Efficient Deployment of Virtual Network Functions to Achieve Load Balance in Cloud Networks

You-Chiun Wang and Shang-Hao Wu
Department of Computer Science and Engineering,
National Sun Yat-sen University, Kaohsiung, 80424, Taiwan
Email: ycwang@cse.nsysu.edu.tw; rifjvn0717@gmail.com

***Abstract*—Network function virtualization (NFV) is a technique to facilitate service deployment by decoupling network functions from dedicated hardware and moving them to software, namely *virtual network functions (VNFs)*. Network services are carried out by *service function chains (SFCs)* comprising multiple VNFs. Given SFCs and *physical machines (PMs)*, the *VNF deployment problem* asks how to assign each SFC's VNFs to PMs, such that the service ratio is maximized. Many methods use as few PMs as possible for deploying VNFs, but they lead to imbalanced loads of PMs in a cloud network. This paper proposes a *load-balanced VNF deployment (LBVD)* scheme, which scores each PM based on the expected ratios of residual resources of that PM and its neighbors. Then, VNFs are deployed on PMs with high scores. If a PM is busy, LBVD lets its VNF migrate to another PM for load sharing. Through simulations, we show that LBVD can balance the loads of PMs and save both deployment and migration costs.**

***Index Terms*—deployment, load balance, migration, network function virtualization (NFV), virtual network function (VNF).**

I. Introduction

Traditionally, network functions like firewall and intrusion detection execute on dedicated hardware called *middle-boxes*. Deploying network services involves selecting middle-boxes and steering data to pass them for processing in specific orders, known as *service function chains (SFCs)*. After deployment, it incurs a high cost to adjust SFCs by changing middle-boxes. Some middle-boxes may be even incompatible with each other. This makes resource management inflexible.

The emerging *network function virtualization (NFV)* technique abstracts network functions and makes them be easily installed and manipulated by software modules called *virtual network functions (VNFs)*. Unlike middle-boxes, VNFs can run on general-purpose commodity machines and migrate between machines [1]. Users can also flexibly expand or shrink SFCs by adding or deleting VNFs. Thus, NFV provides high scalability, facilitates resource management, and reduces hardware costs.

How to assign VNFs to *physical machines (PMs)*, namely the *VNF deployment problem*, makes a great impact on NFV's performance. It is modeled as an NP-hard optimization problem [2]. Many deployment methods consider using the fewest PMs to maximize the *service ratio*, as defined by the ratio of the number of VNFs successfully deployed to the number of VNFs requested by SFCs. In this way, the resource utilization of working PMs improves. However, these methods do not let VNFs migrate from heavy-load PMs to light-load PMs. Hence, some PMs are busy and their performance degrades, whereas other PMs stay idle. Actually, load balance is critical in cloud networks, which contain many PMs for load sharing [3].

Thus, this paper proposes a *load-balanced VNF deployment (LBVD)* scheme to maximize the service ratio and balance the loads of PMs. According to the expected ratios of residual resources of a PM and its neighbors, LBVD scores each PM and chooses the PM with the highest score to deploy each VNF. Once a PM is overloaded, LBVD transfers its VNF to another PM for load sharing (i.e., VNF migration). Simulation results show that the LBVD scheme greatly raises the *Jain's fairness index (JFI)* as compared with other deployment methods in different topologies of PMs, which means that it can efficiently achieve load balance in cloud networks. Moreover, LBVD can reduce the deployment and migration costs for VNFs.

II. Related Work

Some studies aim at path selection for SFCs, which decides routes of PMs to run their VNFs. In [4], the selection problem is converted to a problem in the gray system theory and solved by gray relation analysis. Qu et al. [5] develop a hybrid routing method to provide reliability to SFCs, which combines single-path and multi-path routing. Yang et al. [6] propose a path-based *integer linear programming (ILP)* model to save energy of PMs. With the shortest path algorithm, the study [7] assigns VNFs to PMs to reduce their hop counts. Apparently, these studies have different objectives with our work.

How to deploy VNFs to improve PMs' utilization is widely discussed. The work [8] solves the deployment problem by ILP, and limits candidate PMs to deploy VNFs to reduce ILP's complexity. Given the flow demands of SFCs, an optimization problem is formulated in [9] to minimize the usage of network resources. The study [10] first picks the PMs already assigned with VNFs to deploy the VNFs of new SFC requests. In [11], the deep reinforcement learning is applied to VNF deployment, which reduces the number of working PMs. Nevertheless, the above methods make the loads of PMs become imbalanced.

The VNF migration issue also attracts attention. The work [12] deploys VNFs in a content delivery network, where some VNFs can be reused or migrate to other PMs. In [13], VNFs migrate to ensure high availability against PM failures. Zhang et al. [14] address both VNF deployment and migration for 5G network slice, where VNFs migrate to satisfy service-specific demands. In [15], when a PM is overloaded, some of its VNFs will migrate to other PMs. As can be seen, the load-balancing

issue in VNF deployment is not extensively discussed. This motivates us to propose the LBVD scheme to raise the service ratio while keeping PMs' loads as balanced as possible.

III. System Model

Suppose that a cloud network has a set $\hat{\mathcal{P}}$ of PMs. For each PM $p_i \in \hat{\mathcal{P}}$, $\Phi_i^{\mathtt{T}}$ is the capacity of its type-$\mathtt{T}$ resource, where $\Phi_i^{\mathtt{T}} \in \mathbb{Z}^+$. Without loss of generality, we consider CPU and memory resources, so $\mathtt{T} \in \{\mathtt{CPU}, \mathtt{MEM}\}$. Besides, $H(p_i, p_j)$ is the minimum hop count between two PMs p_i and p_j. There is a set $\hat{\mathcal{S}}$ of SFCs to be served. Each SFC $s_k \in \hat{\mathcal{S}}$ comprises a sequence of VNFs $\hat{\mathcal{V}}_k = \{v_{k,1}, v_{k,2}, \cdots, v_{k,m}\}$, where $\phi_{k,j}^{\mathtt{T}}$ is the amount of type-$\mathtt{T}$ resource required by a VNF $v_{k,j}$.

The JFI is used to evaluate the degree of load balance among all PMs in $\hat{\mathcal{P}}$, which is defined as follows [16]:

$$J(\hat{\mathcal{P}}) = \left(\sum\nolimits_{p_i \in \hat{\mathcal{P}}} \zeta_i\right)^2 / \left(|\hat{\mathcal{P}}| \sum\nolimits_{p_i \in \hat{\mathcal{P}}} \zeta_i^2\right). \quad (1)$$

Here, ζ_i is the load of a PM p_i and $\zeta_i = (C_i^{\mathtt{CPU}} + C_i^{\mathtt{MEM}})/2$, where $C_i^{\mathtt{CPU}}$ and $C_i^{\mathtt{MEM}}$ signify the amount of CPU and memory resources consumed in p_i, respectively. By Eq. (1), we have $1/|\hat{\mathcal{P}}| \le J(\hat{\mathcal{P}}) \le 1$. A larger $J(\hat{\mathcal{P}})$ value implies that the loads of PMs are more balanced.

Let us use an indicator $z_{k,j}^i \in \{0, 1\}$ to reveal whether VNF $v_{k,j}$ is deployed on PM p_i, where $z_{k,j}^i = 1$ if so or $z_{k,j}^i = 0$ otherwise. The VNF deployment problem can be expressed by

$$\text{maxmize} \quad \frac{\sum_{p_i \in \hat{\mathcal{P}}} \sum_{s_k \in \hat{\mathcal{S}}} \sum_{v_{k,j} \in \hat{\mathcal{V}}_k} z_{k,j}^i}{\sum_{s_k \in \hat{\mathcal{S}}} |\hat{\mathcal{V}}_k|}, \quad (2)$$

$$\text{maxmize} \quad J(\hat{\mathcal{P}}), \quad (3)$$

subject to

$$\phi_{k,j}^{\mathtt{T}} \le \min\nolimits_{p_i \in \hat{\mathcal{P}}}\{\Phi_i^{\mathtt{T}}\} \qquad \forall v_{k,j} \in \hat{\mathcal{V}}_k, \forall s_k \in \hat{\mathcal{S}}, \quad (4)$$

$$\sum\nolimits_{p_i \in \hat{\mathcal{P}}} z_{k,j}^i \le 1 \qquad \forall v_{k,j} \in \hat{\mathcal{V}}_k, \forall s_k \in \hat{\mathcal{S}}, \quad (5)$$

$$\sum_{s_k \in \hat{\mathcal{S}}} \sum_{v_{k,j} \in \hat{\mathcal{V}}_k} z_{k,j}^i \phi_{k,j}^{\mathtt{T}} \le \Phi_i^{\mathtt{T}} \qquad \forall p_i \in \hat{\mathcal{P}}. \quad (6)$$

The objectives in Eqs. (2) and (3) are to maximize the service ratio and JFI. For constraints, Eq. (4) means that the amount of resources (for CPU and memory) requested by a VNF cannot exceed the capacity of any PM. In Eq. (5), each VNF can be deployed on at most one PM. Since a PM may serve multiple VNFs, Eq. (6) indicates that the overall amount of resources requested by these VNFs cannot overtake the PM's capacity.

Moreover, we define the *deployment cost* for an SFC $s_k \in \hat{\mathcal{S}}$ as the number of hops between PMs that need to be spanned to complete s_k, that is, $\sum_{j=1}^{m-1} H(f_P(v_{k,j}), f_P(v_{k,j+1}))$, where $f_P(v_{k,j})$ is the PM that serves $v_{k,j}$. When all VNFs of s_k are deployed on the same PM, its deployment cost is zero. On the other hand, if a PM $p_i \in \hat{\mathcal{P}}$ becomes busy, some of its VNFs may migrate to other PMs. Then, the *migration cost* for p_i is defined by the total hop counts that its VNFs migrate. Table I summarizes the notations used in the LBVD scheme.

TABLE I
Summary of notations.

notation	definition
$\hat{\mathcal{P}}$, $\hat{\mathcal{S}}$	sets of PMs and SFCs
$\hat{\mathcal{V}}_k$	sequence of VNFs in SFC s_k
$\Phi_i^{\mathtt{T}}$	capacity of type-$\mathtt{T}$ resource of PM p_i, $\mathtt{T} \in \{\mathtt{CPU}, \mathtt{MEM}\}$
$R_i^{\mathtt{T}}$	residual type-$\mathtt{T}$ resource of p_i
$\phi_{k,j}^{\mathtt{T}}$	amount of type-$\mathtt{T}$ resource requested by VNF $v_{k,j}$
$f_P(v_{k,j})$	PM that serves $v_{k,j}$
$H(p_i, p_j)$	hop count between two PMs p_i and p_j
$G(p_i, v_{k,j})$	p_i's score in terms of $v_{k,j}$
$E(p_i, v_{k,j})$	p_i's ratio of residual resources after serving $v_{k,j}$
$B(p_i, v_{k,j})$	function to check if $v_{k,j}$ makes p_i overloaded

IV. The Proposed LBVD Scheme

The LBVD scheme has two modules: *placement* and *migration*. For every SFC $s_k \in \hat{\mathcal{S}}$, we use the placement module to sequentially assign each of its VNFs in $\hat{\mathcal{V}}_k$ to a PM. However, when a VNF $v_{k,j} \in \hat{\mathcal{V}}_k$ cannot be assigned, which implies that none of PMs has enough resources to meet $v_{k,j}$'s requirement, $v_{k,j}$ is removed from $\hat{\mathcal{V}}_k$. Moreover, if the chosen PM will be overloaded after serving $v_{k,j}$, we use the migration module to transfer the PM's VNF to another PM for load balance.

When a PM consumes too many resources, the PM becomes overloaded and its performance may significantly degrade [15]. Specifically, a PM $p_i \in \hat{\mathcal{P}}$ is *overloaded* if $C_i^{\mathtt{CPU}} > \delta_{\mathtt{CPU}} \Phi_i^{\mathtt{CPU}}$ or $C_i^{\mathtt{MEM}} > \delta_{\mathtt{MEM}} \Phi_i^{\mathtt{MEM}}$, where $0 < \delta_{\mathtt{CPU}}, \delta_{\mathtt{MEM}} < 1$ are two given thresholds. To avoid overloading a PM, the placement module decides whether to assign a VNF to the PM based on its score. Next, we explain the scoring mechanism, detail both placement and migration modules, and discuss our LBVD scheme.

A. Scoring Mechanism

The score of a PM p_i in $\hat{\mathcal{P}}$ depends on the VNF $v_{k,j}$ to be deployed. Let $\hat{\mathcal{N}}_i \subset \hat{\mathcal{P}}$ denote the set of p_i's 1-hop neighbors and $v_{k,j+1}$ be the next VNF to be deployed, where $v_{k,j}$ and $v_{k,j+1}$ belong to the same SFC. Then, p_i's score is defined by

$$G(p_i, v_{k,j}) = E(p_i, v_{k,j}) + \sum\nolimits_{p_a \in \hat{\mathcal{N}}_i} E(p_a, v_{k,j+1}), \quad (7)$$

where $E(p_i, v_{k,j})$ is p_i's expected ratio of residual resources after serving $v_{k,j}$, which is estimated by

$$E(p_i, v_{k,j}) = \frac{1}{2} \times \left(\frac{R_i^{\mathtt{CPU}} - \phi_{k,j}^{\mathtt{CPU}}}{\Phi_i^{\mathtt{CPU}}} + \frac{R_i^{\mathtt{MEM}} - \phi_{k,j}^{\mathtt{MEM}}}{\Phi_i^{\mathtt{MEM}}}\right), \quad (8)$$

where $R_i^{\mathtt{CPU}}$ and $R_i^{\mathtt{MEM}}$ denote the amount of residual CPU and memory resources of p_i, respectively (i.e., $R_i^{\mathtt{CPU}} = \Phi_i^{\mathtt{CPU}} - C_i^{\mathtt{CPU}}$ and $R_i^{\mathtt{MEM}} = \Phi_i^{\mathtt{MEM}} - C_i^{\mathtt{MEM}}$).

There are three considerations in Eq. (7). First, we deploy VNFs $v_{k,j}$ and $v_{k,j+1}$ on different (and adjacent) PMs, instead of the same PM (i.e. p_i). In this way, we may avoid imposing a heavy load on p_i. Second, if the current load of p_i is smaller, it has a higher score (as $R_i^{\mathtt{CPU}}$ and $R_i^{\mathtt{MEM}}$ are larger in Eq. (8)). Third, when p_i has more 1-hop neighbors (i.e., $\hat{\mathcal{N}}_i$), there are more candidate PMs for deploying the next VNF (i.e., $v_{k,j+1}$). In this case, p_i could be a good PM for deploying VNF $v_{k,j}$ and has a high score. Notice that if $v_{k,j}$ is the *last* VNF in $\hat{\mathcal{V}}_k$ (i.e., there is no $v_{k,j+1}$), the right term in Eq. (7) is omitted.

Algorithm 1: Placement of the first VNF $v_{k,1}$

1 $\hat{\mathcal{P}}_\alpha \leftarrow \emptyset$, $\hat{\mathcal{P}}_\beta \leftarrow \emptyset$;
2 **foreach** $p_i \in \hat{\mathcal{P}}$ **do**
3 **if** $R_i^{\text{CPU}} = \Phi_i^{\text{CPU}}$ *and* $R_i^{\text{MEM}} = \Phi_i^{\text{MEM}}$ **then**
4 $\hat{\mathcal{P}}_\alpha \leftarrow \hat{\mathcal{P}}_\alpha \cup \{p_i\}$;
5 **else if** $R_i^{\text{CPU}} \geq \phi_{k,1}^{\text{CPU}}$ *and* $R_i^{\text{MEM}} \geq \phi_{k,1}^{\text{MEM}}$ **then**
6 $\hat{\mathcal{P}}_\beta \leftarrow \hat{\mathcal{P}}_\beta \cup \{p_i\}$;
7 **if** $\hat{\mathcal{P}}_\alpha \cup \hat{\mathcal{P}}_\beta = \emptyset$ **then**
8 remove $v_{k,1}$ and set p_x to null;
9 **else if** $\hat{\mathcal{P}}_\alpha \neq \emptyset$ **then**
10 randomly select a PM p_x from $\hat{\mathcal{P}}_\alpha$;
11 **else**
12 $p_x \leftarrow \arg\max_{p_i \in \hat{\mathcal{P}}_\beta} G(p_i, v_{k,1})$;
13 **return** p_x;

Algorithm 2: Placement of a non-first VNF $v_{k,j}$

1 $\hat{\mathcal{P}}_\gamma \leftarrow \emptyset$;
2 **foreach** $p_i \in \hat{\mathcal{P}}$ **do**
3 **if** $H(p_i, f_P(v_{k,j-1})) = 1$ **then**
4 $\hat{\mathcal{P}}_\gamma \leftarrow \hat{\mathcal{P}}_\gamma \cup \{p_i\}$;
5 $p_x \leftarrow \arg\max_{p_i \in \hat{\mathcal{P}}_\gamma} G(p_i, v_{k,j})$;
6 **if** $B(p_x, v_{k,j}) =$ *true* **then**
7 $p_y \leftarrow$ Algo. 3 $(p_x, v_{k,j})$;
8 **if** $p_y \neq$ *null* **then**
9 **return** p_y;
10 **else**
11 **if** $R_x^{\text{CPU}} < \phi_{k,j}^{\text{CPU}}$ *or* $R_x^{\text{MEM}} < \phi_{k,j}^{\text{MEM}}$ **then**
12 remove $v_{k,j}$ and **return** null;
13 **return** p_x;

B. Placement Module

The VNFs of an SFC s_k (i.e., $\hat{\mathcal{V}}_k$) are sequentially deployed, so the placement of every VNF $v_{k,j}$ depends on its previous VNF $v_{k,j-1}$ in $\hat{\mathcal{V}}_k$, except the first VNF $v_{k,1}$. Moreover, $v_{k,1}$'s location may also decide whether other VNFs of $\hat{\mathcal{V}}_k$ can be successfully deployed. Hence, we cope with the first VNF and other VNFs in different ways.

Algo. 1 shows the pseudocode for the placement of the first VNF $v_{k,1}$. Let $\hat{\mathcal{P}}_\alpha$ be the set of *idle* PMs. In particular, if the residual resources of a PM (i.e., R_i^{CPU} and R_i^{MEM}) are equal to its capacity (i.e., Φ_i^{CPU} and Φ_i^{MEM}), the PM is idle. Besides, $\hat{\mathcal{P}}_\beta$ denotes the set of PMs that have enough residual resources to meet the requirement of $v_{k,1}$ (i.e., $\phi_{k,1}^{\text{CPU}}$ and $\phi_{k,1}^{\text{MEM}}$). Lines 1–6 present the code of finding both $\hat{\mathcal{P}}_\alpha$ and $\hat{\mathcal{P}}_\beta$.

Then, we consider three cases. First, $\hat{\mathcal{P}}_\alpha$ and $\hat{\mathcal{P}}_\beta$ are empty (i.e., line 7). In this case, no PM can serve $v_{k,1}$, so we remove $v_{k,1}$ from $\hat{\mathcal{V}}_k$ and return a null value. Notice that the next VNF in $\hat{\mathcal{V}}_k$ will become the (new) first VNF. Second, if $\hat{\mathcal{P}}_\alpha$ is non-empty (i.e., line 9), we choose one idle PM from $\hat{\mathcal{P}}_\alpha$ to serve $v_{k,1}$. Third, there is no idle PM (i.e., line 11). Hence, we pick a PM from $\hat{\mathcal{P}}_\beta$ with the highest score to serve $v_{k,1}$, as shown in line 12. Theorem 1 analyzes the time complexity of Algo. 1.

Theorem 1: Let ξ_P be the number of PMs in $\hat{\mathcal{P}}$. The time complexity of Algo. 1 is $O(\xi_P \xi_N)$, where ξ_N is the maximum number of 1-hop neighbors of each PM in $\hat{\mathcal{P}}$.

Proof: The for-loop in lines 2–6 takes $O(\xi_P)$ time. Lines 8 and 10 spend $O(1)$ time. In line 12, it takes $O(\xi_N+1)$ time to find $G(p_i, v_{k,1})$ for each PM in $\hat{\mathcal{P}}_\beta$ by Eq. (7). Thus, the time complexity is $O(\xi_P) + \max\{O(1), O(1), |\hat{\mathcal{P}}_\beta| O(\xi_N + 1)\} = O(\xi_P + |\hat{\mathcal{P}}_\beta| \xi_N)$. The worst case occurs when $\hat{\mathcal{P}}_\beta = \hat{\mathcal{P}}$, so the complexity can be simplified to $O(\xi_P \xi_N)$. ■

Algo. 2 gives the pseudocode to place a non-first VNF $v_{k,j}$, where $j > 1$. To reduce the deployment cost while balancing the loads of PMs, we place $v_{k,j}$ on a 1-hop neighbor of the PM that serves the previous VNF $v_{k,j-1}$. The code in lines 1–4 finds the candidate PMs for $v_{k,j}$, whose set is denoted by $\hat{\mathcal{P}}_\gamma$. Let $f_P(v_{k,j-1})$ be the PM where $v_{k,j-1}$ is deployed. If the hop count between a PM p_i and $f_P(v_{k,j-1})$ is equal to 1, p_i is added to $\hat{\mathcal{P}}_\gamma$. Among all candidates, we pick the PM p_x with the highest score, as shown in line 5.

Line 6 checks if p_x will be overloaded after serving $v_{k,j}$. Specifically, function $B(p_x, v_{k,j})$ returns true if $C_x^{\text{CPU}} + \phi_{k,j}^{\text{CPU}} > \delta_{\text{CPU}} \Phi_x^{\text{CPU}}$ or $C_x^{\text{MEM}} + \phi_{k,j}^{\text{MEM}} > \delta_{\text{MEM}} \Phi_x^{\text{MEM}}$; otherwise, it returns false. If so, we transfer a VNF in p_x (possibly $v_{k,j}$) to another PM through the migration module in Algo. 3, which returns the new PM p_y that will serve $v_{k,j}$. The code is given in lines 7–9. However, if p_y is null, which means that VNF migration cannot be carried out, we then check if p_x (i.e., the PM found by line 5) has enough residual resources to serve $v_{k,j}$. If not, $v_{k,j}$ should be removed, since no PM can serve it. Theorem 2 analyzes the time complexity of Algo. 2.

Theorem 2: Suppose that Algo. 3 takes t_3 time. The time complexity of Algo. 2 is $O(\xi_P + \xi_N^2) + t_3$.

Proof: In Algo. 2, the for-loop in lines 2–4 takes $O(\xi_P)$ time. As only the 1-hop neighbors of $f_P(v_{k,j-1})$ are included in $\hat{\mathcal{P}}_\gamma$, we have $|\hat{\mathcal{P}}_\gamma| \leq \xi_N$. Thus, line 5 spends at most time of $\xi_N O(\xi_N + 1) = O(\xi_N^2)$. Except line 7 (which requires time of t_3), all other statements take $O(1)$ time. To sum up, the time complexity is $O(\xi_P + \xi_N^2) + t_3$. ■

C. Migration Module

This module is performed when a VNF $v_{k,j}$ makes the PM p_x found by line 5 in Algo. 2 overloaded, whose pseudocode is given in Algo. 3. Let $\hat{\mathcal{A}}_x$ be the set of VNFs already assigned to p_x. Line 1 finds the VNF from $\hat{\mathcal{A}}_x \cup \{v_{k,j}\}$ that consumes the most resources of p_x, which is calculated by

$$U(p_x, v_{m,n}) = \frac{1}{2} \times \left(\frac{\phi_{m,n}^{\text{CPU}}}{\Phi_x^{\text{CPU}}} + \frac{\phi_{m,n}^{\text{MEM}}}{\Phi_x^{\text{MEM}}} \right). \tag{9}$$

Here, $U(p_x, v_{m,n})$ is the average ratio of p_x's resources consumed by a VNF $v_{m,n}$. Then, we include all L-hop neighbors of p_x in a set $\hat{\mathcal{P}}_\varepsilon$, where $L \geq 1$, as shown in lines 2–5.

Algorithm 3: VNF migration

1 $v_{a,b} \leftarrow \arg\max_{v_{m,n} \in \hat{\mathcal{A}}_x \cup \{v_{k,j}\}} U(p_x, v_{m,n})$;
2 $\hat{\mathcal{P}}_\varepsilon \leftarrow \emptyset$;
3 **foreach** $p_i \in \hat{\mathcal{P}}$ **do**
4 **if** $H(p_i, p_x) \leq L$ **then**
5 $\hat{\mathcal{P}}_\varepsilon \leftarrow \hat{\mathcal{P}}_\varepsilon \cup \{p_i\}$;
6 $p_y \leftarrow \arg\max_{p_i \in \hat{\mathcal{P}}_\varepsilon} E(p_i, v_{a,b})$;
7 **if** $B(p_y, v_{a,b}) =$ *true* **then**
8 **return** null;
9 **else**
10 **if** $v_{a,b} = v_{k,j}$ **then**
11 **return** p_y;
12 **else**
13 transfer $v_{a,b}$ to p_y and **return** p_x;

In line 6, we choose a PM p_y from $\hat{\mathcal{P}}_\varepsilon$ whose $E(p_y, v_{a,b})$ value is the maximum. According to Eq. (8), p_y will be the PM that has the most residual resources (to serve $v_{a,b}$). However, if $B(p_y, v_{a,b})$ is true (i.e, $v_{a,b}$ makes p_y overloaded), $v_{a,b}$ cannot migrate to any PM in $\hat{\mathcal{P}}_\varepsilon$ (since other PMs in $\hat{\mathcal{P}}_\varepsilon$ must be also overloaded after serving $v_{a,b}$). Therefore, Algo. 3 returns a null value. On the other hand, there are two cases to be discussed when $B(p_y, v_{a,b})$ is false. If $v_{k,j}$ is the VNF that consumes the most resources, we transfer $v_{k,j}$ to p_y, as shown in lines 10–11. Otherwise, we transfer $v_{a,b}$ to p_y and let $v_{k,j}$ stay in p_x, as indicated in lines 12–13. Theorem 3 analyzes the time complexity of Algo. 3.

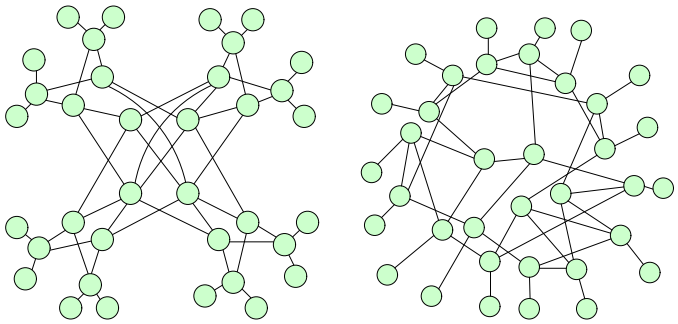
Theorem 3: The time complexity of Algo. 3 is $O(\xi_A + \xi_P)$, where ξ_A is the maximum number of VNFs served by a PM.

Proof: Line 1 requires $O(|\hat{\mathcal{A}}_x| + 1)$ time. The for-loop in lines 3–5 spends $O(\xi_P)$ time. Line 6 takes $O(|\hat{\mathcal{P}}_\varepsilon|)$ time. Other statements take $O(1)$ time. Since $|\hat{\mathcal{A}}_x| \leq \xi_A$ and $|\hat{\mathcal{P}}_\varepsilon| \leq \xi_P$, the time complexity is $O(\xi_A) + 2O(\xi_P) = O(\xi_A + \xi_P)$. ■

D. Discussion

We then discuss the rationale of the LBVD scheme. Unlike the existing solutions that use the fewest PMs to deploy VNFs, LBVD deploys VNFs with the aim of balancing PMs' loads. To do so, it scores each PM $p_i \in \hat{\mathcal{P}}$ based on the VNF $v_{k,j}$ to be deployed. The scoring function in Eq. (7) considers not only p_i's capability to serve $v_{k,j}$ (in terms of its residual resources) but also the capabilities of p_i's 1-hop neighbors to serve the next VNF $v_{k,j+1}$ of the same SFC. In this way, high-score PMs will be good candidates to deploy VNFs for load balance.

Moreover, the placement module differentiates between the first VNF and other VNFs in each SFC. Specifically, the first VNF is deployed on an idle PM. Doing so has two advantages. First, the VNFs of different SFCs could be distributed over the PMs in a cloud network, which helps spread the workload evenly. Second, the probability of successfully deploying other VNFs of the SFC can increase. If there is no idle PM, the first VNF will be deployed on the PM with the highest score.



(a) 4-ary fat tree (b) 4-ary jellyfish

Fig. 1. Network topologies considered in the simulation.

For each VNF $v_{k,j}$ in the rest of the SFC, it is deployed on a neighbor p_x of the PM assigned with the previous VNF $v_{k,j-1}$, whose score is the highest. However, if $v_{k,j}$ makes p_x overloaded, the migration module is performed to transfer one of p_x's VNFs (including $v_{k,j}$) to another PM that has sufficient resources. To reduce both deployment and migration costs, the VNF can migrate no more than L hops. The above designs distinguish our LBVD scheme with the existing solutions and help balance the loads of PMs in a cloud network. Theorem 4 shows the time complexity of our LBVD scheme.

Theorem 4: Suppose that $\xi_P > \xi_N^2$. Given ξ_S SFCs containing ξ_V VNFs, the time complexity of LBVD is $O(\xi_S \xi_P \xi_N + (\xi_V - \xi_S)(\xi_A + \xi_P))$.

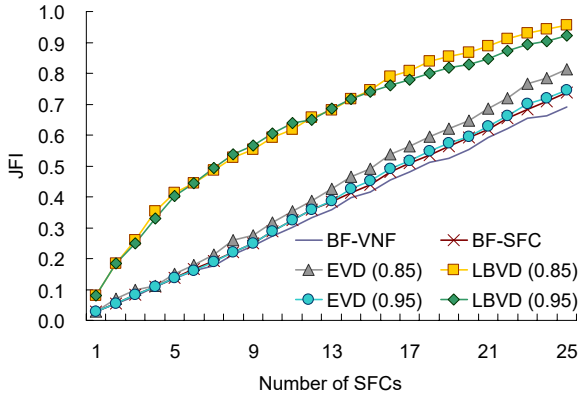
Proof: According to Theorems 1, 2, and 3, the time complexity is $\xi_S O(\xi_P \xi_N) + (\xi_V - \xi_S)[O(\xi_P + \xi_N^2) + O(\xi_A + \xi_P)]$. Since $\xi_P > \xi_N^2$, the complexity is simplified to $O(\xi_S \xi_P \xi_N + (\xi_V - \xi_S)(\xi_A + \xi_P))$. ■

V. Performance Evaluation

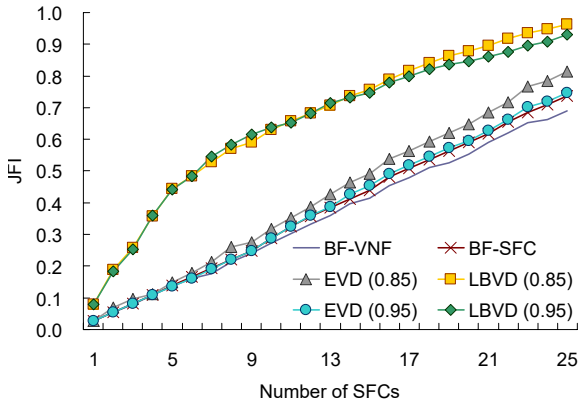
In the simulation, we consider a cloud network containing 36 PMs, which are organized into two topologies, namely *fat tree* [17] and *jellyfish* [18], as shown in Fig. 1. Each PM has 32 units of CPU resources and 64 units of memory resources (i.e., $\Phi_i^{\texttt{CPU}} = 32$ and $\Phi_i^{\texttt{MEM}} = 64$). Besides, there are 25 SFCs, where each SFC has 3 to 5 VNFs. The number of total VNFs is 100. Each VNF requires $[5, 12]$ units of CPU resources and $[10, 15]$ units of memory resources (i.e., $5 \leq \phi_{k,j}^{\texttt{CPU}} \leq 12$ and $10 \leq \phi_{k,j}^{\texttt{MEM}} \leq 15$).

As many methods aim to deploy VNFs on the fewest PMs to raise their utilization, we take the *best-fit (BF)* method for comparison. BF allocates the smallest free partition that meets the demand of a requesting process, which is widely used to improve hardware utilization [19]. Two variations of BF are proposed. The *BF-VNF* method finds a PM for each VNF such that the PM can fulfil the VNF's demand and has the minimum residual resources. For each SFC, the *BF-SFC* method picks a PM with the least residual resources which can allocate the maximum number of its VNFs. This operation is repeated until all VNFs of the SFC have been deployed or no PMs can be found to serve its VNFs.

Except BF-VNF and BF-SFC, we also compare our LBVD scheme with the *efficient VNF deployment (EVD)* method [15].



(a) fat-tree topology



(b) jellyfish topology

Fig. 2. Comparison on the JFI.

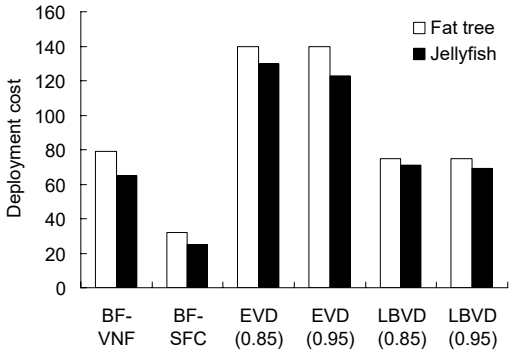


Fig. 3. Comparison on the deployment cost.

Similarly, EVD uses two thresholds δ_{CPU} and δ_{MEM} to check if a PM becomes overloaded. If so, a VNF of the PM will migrate to another PM. We set both thresholds to 0.85 and 0.95. In the LBVD scheme, we set $L = 3$.

Since the number of resources requested by VNFs is smaller than the number of resources possessed by PMs, the service ratio of every method is 100%, which means that all methods can successfully deploy the total VNFs. Thus, our discussion aim at both fairness (i.e., JFI) and cost (including the deployment and migration costs).

A. Comparison on Fairness

In this experiment, we add SFCs to the cloud network one by one and then evaluate the JFI of each method, whose result is given in Fig. 2. As can be seen, the JFI increases when more SFCs are added, because more PMs are assigned with VNFs and their loads become more "balanced". On the other hand, the performance of each method (in terms of the JFI) is similar on different network topologies. This means that changing the topology has insignificant impact on the fairness of PMs.

As BF's policy is to find a PM with the minimum (enough) resources to serve each VNF (or SFC), some PMs will have heavy loads while other PMs are almost idle. Thus, BF results in the lowest JFI. BF-SFC finds a PM to serve the maximum number of VNFs in one SFC, so it would incline to deploy VNFs on those PMs with more resources, as compared with BF-VNF. That is why BF-SFC has a higher JFI than BF-VNF. By transferring some VNFs of overloaded PMs to light-load PMs, EVD can increase the JFI. On the other hand, our LBVD scheme not only exploits VNF migration for load sharing, but also deploys VNF on high-score PMs for load balance. Thus, LBVD can substantially increase the JFI, which shows that it can efficiently improve the fairness of PMs in a cloud network. Notice that when thresholds (i.e., δ_{CPU} and δ_{MEM}) are set to 0.85, it is easier to trigger VNF migration in both EVD and LBVD. That is why they have better performance when thresholds are set to 0.85 (as compared with the case of 0.95).

B. Comparison on Costs

Fig. 3 shows the total deployment cost of all SFCs. When an SFC has a higher deployment cost, it requires more hops between PMs to be spanned to carry out the SFC's VNFs. Since the mean path length can reduce in the jellyfish topology than in the fat-tree topology [20], each PM would have more choices of neighbors for VNF deployment or migration in the jellyfish topology. Thus, all methods have lower deployment costs in the jellyfish topology.

Since BF will use as few PMs as possible to deploy VNFs, the deployment cost will be predictably low. Specifically, BF-SFC prefers deploying the VNFs of an SFC on the same PM, so it has the lowest deployment cost. On the other hand, EVD divides PMs into groups for VNF deployment, but it does not consider the locations of PMs in each group. Some VNFs of the same SFC may be deployed on PMs far away from each other. Thus, EVD results in the highest deployment cost. In our LBVD scheme, the placement module seeks to deploy a VNF $v_{k,j}$ on a 1-hop neighbor of the PM that serves the previous VNF $v_{k,j-1}$. In this way, LBVD can reduce the deployment cost. As compared with the EVD method, our LBVD scheme can save 46.4% and 44.6% of the deployment cost in the fat-tree and jellyfish topologies, respectively.

Fig. 4 compares the migration cost, which is defined as the total number of hop counts that VNFs migrate. Since BF-VNF and BF-SFC do not allow VNFs to migrate, their results are not presented in Fig. 4. As mentioned earlier, when thresholds δ_{CPU} and δ_{MEM} are set to 0.85, it becomes easier to trigger VNF migration in EVD and LBVD. Thus, the migration cost will increase accordingly. In the EVD method, the migration of

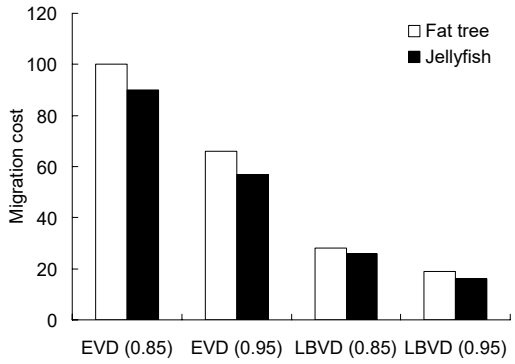


Fig. 4. Comparison on the migration cost.

VNFs does not consider the locations of PMs. On the contrary, our LBVD scheme restricts a VNF to migrate at most L hops. Thus, LBVD has a much lower migration cost than EVD. In particular, LBVD can save 71.6% and 71.5% of the migration cost in the fat-tree and jellyfish topologies, respectively, as compared with EVD.

VI. Conclusion and Future Work

NFV abstracts network functions as VNFs, which facilities service deployment and resource management. How to assign the VNFs of each SFC to PMs, namely the VNF deployment problem, makes a great impact on NFV's performance. Many methods deploy VNFs on the fewest PMs, which would cause unbalanced loads of PMs. This paper thus proposes the LBVD scheme, which comprises two modules. The placement module differentiates between the first VNF and other VNFs of each SFC and deploys them on high-score PMs. If a PM becomes overloaded, the migration module transfers its VNF to another PM for load balance. Simulation results show that the LBVD scheme has much higher JFIs than the BF-VNF, BF-SFC, and EVD methods in the fat-tree and jellyfish topologies of PMs. Moreover, LBVD can efficiently reduce both deployment and migration costs, as compared with EVD.

In this paper, we consider CPU and memory as the resources when scheduling VNFs. For future work, we will take account of more types of resources such as GPU (graphics processing unit), network, and FPGA (field programmable gate array). On the other hand, it is interesting to apply the LBVD scheme to other topologies of PMs (e.g., leaf-spine [21]) and evaluate the effect caused by the different topologies. Moreover, how to deploy VNFs in a cloud network to achieve Pareto optimality [22] deserves further investigation.

Acknowledgment

This work was supported by National Science and Technology Council, Taiwan under Grant 111-2221-E-110-023-MY2.

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] R. Zhou, "An online placement scheme for VNF chains in geo-distributed clouds," in *IEEE/ACM International Symposium on Quality of Service*, 2018, pp. 1–2.

[3] 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.

[4] T. Li, H. Zhou, and H. Luo, "A new method for providing network services: service function chain," *Optical Switching and Networking*, vol. 26, pp. 60–68, 2017.

[5] L. Qu, C. Assi, K. Shaban, and M. J. Khabbaz, "A reliability-aware network service chain provisioning with delay guarantees in NFV-enabled enterprise datacenter networks," *IEEE Transactions on Network and Service Management*, vol. 14, no. 3, pp. 554–568, 2017.

[6] Z. Yang, B. Chen, M. Dai, G. Su, and R. Lin, "VNF placement for service chaining in IP over WDM networks," in *Asia Communications and Photonics Conference*, 2018, pp. 1–3.

[7] S. I. Kim and H. S. Kim, "A VNF placement method considering load and hop count in NFV environment," in *International Conference on Information Networking*, 2020, pp. 707–712.

[8] O. Soualah, M. Mechtri, C. Ghribi, and D. Zeghlache, "A green VNFs placement and chaining algorithm," in *IEEE/IFIP Network Operations and Management Symposium*, 2018, pp. 1–5.

[9] I. Jang, S. Choo, M. Kim, S. Pack, and M. K. Shin, "Optimal network resource utilization in service function chaining," in *IEEE NetSoft Conference and Workshops*, 2016, pp. 11–14.

[10] X. Zhao, X. Jia, and Y. Hua, "An efficient VNF deployment algorithm for SFC scaling-out based on the proposed scaling management mechanism," in *Information Communication Technologies Conference*, 2020, pp. 166–170.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] Y. C. Wang and T. J. Hsiao, "URBM: user-rank-based management of flows in data center networks through SDN," in *IEEE International Conference on Computer Communication and the Internet*, 2022, pp. 142–149.

[18] 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.

[19] A. Silberschatz, G. Gagne, and P. B. Galvin, *Operating System Concepts*. Hoboken: Wiley, 2019.

[20] A. Singla, C. Y. Hong, L. Popa, and P. B. Godfrey, "Jellyfish: networking data centers randomly," in *USENIX Symposium on Networked Systems Design and Implementation*, 2012, pp. 225–238.

[21] S. Luo, H. Xing, and K. Li, "Near-optimal multicast tree construction in leaf-spine data center networks," *IEEE Systems Journal*, vol. 14, no. 2, pp. 2581–2584, 2020.

[22] 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.