing so causes imbalanced loads among PMs.

The issue of moving VNFs between PMs (known as *VNF migration*) receives attention. Jahromi et al. [15] place VNFs in a content delivery network. Some VNFs migrate to other PMs to save the reconfiguration cost. The work [16] performs VNF migration to satisfy service-specific demands in a 5G network-slicing system. Abdelaal et al. [17] let VNF migrate to keep high availability against PM failures. A few studies leverage VNF migration to balance workloads of PMs. When a PM is overloaded, the study [18] chooses another light-load PM to take over some VNFs. The work [19] scores each PM (used to deploy VNFs) based on the resource states of the PM and its neighbors. It also conducts VNF migration when PMs become overloaded. However, both [18] and [19] do not consider the communication cost of SFCs.

III. System Model

Let us model the system as a connected graph $G = (\hat{P}, \hat{E})$. The vertex set $\hat{P}$ contains PMs, and the edge set $\hat{E}$ includes links between PMs. For each PM $p_x \in \hat{P}$, $\Gamma_x^{\mathtt{R}}$ is its capacity of type-$\mathtt{R}$ resources, where $\Gamma_x^{\mathtt{R}} \in \mathbb{Z}^+$ and $\mathtt{R} \in \{\mathtt{C}, \mathtt{M}\}$. Here, $\mathtt{C}$ and $\mathtt{M}$ are the types of computing (e.g., CPU) and memory resources, common types of resources required by VNFs [20]. If two PMs, p_x and p_y, are directly connected, there is a link $e_{x,y}$ in $\hat{E}$. The link has a communication cost $c_{x,y}$.

There is a set $\hat{S}$ of SFCs to be served. Each SFC $s_i \in \hat{S}$ consists of a series of VNFs $\hat{V}_i = \{v_{i,1}, v_{i,2}, \cdots, v_{i,m}\}$. Besides, $\gamma_{i,j}^{\mathtt{R}}$ signifies the number of type-$\mathtt{R}$ resources required by each VNF $v_{i,j}$ in $\hat{V}_i$. An indicator $z_{i,j}^x$ is used to check if VNF $v_{i,j}$ is deployed on PM p_x ($z_{i,j}^x = 1$ if so or $z_{i,j}^x = 0$ otherwise). The first VNF of each SFC s_i is referred to as s_i's *leading VNF*. Let $\varphi_{j,k}^i$ denote the *aggregate communication cost (ACC)* between two PMs on which VNFs $v_{i,j}$ and $v_{i,k}$ (of the same SFC) are deployed. Suppose that the shortest path between these two PMs contains a subset $\hat{E}'$ of links in $\hat{E}$. We derive that $\varphi_{j,k}^i = \sum_{e_{x,y} \in \hat{E}'} c_{x,y}$. If $v_{i,j}$ and $v_{i,k}$ are deployed on the same PM, we have $\varphi_{j,k}^i = 0$ (i.e., no ACC).

The VNF deployment problem asks how to assign PMs in $\hat{P}$ to handle VNFs of each SFC in $\hat{S}$ with three objectives:

$$\text{maxmize} \quad \frac{\sum_{p_x \in \hat{P}} \sum_{s_i \in \hat{S}} \sum_{v_{i,j} \in \hat{V}_i} z_{i,j}^x}{\sum_{s_i \in \hat{S}} |\hat{V}_i|}, \tag{1}$$

TABLE I
Summary of acronyms.

acronym	full name
ACC/RCC	aggregate/required communication cost
AVO	adaptive VNF offloading
BFSP	best-fit SFC placement
CLVS	cost-aware and load-balancing VNF scheduling
EVD	efficient VNF deployment
GRD	general resource demand
HCD/HMD	high computing/memory resource demand
LBVD	load-balanced VNF deployment
LVD/SVD	leading/subsequent VNF deployment
NFV	network function virtualization
PM	physical machine
SFC	service function chain
VNF	virtual network function

$$\text{maxmize} \quad \left(\sum_{p_x \in \hat{P}} l_x\right)^2 / \left(|\hat{P}| \sum_{p_x \in \hat{P}} l_x^2\right), \tag{2}$$

$$\text{minimize} \quad \sum_{s_i \in \hat{S}} \sum_{j=1}^{|\hat{V}_i|-1} \varphi_{j,j+1}^i, \tag{3}$$

subject to

$$c_{\mathtt{min}} \le c_{x,y} \le c_{\mathtt{max}}, \forall e_{x,y} \in \hat{E}, \tag{4}$$

$$z_{i,j}^x \in \{0,1\}, \forall v_{i,j} \in \hat{V}_i, \forall s_i \in \hat{S}, \forall p_x \in \hat{P}, \tag{5}$$

$$\gamma_{i,j}^{\mathtt{R}} \le \min_{p_x \in \hat{P}} \{\Gamma_x^{\mathtt{R}}\}, \forall v_{i,j} \in \hat{V}_i, \forall s_i \in \hat{S}, \forall \mathtt{R} \in \{\mathtt{C}, \mathtt{M}\}, \tag{6}$$

$$\sum_{p_x \in \hat{P}} z_{i,j}^x \le 1, \forall v_{i,j} \in \hat{V}_i, \forall s_i \in \hat{S}, \tag{7}$$

$$\sum_{s_i \in \hat{S}} \sum_{v_{i,j} \in \hat{V}_i} z_{i,j}^x \gamma_{i,j}^{\mathtt{R}} \le \Gamma_x^{\mathtt{R}}, \forall p_x \in \hat{P}, \forall \mathtt{R} \in \{\mathtt{C}, \mathtt{M}\}. \tag{8}$$

Regarding objective functions, Eq. (1) maximizes the service ratio, the ratio of successfully deployed VNFs to total VNFs in $\hat{S}$. Eq. (2) uses Jain's fairness index [21] to evaluate the degree of load balance between PMs. In particular, l_x is the amount of load of each PM p_x in $\hat{P}$:

$$l_x = \frac{1}{|\mathtt{R}|} \sum_{\mathtt{R} \in \{\mathtt{C}, \mathtt{M}\}} \left(\frac{1}{\Gamma_x^{\mathtt{R}}} \sum_{s_i \in \hat{S}} \sum_{v_{i,j} \in \hat{V}_i} z_{i,j}^x \gamma_{i,j}^{\mathtt{R}} \right). \tag{9}$$

In Eq. (3), $\sum_{j=1}^{|\hat{V}_i|-1} \varphi_{j,j+1}^i$ gives the total communication cost between PMs needed to be spanned to complete s_i. Below, it is called s_i's *required communication cost (RCC)*. Evidently, Eq. (3) is to minimize the total RCC of all SFCs.

For constraints, Eq. (4) gives lower and upper bounds on the communication cost of each link $e_{x,y}$. Eq. (5) points out that $z_{i,j}^x$ is an indicator whose value is either zero or one. Eq. (6) implies that the number of resources (for computing and memory) requested by a VNF cannot exceed the capacity of any PM. In Eq. (7), a VNF cannot be split and deployed on different PMs. Then, Eq. (8) means that the total number of resources asked by these VNFs cannot be more than the PM's capacity. Table I lists acronyms, while Table II summarizes notations used in this article.

IV. The Proposed CLVS Scheme

Let $u_x^{\mathtt{R}}$ denote the number of consumed type-$\mathtt{R}$ resources of $p_x \in \hat{P}$, as calculated by $u_x^{\mathtt{R}} = \sum_{s_i \in \hat{S}} \sum_{v_{i,j} \in \hat{V}_i} z_{i,j}^x \gamma_{i,j}^{\mathtt{R}}$. Like [18] and [19], we adopt a *warning threshold* $\delta_{\mathtt{R}}$ on the type-$\mathtt{R}$ resource consumption ratio for each PM, where $0.5 < \delta_{\mathtt{R}} < 1$. When $u_x^{\mathtt{R}}/\Gamma_x^{\mathtt{R}} > \delta_{\mathtt{R}}$, for $\mathtt{R} = \mathtt{C}$ or $\mathtt{R} = \mathtt{M}$,

TABLE II
SUMMARY OF NOTATIONS.

notation	definition
$\hat{P}, \hat{P}_{\texttt{can}}, \hat{P}_{\texttt{edg}}$	sets of all PMs, candidate PMs, and edge PMs
$\hat{D}_x, \hat{M}_x$	sets of PMs whose ACCs to PM $p_x \in \hat{P}$ do not exceed thresholds $\sigma_{\texttt{dep}}$ and $\sigma_{\texttt{mig}}$
$\hat{S}$	set of SFCs ($\hat{V}_i$: series of VNFs in SFC $s_i \in \hat{S}$)
$\Gamma_x^{\texttt{R}}$	capacity of type-R resources owned by p_x
$c_{x,y}$	communication cost of link $e_{x,y}$
$z_{i,j}^x$	indicator to check if VNF $v_{i,j} \in \hat{V}_i$ is served by p_x
$\gamma_{i,j}^{\texttt{R}}$	the number of type-R resources needed by $v_{i,j}$
$\delta_{\texttt{R}}$	warning threshold on type-R resource consumption
$u_x^{\texttt{R}}$	the number of consumed type-R resources of p_x

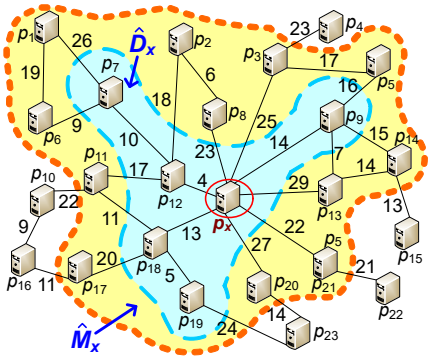


Fig. 1. Sets $\hat{D}_x$ and $\hat{M}_x$ for PM p_x, where $\sigma_{\texttt{dep}} = 20$ and $\sigma_{\texttt{mig}} = 40$.

Algorithm 1: LVD algorithm

Data: $v_{i,1}$: SFC s_i's leading VNF
Result: p_y: the target PM for deploying $v_{i,1}$

$f_{\texttt{edg}} \leftarrow$ false, $f_{\texttt{emp}} \leftarrow$ false, $\hat{P}_{\texttt{can}} \leftarrow \emptyset$;
for $p_x \in \hat{P}$ **do**
  if $\Gamma_x^R - u_x^R \geq \gamma_{i,1}^R$, $\forall R \in \{C, M\}$ **then**
    if $u_x^R = 0$, $\forall \texttt{R} \in \{C, M\}$ **then**
      if $p_x \in \hat{P}_{\texttt{edg}}$ **then**
        if $f_{edg} = false$ **then**
          $f_{\texttt{edg}} \leftarrow$ true and $\hat{P}_{\texttt{can}} \leftarrow \emptyset$;
        $\hat{P}_{\texttt{can}} \leftarrow \hat{P}_{\texttt{can}} \cup \{p_x\}$;
      else if $f_{edg} = false$ **then**
        if $f_{emp} = false$ **then**
          $f_{\texttt{emp}} \leftarrow$ true and $\hat{P}_{\texttt{can}} \leftarrow \emptyset$;
        $\hat{P}_{\texttt{can}} \leftarrow \hat{P}_{\texttt{can}} \cup \{p_x\}$;
    else if $f_{emp} = false$ *and* $f_{edg} = false$ **then**
      $\hat{P}_{\texttt{can}} \leftarrow \hat{P}_{\texttt{can}} \cup \{p_x\}$;
if $\hat{P}_{\texttt{can}} = \emptyset$ **then**
  return null;
$p_y \leftarrow \arg\max_{p_x \in \hat{P}_{\texttt{can}}} \xi(v_{i,1}, p_x)$;
$z_{i,1}^y \leftarrow 1$ and $u_y^{\texttt{R}} \leftarrow u_y^{\texttt{R}} + \gamma_{i,1}^{\texttt{R}}$, $\forall \texttt{R} \in \{\texttt{C}, \texttt{M}\}$;
return p_y;

p_x is considered *overloaded*. An overloaded PM is capable of accommodating VNFs provided that it can still meet the constraint in Eq. (8). However, we should avoid making PMs overloaded.

To facilitate the selection of PMs to handle VNFs, given a VNF $v_{i,j}$ to be deployed, we use a *utility function* $\xi(v_{i,j}, p_x)$ for PM p_x (detailed in Section IV-A). Then, CLVS sequentially deploys each VNF of an SFC s_i. Except for the leading VNF, the deployment of each VNF $v_{i,j}$ will depend on its previous VNF $v_{i,j-1}$ in $\hat{V}_i$. The location of the leading VNF affects the chance of successful deployment of the remaining VNFs in $\hat{V}_i$. As a result, we distinguish the deployment of the leading VNF from other VNFs. The leading VNF and other VNFs are assigned to PMs using the *leading VNF deployment (LVD)* and *subsequent VNF deployment (SVD)* algorithms, as discussed in Sections IV-B and IV-C.

Afterward, we check if the selected PM is overloaded after serving that VNF. If so, the *adaptive VNF offloading (AVO)* algorithm, as detailed in Section IV-D, is employed to move the PM's VNF to another PM for achieving load balance. This iteration is repeated until all VNFs in $\hat{V}_i$ have been checked.

A. Utility Function

The utility function $\xi(v_{i,j}, p_x)$ helps quantify the suitability of deploying VNF $v_{i,j}$ on PM p_x. Let us denote by $\hat{D}_x$ the set of PMs whose ACCs to p_x are no larger than a threshold $\sigma_{\texttt{dep}}$, where $\hat{D}_x \subseteq \hat{P} \setminus \{p_x\}$. Fig. 1 gives an example, where $\hat{D}_x = \{p_7, p_9, p_{12}, p_{18}, p_{19}\}$ when $\sigma_{\texttt{dep}}$ is set to 20. Then, the utility function is defined by

$$\xi(v_{i,j}, p_x) = \min_{\texttt{R} \in \{\texttt{C},\texttt{M}\}} H(p_x, v_{i,j}, \texttt{R}) + \sum_{p_y \in \hat{D}_x} \min_{\texttt{R} \in \{\texttt{C},\texttt{M}\}} H(p_y, v_{i,j+1}, \texttt{R}), \quad (10)$$

where $H(p_x, v_{i,j}, \texttt{R})$ is the ratio of residual type-R resources for p_x after serving $v_{i,j}$:

$$H(p_x, v_{i,j}, \texttt{R}) = \max\left\{(\Gamma_x^{\texttt{R}} - u_x^{\texttt{R}} - \gamma_{i,j}^{\texttt{R}})/\Gamma_x^{\texttt{R}}, 0\right\}. \quad (11)$$

The higher the value of $\xi(v_{i,j}, p_x)$, the more likely p_x is to be selected for handling $v_{i,j}$. Note that in Eq. (10), if $v_{i,j+1}$ does not exist in $\hat{V}_i$, we have $H(p_y, v_{i,j+1}, \texttt{R}) = 0$.

Our utility function has three considerations:

- If p_x's load is light, it has more resources left after serving $v_{i,j}$ (i.e., with a larger $H(p_x, v_{i,j}, \texttt{R})$ value). This increases the possibility to select p_x to serve $v_{i,j}$. Eq. (10) picks out the one with the least remaining resources in computing or memory (i.e., $\min_{\texttt{R} \in \{\texttt{C},\texttt{M}\}} H(p_x, v_{i,j}, \texttt{R})$), as it is p_x's bottleneck resource.
- When there is a VNF $v_{i,j+1}$ after $v_{i,j}$ in the same SFC, we deploy $v_{i,j+1}$ on a PM in $\hat{D}_x$, whose ACC to p_x is no larger than $\sigma_{\texttt{dep}}$. Doing so helps balance loads between PMs since $v_{i,j}$ and $v_{i,j+1}$ are deployed on different PMs. Moreover, we can limit the RCC spent to run the SFC.
- If $\sum_{p_y \in \hat{D}_x} \min_{\texttt{R} \in \{\texttt{C},\texttt{M}\}} H(p_y, v_{i,j+1}, \texttt{R})$ is larger, there are more candidates in $\hat{D}_x$ to deploy $v_{i,j+1}$. Hence, we prefer deploying $v_{i,j}$ on p_x, as the chance to successfully deploy $v_{i,j}$'s next VNF (i.e., $v_{i,j+1}$) can also increase.

B. LVD Algorithm

In each SFC $s_i \in \hat{S}$, its leading VNF $v_{i,1} \in \hat{V}_i$ is deployed using the LVD algorithm. Since the deployment of $v_{i,1}$ is not affected by the locations of other VNFs in s_i, we can search PMs in $\hat{P}$, thereby deploying $v_{i,1}$ more flexibly. To raise the

service ratio and make PMs' loads balanced (i.e., objectives in Eqs. (1) and (2)), we prefer deploying $v_{i,1}$ on an *idle PM* that currently has no VNF to process.

On the other hand, $v_{i,1}$'s location may affect the deployment of subsequent VNFs in s_i due to the RCC consideration in Eq. (3). Specifically, the leading or last VNF in an SFC is better deployed on an *edge PM*, which is a PM with only one neighbor. Doing so helps reduce the RCC of that SFC. Take Fig. 1 as an example. Suppose that an SFC s_i has three VNFs $\hat{V}_i = \{v_{i,1}, v_{i,2}, v_{i,3}\}$. When $v_{i,1}$, $v_{i,2}$, and $v_{i,3}$ are deployed on PMs p_{14}, p_{15} (i.e., an edge PM), and p_{13}, the order to steer data between PMs to complete s_i is $p_{14} \rightarrow p_{15} \rightarrow p_{14} \rightarrow p_{13}$. Hence, s_i's RCC is $13+13+14 = 40$. If we deploy $v_{i,1}$, $v_{i,2}$, and $v_{i,3}$ on p_{15}, p_{14}, and p_{13}, the order is $p_{15} \rightarrow p_{14} \rightarrow p_{13}$, and s_i's RCC decreases to $13 + 14 = 27$. In practice, it is difficult to decide in advance where to deploy the last VNF of an SFC. As a result, it is naturally a better choice to deploy the leading VNF on an edge PM.

Let $\hat{P}_{\texttt{edg}}$ be the set of edge PMs. Moreover, $\hat{P}_{\texttt{can}}$ denotes the set of candidate PMs for deploying the current VNF. Algorithm 1 gives LVD's pseudocode, which uses two boolean variables: 1) $f_{\texttt{edg}}$ indicates whether there are edge PMs that are also idle, and 2) $f_{\texttt{emp}}$ reveals whether there exist idle PMs, but none of them are edge PMs. Note that $f_{\texttt{edg}}$ and $f_{\texttt{emp}}$ cannot be true at the same time. Then, the for-loop in lines 2–14 iteratively checks whether each selectable PM can be added to $\hat{P}_{\texttt{can}}$. If a PM p_x has enough residual resources to meet $v_{i,1}$'s demand (i.e., $\Gamma^{\mathrm{R}}_x - u^{\mathrm{R}}_x \geq \gamma^{\mathrm{R}}_{i,1}$, $\forall \mathrm{R} \in \{\mathtt{C}, \mathtt{M}\}$, as line 3 shows), it is a selectable PM. Depending on $f_{\texttt{edg}}$ and $f_{\texttt{emp}}$, there are three mutually exclusive cases for finding candidate PMs $\hat{P}_{\texttt{can}}$ from selectable PMs (case 1 is the top priority, followed by cases 2 and 3):

Case 1. $f_{\texttt{edg}} = \texttt{true}$ and $f_{\texttt{emp}} = \texttt{false}$: There are some edge PMs that are idle, so $\hat{P}_{\texttt{can}}$ contains such PMs. The code is presented in lines 4–8.

Case 2. $f_{\texttt{edg}} = \texttt{false}$ and $f_{\texttt{emp}} = \texttt{true}$: None of the edge PMs are idle, but some non-edge PMs are idle. These idle PMs are included in $\hat{P}_{\texttt{can}}$. The code is given in lines 9–12.

Case 3. $f_{\texttt{edg}} = \texttt{false}$ and $f_{\texttt{emp}} = \texttt{false}$: No PM is idle. In this case, $\hat{P}_{\texttt{can}}$ contains all selectable PMs that are non-idle. The code is shown in lines 13–14.

If $\hat{P}_{\texttt{can}} = \emptyset$, meaning that none of the PMs in $\hat{P}$ has enough resources to handle $v_{i,1}$, LVD returns a null value (i.e., lines 15–16). Otherwise, among all candidate PMs in $\hat{P}_{\texttt{can}}$, line 17 picks PM p_y with the maximum utility value. In line 18, we set $z^y_{i,1} = 1$ to indicate that p_y will serve $v_{i,1}$ and update the number of consumed resources of p_y.

Theorem 1. *Let ζ_P be the number of PMs in $\hat{P}$. The LVD algorithm has a time complexity of $O(\zeta_P \zeta_D)$, where ζ_D is the maximum number of PMs in $\hat{D}_x$, $\forall p_x \in \hat{P}$.*

Proof. According to Algorithm 1, the for-loop in lines 2–14 checks each PM in $\hat{P}$ once, which spends $O(\zeta_P)$ time. In line 17, it requires $O(\zeta_D + 1)$ time to compute $\xi(v_{i,1}, p_x)$ by using Eq. (10). Since there are $|\hat{P}_{\texttt{can}}|$ PMs to be checked, line 17 consumes time of $|\hat{P}_{\texttt{can}}| \times O(\zeta_D + 1)$. All other statements take a constant time. The worst case occurs when $\hat{P}_{\texttt{can}} = \hat{P}$. Hence, the time complexity is $O(\zeta_P) + |\hat{P}_{\texttt{can}}| \times O(\zeta_D + 1) + O(1) = O(\zeta_P \zeta_D)$. □

Algorithm 2: SVD algorithm

Data: $v_{i,j}$: SFC s_i's VNF, where $j > 1$
Result: p_y: the target PM for deploying $v_{i,j}$

1 $f_{\texttt{idl}} \leftarrow$ false, $\hat{P}_{\texttt{can}} \leftarrow \emptyset$, $u_{\texttt{max}} \leftarrow 0$, $p_y \leftarrow$ null;
2 **for** $p_x \in \hat{P}$ **do**
3 **if** $z^x_{i,j-1} = 1$ **then**
4 $p_{\texttt{pre}} \leftarrow p_x$ and break;
5 **for** $p_x \in \hat{D}_{\texttt{pre}}$ **do**
6 **if** $\Gamma^R_x - u^R_x \geq \gamma^R_{i,j}$, $\forall R \in \{C, M\}$ **then**
7 **if** $u^R_x = 0$, $\forall R \in \{C, M\}$ **then**
8 **if** $f_{idl} = \mathit{false}$ **then**
9 $f_{\texttt{idl}} \leftarrow$ true and $\hat{P}_{\texttt{can}} \leftarrow \emptyset$;
10 $\hat{P}_{\texttt{can}} \leftarrow \hat{P}_{\texttt{can}} \cup \{p_x\}$;
11 **else if** $f_{idl} = \mathit{false}$ **then**
12 $\hat{P}_{\texttt{can}} \leftarrow \hat{P}_{\texttt{can}} \cup \{p_x\}$;
13 **if** $\hat{P}_{\texttt{can}} = \emptyset$ **then**
14 **return** null;
15 **for** $p_x \in \hat{P}_{\texttt{can}}$ **do**
16 **if** $\xi(v_{i,j}, p_x) > u_{\texttt{max}}$ **then**
17 **if** $\frac{\xi(v_{i,j},p_x) - u_{\texttt{max}}}{\xi(v_{i,j},p_x)} \leq \tau$ *and* $ACC(p_{\texttt{pre}}, p_x) > ACC(p_{\texttt{pre}}, p_y)$ **then**
18 continue;
19 $p_y \leftarrow p_x$ and $u_{\texttt{max}} \leftarrow \xi(v_{i,j}, p_x)$;
20 **else if** $\frac{u_{\texttt{max}} - \xi(v_{i,j},p_x)}{u_{\texttt{max}}} \leq \tau$ *and* $ACC(p_{\texttt{pre}}, p_x) < ACC(p_{\texttt{pre}}, p_y)$ **then**
21 $p_y \leftarrow p_x$ and $u_{\texttt{max}} \leftarrow \xi(v_{i,j}, p_x)$;
22 $z^y_{i,j} \leftarrow 1$, $u^{\mathrm{R}}_y \leftarrow u^{\mathrm{R}}_y + \gamma^{\mathrm{R}}_{i,j}$, $\forall \mathrm{R} \in \{\mathtt{C}, \mathtt{M}\}$;
23 **return** p_y;

C. SVD Algorithm

For each VNF $v_{i,j}$ of SFC s_i, where $j > 1$, we employ the SVD algorithm to find a PM to deploy it. As mentioned earlier, the deployment of $v_{i,j}$ is affected by its previous VNF $v_{i,j-1}$ in $\hat{V}_i$. To reduce the RCC taken to complete s_i, we have to limit the ACC between the two PMs, to which $v_{i,j-1}$ and $v_{i,j}$ are assigned, not exceeding threshold $\sigma_{\texttt{dep}}$.

Algorithm 2 presents SVD's pseudocode. Line 1 initializes four variables: 1) $f_{\texttt{idl}}$ is a boolean variable to check if there exist idle PMs; 2) $\hat{P}_{\texttt{can}}$ is the set of candidate PMs; 3) $u_{\texttt{max}}$ is the largest utility value; and 4) p_y is the PM to deploy $v_{i,j}$. The for-loop in lines 2–4 finds out the PM where VNF $v_{i,j-1}$ is deployed, as stored in variable $p_{\texttt{pre}}$. Then, we search for candidate PMs from $\hat{D}_{\texttt{pre}}$ (i.e., the set of PMs whose ACCs to PM $p_{\texttt{pre}}$ are no larger than threshold $\sigma_{\texttt{dep}}$). The code is given in lines 5–12. Specifically, if a PM p_x has sufficient resources to handle VNF $v_{i,j}$ (i.e., $\Gamma^{\mathrm{R}}_x - u^{\mathrm{R}}_x \geq \gamma^{\mathrm{R}}_{i,j}$, $\forall \mathrm{R} \in \{\mathtt{C}, \mathtt{M}\}$ in line 6), it is a candidate PM. Due to the code in lines 7–10, if there exist idle PMs in $\hat{D}_{\texttt{pre}}$, the candidate set $\hat{P}_{\texttt{can}}$ contains only idle PMs. Otherwise, all PMs in $\hat{D}_{\texttt{pre}}$ whose resources are enough (and they must be non-idle) will be included in $\hat{P}_{\texttt{can}}$, as shown in lines 11–12. In other words, we tend to deploy VNF $v_{i,j}$ on an idle PM first. However,

if no candidate PM is found, SVD returns a null value, as shown in lines 13–14.

The residual code selects a PM from the candidate set $\hat{P}_{\texttt{can}}$ to handle VNF $v_{i,j}$. The for-loop in lines 15–21 checks each PM in $\hat{P}_{\texttt{can}}$ and then chooses PM p_y that has the largest utility value (as stored in $u_{\texttt{max}}$). However, instead of simply finding the PM whose utility value is the maximum, we additionally consider two cases:

Case 1. (lines 16–19): PM p_x has a larger utility value than $u_{\texttt{max}}$ (i.e., p_y's utility value), as shown in line 16. In general, we can replace p_y by p_x and update $u_{\texttt{max}}$ to $\xi(v_{i,j}, p_x)$ (i.e., line 19). However, a special situation shall be excluded: p_x's utility value is close to $u_{\texttt{max}}$, and p_x has a larger ACC to $p_{\texttt{pre}}$ than p_y does. In this situation, compared with p_y, choosing p_x may not bring benefits to the deployment of subsequent VNFs, but it increases SFC s_i's RCC (i.e., raising the cost). That is why we exclude the situation by lines 17–18. Here, p_x's utility value is said to be close to $u_{\texttt{max}}$ if $\frac{\xi(v_{i,j},p_x)-u_{\texttt{max}}}{\xi(v_{i,j},p_x)} \leq \tau$, where τ is a small positive value (e.g., $0 < \tau < 0.1$). Furthermore, $ACC(p_{\texttt{pre}}, \text{null})$ is set to infinity.

Case 2. (lines 20–21): The utility value of PM p_x is slightly below $u_{\texttt{max}}$, and p_x has a smaller ACC to $p_{\texttt{pre}}$ than p_y does. Compared to p_y, selecting p_x may not have much impact on the deployment of subsequent VNFs. However, doing so saves s_i's RCC. As a result, we prefer replacing p_y by p_x.

Afterward, line 22 marks that VNF $v_{i,j}$ is assigned to PM p_y (i.e., $z^y_{i,j} \leftarrow 1$) and updates p_y's consumed resources (i.e., $u^{\texttt{R}}_y \leftarrow u^{\texttt{R}}_y + \gamma^{\texttt{R}}_{i,j}$, $\forall \texttt{R} \in \{\texttt{C}, \texttt{M}\}$).

Theorem 2. *Suppose that there are ζ_P PMs in $\hat{P}$, and for each PM $p_x \in \hat{P}$, $\hat{D}_x$ has no more than ζ_D PMs. Then, the SVD algorithm requires $O(\zeta_P + \zeta_D^2)$ time in the worst case.*

Proof. In Algorithm 2, the first for-loop in lines 2–4 searches for the PM on which VNF $v_{i,j-1}$ is deployed from $\hat{P}$, which spends $O(\zeta_P)$ time. The second for-loop in lines 5–12 checks if each PM in $\hat{D}_{\texttt{pre}}$ is a candidate. This evidently takes $O(\zeta_D)$ time. The third for-loop in lines 15–21 checks each candidate PM in $\hat{P}_{\texttt{can}}$. Since $\hat{P}_{\texttt{can}} \subseteq \hat{D}_{\texttt{pre}}$, at most ζ_D PMs are checked. Using Eq. (10) to calculate $\xi(v_{i,j}, p_x)$ will require $O(\zeta_D + 1)$ time. Hence, the third for-loop spends time of $\zeta_D \times O(\zeta_D + 1) = O(\zeta_D^2)$. Since the remaining code consumes $O(1)$ time, the total time complexity is $O(\zeta_P)+O(\zeta_D)+O(\zeta_D^2)+O(1) = O(\zeta_P + \zeta_D^2)$. □

D. AVO Algorithm

Once a PM p_o is overloaded, its VNFs will be offloaded to nearby, non-overloaded PMs. To do so, we define a threshold $\sigma_{\texttt{mig}}$ ($\sigma_{\texttt{mig}} > \sigma_{\texttt{dep}}$) and denote by $\hat{M}_o$ the set of PMs whose ACCs to p_o do not overtake $\sigma_{\texttt{mig}}$. Because $\sigma_{\texttt{mig}} > \sigma_{\texttt{dep}}$, the condition of $\hat{D}_o \subset \hat{M}_o$ must hold. Fig. 1 gives an example. Then, we select a target PM from $\hat{M}_o$ for offloading one of p_o's VNFs. This is carried out by the AVO algorithm, whose pseudocode is given in Algorithm 3.

Suppose that PM p_o is overloaded due to insufficient type-R resources (i.e., $u^{\texttt{R}}_o > \delta_{\texttt{R}} \times \Gamma^{\texttt{R}}_o$, for $\texttt{R} = \texttt{C}$ or $\texttt{M}$). Let us denote by $\texttt{R}'$ the other type of resources. In line 1, we initialize the six variables: 1) $\hat{P}_{\texttt{can}}$ contains candidate PMs, 2) $\lambda_{\texttt{R}}$ gives the maximum number of affordable type-R resources among PMs in $\hat{M}_o$, 3) $\lambda_{\texttt{R}'}$ is the number of affordable type-$\texttt{R}'$ resources

Algorithm 3: AVO algorithm

Data: p_o: the overloaded PM (due to type-R resources)
Result: 1) p_t: the PM chosen to share p_o's load, and 2) $v_{\texttt{mig}}$: p_o's VNF selected to be moved to p_t

1 $\hat{P}_{\texttt{can}} \leftarrow \emptyset$, $\lambda_{\texttt{R}} \leftarrow 0$, $\lambda_{\texttt{R}'} \leftarrow 0$, $v_{\texttt{mig}} \leftarrow$ null, $p_t \leftarrow$ null, $c_{\texttt{min}} \leftarrow \infty$;
2 **for** $p_x \in \hat{M}_o$ **do**
3 **if** $\delta_R \times \Gamma^R_x - u^R_x > 0$ *and* $\delta_{R'} \times \Gamma^{R'}_x - u^{R'}_x > 0$ **then**
4 $\hat{P}_{\texttt{can}} \leftarrow \hat{P}_{\texttt{can}} \cup \{p_x\}$;
5 **if** $\delta_R \times \Gamma^R_x - u^R_x > \lambda_R$ **then**
6 $\lambda_{\texttt{R}} \leftarrow \delta_{\texttt{R}} \times \Gamma^{\texttt{R}}_x - u^{\texttt{R}}_x$, $\lambda_{\texttt{R}'} \leftarrow \delta_{\texttt{R}'} \times \Gamma^{\texttt{R}'}_x - u^{\texttt{R}'}_x$;
7 **if** $\hat{P}_{\texttt{can}} = \emptyset$ **then**
8 **return** null;
9 Sort(Λ_o, $\gamma^{\texttt{R}}_{i,j}$);
10 **for** $v_{i,j} \in \Lambda_o$ **do**
11 **if** $\lambda_R \geq \gamma^R_{i,j}$ *and* $\lambda_{R'} \geq \gamma^{R'}_{i,j}$ **then**
12 $v_{\texttt{mig}} \leftarrow v_{i,j}$ and break;
13 **if** $v_{\texttt{mig}} =$ *null* **then**
14 **return** null;
15 **for** $p_x \in \hat{P}_{\texttt{can}}$ **do**
16 **if** $\delta_R \times \Gamma^R_x - u^R_x \geq \gamma^R_{\texttt{mig}}$ *and* $\delta_{R'} \times \Gamma^{R'}_x - u^{R'}_x \geq \gamma^{R'}_{\texttt{mig}}$ **then**
17 **if** $p_t =$ *null or* $ACC(p_o, p_x) < c_{\texttt{min}}$ **then**
18 $p_t \leftarrow p_x$, $c_{\texttt{min}} \leftarrow ACC(p_o, p_x)$;
19 $z^o_{\texttt{mig}} \leftarrow 0$, $u^{\texttt{R}}_o \leftarrow u^{\texttt{R}}_o - \gamma^{\texttt{R}}_{\texttt{mig}}$, $u^{\texttt{R}'}_o \leftarrow u^{\texttt{R}'}_o - \gamma^{\texttt{R}'}_{\texttt{mig}}$;
20 $z^t_{\texttt{mig}} \leftarrow 1$, $u^{\texttt{R}}_t \leftarrow u^{\texttt{R}}_t + \gamma^{\texttt{R}}_{\texttt{mig}}$, $u^{\texttt{R}'}_t \leftarrow u^{\texttt{R}'}_t + \gamma^{\texttt{R}'}_{\texttt{mig}}$;
21 **return** p_t and $v_{\texttt{mig}}$;

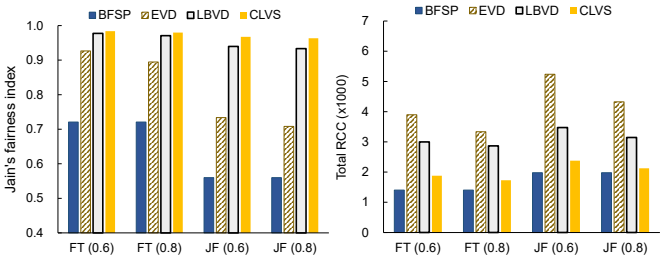
owned by the PM whose number of type-R resources is $\lambda_{\texttt{R}}$, 4) $v_{\texttt{mig}}$ is p_o's VNF to be offloaded, 5) p_t is the target PM, and 6) $c_{\texttt{min}}$ is the minimum ACC between p_o and another PM. The for-loop in lines 2–6 finds non-overloaded PMs from $\hat{M}_o$ (i.e., $\delta_{\texttt{R}} \times \Gamma^{\texttt{R}}_x - u^{\texttt{R}}_x > 0$ and $\delta_{\texttt{R}'} \times \Gamma^{\texttt{R}'}_x - u^{\texttt{R}'}_x > 0$, as line 3 shows) and adds them to $\hat{P}_{\texttt{can}}$. Then, the code in lines 5–6 is used to calculate $\lambda_{\texttt{R}}$ and $\lambda_{\texttt{R}'}$. However, if no candidate PM can be found, AVO returns a null value, as shown in 7–8.

Let Λ_o signify the set of VNFs assigned to PM p_o. Line 9 sorts VNFs in Λ_o decreasingly based on the number of type-R resources required by them (i.e., $\gamma^{\texttt{R}}_{i,j}$). Then, the for-loop in lines 10–12 searches for the VNF from Λ_o (i.e., $v_{\texttt{mig}}$) with the maximum $\gamma^{\texttt{R}}_{i,j}$ value based on two conditions: $\lambda_{\texttt{R}} \geq \gamma^{\texttt{R}}_{i,j}$ and $\lambda_{\texttt{R}'} \geq \gamma^{\texttt{R}'}_{i,j}$. These two conditions are used to make sure that there exist PMs in $\hat{M}_o$ capable of accommodating $v_{\texttt{mig}}$. However, if no such VNF can be found, the AVO algorithm terminates, as shown in lines 13–14.

The for-loop in lines 15–18 checks each candidate PM p_x in $\hat{P}_{\texttt{can}}$. When p_x has affordable resources to cope with VNF $v_{\texttt{mig}}$ (i.e., line 16) and the ACC between p_o and p_x is smaller than $c_{\texttt{min}}$ (i.e., line 17), we replace p_t with p_x and update $c_{\texttt{min}}$ to $ACC(p_o, p_x)$. In other words, among those PMs in $\hat{P}_{\texttt{can}}$ that will not become overloaded after serving $v_{\texttt{mig}}$, we select

TABLE III
SIMULATION PARAMETERS.

parameter	value
the number of PMs	99 (each PM has at most 6 links)
PM's resource capacity	computing ($\Gamma_x^{\mathtt{C}}$): 32 units memory ($\Gamma_x^{\mathtt{M}}$): 64 units
link's communication cost	minimum ($c_{\mathtt{min}}$): 1 unit maximum ($c_{\mathtt{max}}$): 20 units
the number of SFCs	25 (each SFC contains 10 to 12 VNFs)
the number of total VNFs	275
VNF's resource demands	GRD: $\gamma_{i,j}^{\mathtt{C}} \in [5,7]$, $\gamma_{i,j}^{\mathtt{M}} \in [9,11]$ HCD: $\gamma_{i,j}^{\mathtt{C}} \in [6,8]$, $\gamma_{i,j}^{\mathtt{M}} \in [9,11]$ HMD: $\gamma_{i,j}^{\mathtt{C}} \in [5,7]$, $\gamma_{i,j}^{\mathtt{M}} \in [12,14]$
warning thresholds	$\delta_{\mathtt{R}} = 0.6$ and 0.8, $\forall \mathtt{R} \in \{\mathtt{C},\mathtt{M}\}$



(a) fairness (b) total RCC

Fig. 2. Performance comparison in the GRD scenario (FT: fat-tree topology, JF: jellyfish topology).

the PM that has the smallest ACC to p_o. Doing so can reduce the RCC of $v_{\mathtt{mig}}$'s SFC. Then, lines 19–20 indicate that $v_{\mathtt{mig}}$ is moved from p_o to p_t (i.e., $z_{\mathtt{mig}}^o \leftarrow 0$ and $z_{\mathtt{mig}}^t \leftarrow 1$) and update the consumed numbers of resources of these two PMs.

Theorem 3. *Consider that $\hat{M}_x$ has no more than ζ_M PMs for any PM p_x in $\hat{P}$. The AVO algorithm has a time complexity of $O(\zeta_M + \zeta_h \log_2 \zeta_h)$, where ζ_h is the maximum number of VNFs that a PM can handle.*

Proof. Based on Algorithm 3, the first for-loop in lines 2–6 checks each PM in $\hat{M}_o$ once, which requires $O(\zeta_M)$ time. In line 9, sorting the VNFs in Λ_o (i.e., the set of VNFs deployed on PM p_o) spends time of $O(\zeta_h \log_2 \zeta_h)$. Then, the second for-loop in lines 10–12 needs $O(\zeta_h)$ time. The third for-loop in lines 15–18 checks each PM in $\hat{P}_{\mathtt{can}}$. As $\hat{P}_{\mathtt{can}} \subseteq \hat{M}_o$ holds, this for-loop uses $O(\zeta_M)$ time. The residual code takes $O(1)$ time. Hence, the time complexity is $O(\zeta_M)+O(\zeta_h \log_2 \zeta_h)+O(\zeta_h)+O(\zeta_M)+O(1) = O(\zeta_M + \zeta_h \log_2 \zeta_h)$. □

V. PERFORMANCE EVALUATION

For performance evaluation, we build a simulation in C++. It adopts two popular topologies for large networks (e.g., data center networks), namely *fat tree* and *jellyfish*. In a fat tree, PMs are well organized into a three-layer tree structure. There exist multiple shortest paths between any two edge PMs [22]. The jellyfish topology uses a random regular graph. Non-edge PMs possess the same degree (i.e., the number of links for a PM to its adjacent PMs), but they are arbitrarily connected [23]. The NFV system contains 99 PMs, of which there are 54 edge PMs (i.e., $|\hat{P}| = 99$ and $|\hat{P}_{\mathtt{edg}}| = 54$). Using fat-tree and jellyfish topologies, there are 162 and 137 links in total.

Table III lists parameters used in the simulation. Each PM in $\hat{P}$ has a capacity of 32 units of computing resources and 64 units of memory resources. The communication cost of a link is within $[1, 20]$ units. There are 25 SFCs in $\hat{S}$, where each SFC has 10 to 12 VNFs. The number of total VNFs is set to 275. To observe the effect of VNF resource demands, we design three scenarios:

1) *General resource demand (GRD)*: Each VNF needs $[5, 7]$ and $[9, 11]$ units of computing and memory resources.
2) *High computing resource demand (HCD)*: A VNF requires $[6, 8]$ and $[9, 11]$ units of computing and memory resources.
3) *High memory resource demand (HMD)*: A VNF asks for $[5, 7]$ and $[12, 14]$ units of computing and memory resources.

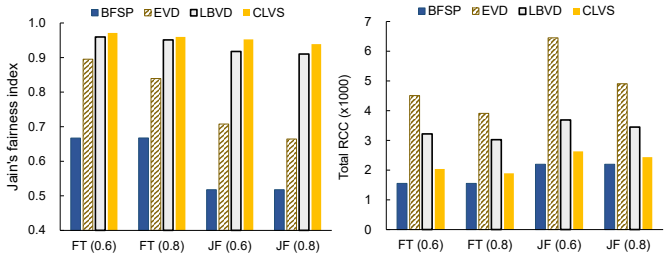
Three methods are selected for comparison with our CLVS scheme. The *best-fit SFC placement (BFSP)* method [9] finds a PM whose resources are best fit to the demand of an SFC. In the *efficient VNF deployment (EVD)* method [18], if PMs are overloaded, some of their VNFs are moved to light-load PMs for load sharing. The *load-balanced VNF deployment (LBVD)* method [19] scores each PM p_x according to the expected ratios of remaining resources of p_x as well as p_x's one-hop neighbors. Then, LBVD deploys VNFs on high-score PMs. For an overloaded PM, LBVD chooses the VNF consuming the most resources to migrate to another PM. EVD, LBVD, and CLVS employ warning thresholds $\delta_{\mathtt{R}}$ ($\mathtt{R} \in \{\mathtt{C},\mathtt{M}\}$) to check if a PM is overloaded, where $\delta_{\mathtt{R}}$ is set to 0.6 and 0.8. In CLVS, we set $\sigma_{\mathtt{dep}} = 20$, $\sigma_{\mathtt{mig}} = 40$, and $\tau = 0.075$.

Next, we compare the performance of BFSP, EVD, LBVD, and CLVS methods in the GRD, HCD, and HMD scenarios, as discussed in Sections V-A, V-B, and V-C, respectively. Recall that the VNF deployment problem has three objectives: 1) increasing the service ratio, 2) balancing loads between PMs (i.e., maximizing their fairness), and 3) minimizing the total RCC of SFCs. As the number of resources of PMs is more than that requested by all VNFs, the service ratio using each method can achieve 100%. Consequently, we focus on measuring the fairness and RCC.

A. GRD Scenario

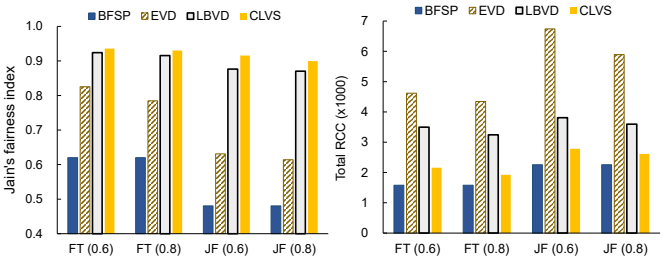
Fig. 2(a) presents the Jain's fairness index of each method in the GRD scenario. The numbers in parentheses indicate the values of warning thresholds $\delta_{\mathtt{R}}$, for $\mathtt{R} \in \{\mathtt{C},\mathtt{M}\}$. Since the fat-tree topology contains more links, it increases the selectivity of PMs when deploying VNFs. Moreover, this topology has a well-organized structure. As a result, all methods have higher index values (i.e., better performance) in the fat-tree topology. Because BFSP does not employ warning thresholds, its index values will not change with different $\delta_{\mathtt{R}}$ values. As for other methods, each PM is capable of accommodating more VNFs when $\delta_{\mathtt{R}}$ is larger. Doing so makes a few VNFs gather in some PMs, causing a slight load imbalance among PMs. That is why their index values slightly reduce by increasing $\delta_{\mathtt{R}}$.

As can be seen, BFSP results in the lowest fairness, since it considers deploying the VNFs of the same SFC on a single PM. EVD moves some VNFs of high-load PMs to light-load PMs, thereby increasing index values. Both LBVD and CLVS deploy VNFs on PMs that have relatively abundant resources, and they differentiate the deployment between leading VNFs and other VNFs. Doing so significantly improves the fairness, especially in the jellyfish topology. Specifically, LBVD and CLVS keep index values above 0.93 and 0.96, while EVD's index values drop below 0.73 in the jellyfish topology.



(a) fairness (b) total RCC

Fig. 3. Performance comparison in the HCD scenario (FT: fat-tree topology, JF: jellyfish topology).



(a) fairness (b) total RCC

Fig. 4. Performance comparison in the HMD scenario (FT: fat-tree topology, JF: jellyfish topology).

Fig. 2(b) displays the total RCC of every method in the GRD scenario. Since the jellyfish topology employs a random regular graph and possesses fewer links, the VNFs of an SFC may be deployed on some PMs that have higher ACCs with each other (due to fewer choices of links). That explains why each method has a higher total RCC in the jellyfish topology. As mentioned earlier, a larger δ_{R} value may make more VNFs be deployed on some PMs, which in turn helps reduce RCCs of their SFCs. Therefore, increasing δ_{R} values could result in lower total RCCs in EVD, LBVD, and CLVS.

Then, we analyze the performance of each method on the total RCC. For each SFC, BFSP picks a PM whose resources best fit its demand. This may decrease the probability that the SFC's VNFs are deployed on many PMs. Hence, BFSP has the lowest total RCC. EVD has the highest total RCC since it does not take account of ACCs between PMs when deploying and moving VNFs. Inevitably, some VNFs of the same SFC may be deployed on those PMs far away from each other. To mitigate this problem, LBVD limits hop counts between PMs for VNF deployment, so it has a lower total RCC than EVD. However, LBVD does not consider the communication cost of each link. In light of this, our CLVS scheme uses two sets, $\hat{D}_x$ and $\hat{M}_x$, which contain PMs whose ACCs to a PM p_x do not exceed thresholds $\sigma_{\mathtt{dep}}$ and $\sigma_{\mathtt{mig}}$ for VNF deployment and offloading. Moreover, CLVS prefers choosing edge PMs to deploy leading VNFs to reduce RCCs of their SFCs. Thus, CLVS can greatly reduce the total RCC compared to LBVD. When we use BFSP's total RCC as a baseline, EVD, LBVD, and CLVS increase to 148.35%, 84.63%, and 19.93% of the total RCC. This result, together with the result in Fig. 2(a), demonstrates that our CLVS scheme effectively achieves load balance among PMs without significantly increasing the total RCC in the GRD scenario.

B. HCD Scenario

In the HCD scenario, VNFs use more computing resources. Specifically, 275 VNFs totally ask for 1673 and 1874 units of computing resources in the GRD and HCD scenarios, so each VNF needs about 6.08 and 6.81 units of computing resources, respectively. From the viewpoint of computing resources, a PM can serve $32/6.08$ (≈ 5.26) VNFs in the GRD scenario but only $32/6.81$ (≈ 4.70) VNFs in the HCD scenario.

Let us compare Figs. 2 and 3, which present the results in the GRD and HCD scenarios. The performance trends of each method in both scenarios are similar. However, VNFs ask for more computing resources in the HCD scenario, but their demands for memory resources remain similar to those in the GRD scenario. Some PMs may not accommodate VNFs due to insufficient computing resources, even if they still retain memory resources. Compared to the GRD scenario, the Jain's fairness index of each method decreases in the HCD scenario. More concretely, the index values of BFSP, EVD, LBVD, and CLVS are reduced by 7.49%, 4.80%, 2.17%, and 1.84% in the HCD scenario than in the GRD scenario. As the number of VNFs that can be handled by a PM decreases in the HCD scenario, the VNFs of the same SFC could be assigned to more PMs. That is why all methods have higher total RCCs in the HCD scenario. Compared to the GRD scenario, the total RCCs of BFSP, EVD, LBVD, and CLVS are increased by 10.94%, 17.73%, 7.13%, and 10.98%.

As can be seen from Fig. 3, compared with BFSP, EVD, and LBVD, CLVS can improve the Jain's fairness index by 61.31%, 23.04%, and 2.25%. Moreover, taking BFSP's total RCC as a baseline, EVD, LBVD, and CLVS raise 163.54%, 78.29%, and 19.97% of the total RCC. This result verifies that our CLVS scheme can balance PMs' loads while reducing the total RCC of all SFCs in the HCD scenario.

C. HMD Scenario

In the HMD scenario, VNFs need more memory resources. The overall demand for memory resources of 275 VNFs is 2771 and 3593 units, so each VNF will request around 10.08 and 13.07 units of memory resources in the GRD and HMD scenarios. From the viewpoint of memory resources, a PM can handle $64/10.08$ (≈ 6.35) VNFs in the GRD scenario but just $64/13.07$ (≈ 4.90) VNFs in the HMD scenario.

Let us compare Figs. 2 and 4, which display the results in the GRD and HMD scenarios. Evidently, there is not much difference between the performance trends of all methods in both scenarios. However, since VNFs require more memory resources in the HMD scenario, some PMs will have more computing resources left after using up memory resources. Compared to the GRD scenario, the values of Jain's fairness index in BFSP, EVD, LBVD, and CLVS are cut down by 14.08%, 12.51%, 6.15%, and 5.48% in the HMD scenario. The total RCCs of BFSP, EVD, LBVD, and CLVS are raised by 13.43%, 28.61%, 13.26%, and 16.85%.

According to the result in Fig. 4, as compared with BFSP, EVD, and LBVD, CLVS improves the Jain's fairness index by 67.24%, 28.92%, and 2.64%. In addition, as we employ BFSP's total RCC to be a baseline, EVD, LBVD, and CLVS enlarge 181.60%, 84.36%, and 23.54% of the total RCC. This result shows that our CLVS scheme can efficiently achieve load balance between PMs in the HMD scenario. Moreover,

CLVS can reduce the total RCC as compared with both EVD and LBVD methods.

VI. Conclusion and Future Work

The NFV technique abstracts network functions as VNFs for facilitating resource management. The assignment of PMs to execute each SFC's VNFs has a great influence on NFV's performance. This article proposes the CLVS scheme with the objectives of maximizing the service ratio, balancing loads between PMs, and saving the total RCC of all SFCs. CLVS assesses the suitability of deploying a VNF on each PM by using a utility function. It deploys the leading VNF and other VNFs of each SFC in different ways. If a PM is overloaded, CLVS picks out its VNFs to be offloaded to nearby PMs. Our simulation takes account of fat-tree and jellyfish topologies as well as three scenarios with different resource demands of VNFs. Simulation results display that on the premise of achieving 100% of the service ratio, CLVS greatly improves the Jain's fairness index as compared to the BFSP, EVD, and LBVD methods. In addition, CLVS can efficiently decrease the total RCC of all SFCs compared to both EVD and LBVD.

In the CLVS scheme, a PM provides exactly the required number of resources for each VNF. As future work, we will consider that resources allocated to VNFs can be dynamically adjusted [24]. For example, when a PM has more residual resources, the PM can strategically allocate more resources to its VNFs, thereby accelerating the execution of these VNFs. Moreover, it is interesting to take node heterogeneity of PMs into account. Specifically, some PMs only have CPUs as their computing resources, while others have both CPU and GPU computing resources. In this case, how to achieve the Pareto optimality [25] of VNF deployment deserves further investigation. Finally, in the upcoming 6G, a network may include aerial (e.g., LAPS/HAPS), satellite (e.g., LEO/MEO/GEO), and edge nodes [26]. Since link costs vary dynamically due to topology changes, the issue of VNF deployment becomes more challenging.

Acknowledgment

This work was supported by National Science and Technology Council, Taiwan (Grant: 113-2221-E-110-056-MY3).

References

[1] T. Zhang, H. Qiu, L. Linguaglossa, W. Cerroni, and P. Giaccone, "NFV platforms: Taxonomy, design choices and future challenges," *IEEE Transactions on Network and Service Management*, vol. 18, no. 1, pp. 30–48, 2021.

[2] W. Rafique, J. R. Barai, A. O. Fapojuwo, and D. Krishnamurthy, "A survey on beyond 5G network slicing for smart cities applications," *IEEE Communications Surveys & Tutorials*, vol. 27, no. 1, pp. 595–628, 2025.

[3] W. Attaoui, E. Sabir, H. Elbiaze, and M. Guizani, "VNF and CNF placement in 5G: Recent advances and future trends," *IEEE Transactions on Network and Service Management*, vol. 20, no. 4, pp. 4698–4733, 2023.

[4] A. D. Domenico, Y. F. Liu, and W. Yu, "Optimal virtual network function deployment for 5G network slicing in a hybrid cloud infrastructure," *IEEE Transactions on Wireless Communications*, vol. 19, no. 12, pp. 7942–7956, 2020.

[5] Y. Wang, C. K. Huang, S. H. Shen, and G. M. Chiu, "Adaptive placement and routing for service function chains with service deadlines," *IEEE Transactions on Network and Service Management*, vol. 18, no. 3, pp. 3021–3036, 2021.

[6] X. Li, H. Zhou, L. Ma, J. Xin, and S. Huang, "Cost and latency customized SFC deployment in hybrid VNF and PNF environment," *IEEE Transactions on Network and Service Management*, vol. 21, no. 4, pp. 4312–4331, 2024.

[7] M. B. Taha, Y. Sanjalawe, A. Al-Daraiseh, S. Fraihat, and S. R. Al-E'mari, "Proactive auto-scaling for service function chains in cloud computing based on deep learning," *IEEE Access*, vol. 12, pp. 38 575–38 593, 2024.

[8] O. Soualah, M. Mechtri, C. Ghribi, and D. Zeghlache, "Online and batch algorithms for VNFs placement and chaining," *Computer Networks*, vol. 158, pp. 98–113, 2019.

[9] C. Pham, N. H. Tran, S. Ren, W. Saad, and C. S. Hong, "Traffic-aware and energy-efficient VNF placement for service chaining: Joint sampling and matching approach," *IEEE Transactions on Services Computing*, vol. 13, no. 1, pp. 172–185, 2020.

[10] S. Qi, S. Li, S. Lin, M. Y. Saidi, and K. Chen, "Energy-efficient VNF deployment for graph-structured SFC based on graph neural network and constrained deep reinforcement learning," in *Asia-Pacific Network Operations and Management Symposium*, 2021, pp. 348–353.

[11] L. Liu, S. Guo, G. Liu, and Y. Yang, "Joint dynamical VNF placement and SFC routing in NFV-enabled SDNs," *IEEE Transactions on Network and Service Management*, vol. 18, no. 4, pp. 4263–4276, 2021.

[12] S. Yang, F. Li, R. Yahyapour, and X. Fu, "Delay-sensitive and availability-aware virtual network function scheduling for NFV," *IEEE Transactions on Services Computing*, vol. 15, no. 1, pp. 188–201, 2022.

[13] D. H. P. Nguyen, Y. H. Lien, B. H. Liu, S. I. Chu, and T. N. Nguyen, "Virtual network function placement for serving weighted services in NFV-enabled networks," *IEEE Systems Journal*, vol. 17, no. 4, pp. 5648–5659, 2023.

[14] M. Avgeris, A. Leivadeas, I. Lambadaris, J. Chinneck, T. Morris, and P. Djukic, "Service function chain network planning through offline, online and infeasibility restoration techniques," *Computer Networks*, vol. 242, pp. 1–17, 2024.

[15] N. T. Jahromi, S. Kianpisheh, and R. H. Glitho, "Online VNF placement and chaining for value-added services in content delivery networks," in *IEEE International Symposium on Local and Metropolitan Area Networks*, 2018, pp. 19–24.

[16] Q. Zhang, F. Liu, and C. Zeng, "Online adaptive interference-aware VNF deployment and migration for 5G network slice," *IEEE/ACM Transactions on Networking*, vol. 29, no. 5, pp. 2115–2128, 2021.

[17] M. A. Abdelaal, G. A. Ebrahim, and W. R. Anis, "High availability deployment of virtual network function forwarding graph in cloud computing environments," *IEEE Access*, vol. 9, pp. 53 861–53 884, 2021.

[18] J. Fu and G. Li, "An efficient VNF deployment scheme for cloud networks," in *IEEE International Conference on Communication Software and Networks*, 2019, pp. 497–502.

[19] Y. C. Wang and S. H. Wu, "Efficient deployment of virtual network functions to achieve load balance in cloud networks," in *Asia-Pacific Network Operations and Management Symposium*, 2022, pp. 1–6.

[20] A. Laghrissi and T. Taleb, "A survey on the placement of virtual resources and virtual network functions," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 2, pp. 1409–1434, 2019.

[21] Y. C. Wang and D. R. Jhong, "Efficient allocation of LTE downlink spectral resource to improve fairness and throughput," *International Journal of Communication Systems*, vol. 30, no. 14, pp. 1–13, 2017.

[22] Y. C. Wang and S. Y. You, "An efficient route management framework for load balance and overhead reduction in SDN-based data center networks," *IEEE Transactions on Network and Service Management*, vol. 15, no. 4, pp. 1422–1434, 2018.

[23] Z. Alzaid, S. Bhowmik, and X. Yuan, "Multi-path routing in the jellyfish network," in *IEEE International Parallel and Distributed Processing Symposium*, 2021, pp. 832–841.

[24] W. K. Lai, Y. C. Wang, and S. C. Wei, "Delay-aware container scheduling in Kubernetes," *IEEE Internet of Things Journal*, vol. 10, no. 13, pp. 11 813–11 824, 2023.

[25] Y. C. Wang, "A two-phase dispatch heuristic to schedule the movement of multi-attribute mobile sensors in a hybrid wireless sensor network," *IEEE Transactions on Mobile Computing*, vol. 13, no. 4, pp. 709–722, 2014.

[26] S. Mahboob and L. Liu, "Revolutionizing future connectivity: A contemporary survey on AI-empowered satellite-based non-terrestrial networks in 6G," *IEEE Communications Surveys & Tutorials*, vol. 26, no. 2, pp. 1279–1321, 2024.