Cost-Oriented VNF Placement for Load Balance

You-Chiun Wang and Cheng-Ying Tsai
Department of Computer Science and Engineering,
National Sun Yat-sen University, Kaohsiung, 804, Taiwan
Email: ycwang@cse.nsysu.edu.tw; bernietw123456@gmail.com

***Abstract*—Network function virtualization (NFV)** **is a technique to decouple network functions from hardware and convert them to software appliances, namely *virtual network functions (VNFs)*. Network services are carried out through *service function chains (SFCs)*, each with some VNFs. Given SFCs and *physical machines (PMs)*, this paper designs a *cost-oriented VNF placement (COVP)* method to assign each SFC's VNFs to PMs, with the objectives of maximizing served VNFs, balancing loads of PMs, and saving the total link cost to run SFCs. COVP allocates a PM for each VNF according to its position in the SFC as well as the PM's resource usage status. If a PM is busy, some VNFs are moved to other PMs in the vicinity for load balance. Both placement and movement of VNFs take link costs into account. Simulation results demonstrate that COVP can achieve the objectives efficiently.**

***Index Terms*—cost, load balance, NFV, VNF placement.**

I. Introduction

Traditionally, network functionalities are bundled into dedicated hardware called *middle boxes*, causing many drawbacks. For example, middle boxes are expensive and require domain-specific knowledge to maintain. It is also difficult to customize or develop network functionalities. Thus, the *network function virtualization (NFV)* technique is proposed to replace middle boxes by software appliances called *virtual network functions (VNFs)*. They can run on commodity *physical machines (PMs)* and move between PMs, thereby decoupling network functions from the underlying hardware. In this way, NFV can enhance scalability, facilitate management, and reduce expenditures [1].

Multiple VNFs constitute a *service function chain (SFC)* to perform a network service. How to pick PMs to run each SFC's VNFs, referred to as *VNF placement*, significantly affects NFV performance [2]. Many studies aim to improve PMs' resource utilization. Soualah et al. [3] model the placement problem via an integer linear program, whose goal is to optimize resource usage and revenue. In [4], a PM with the best-fit resources is first selected to place the VNFs of an SFC. With deep learning, the work [5] places VNFs to improve the service ratio and also decrease working PMs. Both studies [6], [7] improve resource consumption of PMs while ensuring service latency. The work [8] assigns every SFC a weight and places VNFs to increase the total weight of served SFCs. However, these methods may lead VNFs to gather on a few PMs to raise resource utilization. Doing so results in imbalanced loads among PMs, where some PMs are busy whereas others stay idle.

Some studies consider moving VNFs among PMs. The work [9] moves VNFs to keep high availability against PM failures. In [10], if a PM is busy, some of its VNFs are moved to other PMs for load balance. The study [11] restricts the hop counts for VNF movement to avoid an SFC's VNFs being placed on PMs far away from each other. However, these studies do not take account of the link cost between two adjacent PMs, which may significantly raise the cost taken to complete some SFCs.

This paper proposes a *cost-oriented VNF placement (COVP)* method to address the VNF placement issue with three goals:

- Serve as many VNFs as possible.
- Balance loads among PMs.
- Decrease the total link cost to execute SFCs.

For each SFC, COVP differentiates the placement of its head VNF from others. Moreover, the placement considers not only the suitability for a PM to serve that VNF but also the SFC's link cost. If a PM has spent many resources to serve VNFs and becomes overloaded, COVP moves some VNFs to other PMs without significantly increasing the link cost. With simulations, we show that our COVP method can serve all VNFs, achieve load balance among PMs, and save the total link cost.

II. Problem Formulation

Consider a network with a set $\mathcal{P}$ of PMs. Each PM $p_x \in \mathcal{P}$ has type-α resources with the capacity of Θ_x^α, where $\Theta_x^\alpha \in \mathbb{Z}^+$ and $\alpha \in \mathcal{R}$. In particular, $\mathcal{R}$ is the set of all resource types, and $\mathcal{R} = \{\texttt{CPU}, \texttt{MEM}\}$ (i.e., CPU computing power and memory storage). Moreover, we are given a set $\mathcal{S}$ of SFCs, where each SFC $s_i \in \mathcal{S}$ contains a series of VNFs $\mathcal{V}_i = \{v_{i,1}, \cdots, v_{i,m}\}$. Here, $v_{i,1}$ is the *head VNF* of s_i. Each VNF $v_{i,j} \in \mathcal{V}_i$ needs a number $\theta_{i,j}^\alpha$ of type-α resources, where $\theta_{i,j}^\alpha > 0$, $\forall \alpha \in \mathcal{R}$.

Let $\mathcal{L}$ be the set of all links between PMs. There exists a link $l_{x,y}$ in $\mathcal{L}$ if PMs p_x and p_y are neighbors. It is assigned a cost $c_{x,y}$ (e.g., $l_{x,y}$'s average packet delay). We also denote by $\psi(p_x, p_y)$ the *required link cost (RLC)* between PMs p_x and p_y. We calculate that $\psi(p_x, p_y) = \sum_{l_{x,y} \in \mathcal{L}'} c_{x,y}$, where $\mathcal{L}'$ is a set of links that constitute the shortest path from p_x to p_y.

Let ζ_x be the amount of PM p_x's load. It can be computed by $\zeta_x = \frac{1}{2} \sum_{\forall \alpha \in \mathcal{R}} (\sum_{s_i \in \mathcal{S}} \sum_{v_{i,j} \in \mathcal{V}_i} z_{i,j}^x \theta_{i,j}^\alpha / \Theta_x^\alpha)$, where $z_{i,j}^x$ is an indicator to reveal whether $v_{i,j}$ is placed on p_x ($z_{i,j}^x = 1$ if so or $z_{i,j}^x = 0$ otherwise). In other words, ζ_x is the average resource utilization for p_x to deal with its serving VNFs. Then, we measure the degree of load balance among PMs in $\mathcal{P}$ using Jain's fairness index [12] as follows:

$$J(\mathcal{P}) = (\sum\nolimits_{p_x \in \mathcal{P}} \zeta_x)^2 / (|\mathcal{P}| \times \sum\nolimits_{p_x \in \mathcal{P}} \zeta_x^2), \qquad (1)$$

where $1/|\mathcal{P}| \leq J(\mathcal{P}) \leq 1$. If $J(\mathcal{P})$ is larger, then loads of all PMs in $\mathcal{P}$ will be more balanced.

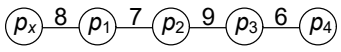


Fig. 1. Example of finding sets $\mathcal{P}_x^{\texttt{PL}}$ and $\mathcal{P}_x^{\texttt{MV}}$.

Our problem asks how to pick PMs in $\mathcal{P}$ for serving VNFs of each SFC in $\mathcal{S}$ to meet three objectives:

$$\text{maxmize} \quad \sum\nolimits_{p_x\in\mathcal{P}}\sum\nolimits_{s_i\in\mathcal{S}}\sum\nolimits_{v_{i,j}\in\mathcal{V}_i} z_{i,j}^x, \tag{2}$$

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

$$\text{minimize} \quad \sum\nolimits_{s_i\in\mathcal{S}}\sum\nolimits_{j=1}^{|\mathcal{V}_i|-1}\psi(\bar{\mathbf{p}}(v_{i,j}),\bar{\mathbf{p}}(v_{i,j+1})), \tag{4}$$

with constraints of

$$c_{x,y}\in[c_{\min},c_{\max}],\quad \forall l_{x,y}\in\mathcal{L}, \tag{5}$$

$$\theta_{i,j}^{\alpha}\le\min\nolimits_{p_x\in\mathcal{P}}\{\Theta_x^{\alpha}\},\quad \forall v_{i,j}\in\mathcal{V}_i,\forall s_i\in\mathcal{S},\forall\alpha\in\mathcal{R}, \tag{6}$$

$$z_{i,j}^x\in\{0,1\},\quad \forall v_{i,j}\in\mathcal{V}_i,\forall s_i\in\mathcal{S},\forall p_x\in\mathcal{P}, \tag{7}$$

$$\sum\nolimits_{p_x\in\mathcal{P}} z_{i,j}^x\le 1,\quad \forall v_{i,j}\in\mathcal{V}_i,\forall s_i\in\mathcal{S}, \tag{8}$$

$$\sum\nolimits_{s_i\in\mathcal{S}}\sum\nolimits_{v_{i,j}\in\mathcal{V}_i} z_{i,j}^x\theta_{i,j}^{\alpha}\le\Theta_x^{\alpha},\quad \forall p_x\in\mathcal{P},\forall\alpha\in\mathcal{R}. \tag{9}$$

Objective functions in Eqs. (2), (3), and (4) indicate serving the maximum VNFs, balancing loads of PMs, and minimizing the total link cost to run SFCs. Here, $\bar{\mathbf{p}}(v_{i,j})$ denotes the PM that serves VNF $v_{i,j}$. As for constraints, Eq. (5) imposes lower and upper bounds on the cost of each link in $\mathcal{L}$. Eq. (6) points out that a VNF cannot ask for more resources than the capacity of any PM. Eq. (7) restricts $z_{i,j}^x$'s value to 0 or 1. In Eq. (8), each VNF can be placed on at most one PM. Since a PM may serve multiple VNFs, Eq. (9) means that the total number of resources asked by them cannot overtake the PM's capacity.

III. The Proposed COVP Method

To check whether a PM p_x becomes overloaded, we employ a threshold δ_α on type-α resource consumption ratio for p_x ($0.5<\delta_\alpha<1$). Let u_x^α be the number of consumed type-α resources of p_x, as calculated by $u_x^\alpha=\sum_{s_i\in\mathcal{S}}\sum_{v_{i,j}\in\mathcal{V}_i} z_{i,j}^x\theta_{i,j}^\alpha$. When $u_x^\alpha/\Theta_x^\alpha>\delta_\alpha$, where $\alpha=\texttt{CPU}$ or $\texttt{MEM}$, p_x is considered *overloaded*. In this case, if p_x still meets Eq. (9)'s constraint, p_x is capable of serving additional VNFs. However, we shall prevent PMs in $\mathcal{P}$ from being overloaded as much as possible for load-balancing purposes.

To efficiently reduce the total link cost to run an SFC s_i, we need to constrain the relative positions (i.e., PMs) to place any two adjacent VNFs of s_i. Suppose that $v_{i,j}\in\mathcal{V}_i$ is placed on PM p_x. Then, the subsequent PM $v_{i,j+1}\in\mathcal{V}_i$ is allowed to be placed on a PM in set $\mathcal{P}_x^{\texttt{PL}}$, of which any PM whose RLC to p_x does not exceed threshold $\varphi_{\texttt{PL}}$. Assume that SFC s_i has n_i VNFs and no VNFs are moved after being placed. Doing so ensures that the total link cost to run s_i is at most $(n_i-1)\varphi_{\texttt{PL}}$. In addition, we define a set $\mathcal{P}_x^{\texttt{MV}}$, where any PM whose RLC to p_x is below or equal to threshold $\varphi_{\texttt{MV}}$ and $\varphi_{\texttt{MV}}>\varphi_{\texttt{PL}}$. When we need to move $v_{i,j}$, it will be restricted to moving to a PM only in $\mathcal{P}_x^{\texttt{MV}}$. Fig. 1 gives an example. By setting $\varphi_{\texttt{PL}}$ to 15 and $\varphi_{\texttt{MV}}$ to 25, we have $\mathcal{P}_x^{\texttt{PL}}=\{p_1,p_2\}$ and $\mathcal{P}_x^{\texttt{MV}}=\{p_1,p_2,p_3\}$.

Algorithm 1: PM Selection

Data: $v_{i,j}$ (VNF to be served) and p_o (PM where the previous VNF $v_{i,j-1}$ is placed)
Result: p_y (PM that serve $v_{i,j}$)

1 **if** $j=1$ **then** /* $v_{i,j}$ is the head VNF */
2 **if** $\mathcal{P}_{\texttt{IDL}}\neq\emptyset$ **then**
3 $\mathcal{P}_{\texttt{CAN}}\leftarrow\mathcal{P}_{\texttt{IDL}}$;
4 **else if** $\mathcal{P}_{\texttt{AVL}}\neq\emptyset$ **then**
5 $\mathcal{P}_{\texttt{CAN}}\leftarrow\mathcal{P}_{\texttt{AVL}}$;
6 **else**
7 **return** null;
8 $p_y\leftarrow\arg\max_{p_x\in\mathcal{P}_{\texttt{CAN}}} f(p_x,v_{i,j})$;
9 **else** /* otherwise (so we have $\mathcal{P}_o^{\texttt{PL}}\neq\emptyset$) */
10 **if** $\mathcal{P}_{\texttt{IDL}}\cap\mathcal{P}_o^{\texttt{PL}}\neq\emptyset$ **then**
11 $\mathcal{P}_{\texttt{CAN}}\leftarrow\mathcal{P}_{\texttt{IDL}}\cap\mathcal{P}_o^{\texttt{PL}}$;
12 **else if** $\mathcal{P}_{\texttt{AVL}}\cap\mathcal{P}_o^{\texttt{PL}}\neq\emptyset$ **then**
13 $\mathcal{P}_{\texttt{CAN}}\leftarrow\mathcal{P}_{\texttt{AVL}}\cap\mathcal{P}_o^{\texttt{PL}}$;
14 **else**
15 **return** null;
16 $F\leftarrow 0$, $p_y\leftarrow$ null;
17 **foreach** $p_x\in\mathcal{P}_{\texttt{CAN}}$ **do**
18 **if** $f(p_x,v_{i,j})\ge F+\Delta$ *or* $(f(p_x,v_{i,j})\ge F-\Delta$ *and* $\psi(p_o,p_x)<\psi(p_o,p_y))$ **then**
19 $p_y\leftarrow p_x$, $F\leftarrow f(p_x,v_{i,j})$;
20 $u_y^\alpha\leftarrow u_y^\alpha+\theta_{i,j}^\alpha$, $\forall\alpha\in\mathcal{R}$;
21 **return** p_y;

For each VNF $v_{i,j}$, we adopt a function $f(p_x,v_{i,j})$ to assess the suitability of placing $v_{i,j}$ on a PM p_x by

$$f(p_x,v_{i,j})=\varepsilon_1\min_{\alpha\in\mathcal{R}} e(p_x,v_{i,j},\alpha)+\varepsilon_2\sum\nolimits_{p_y\in\mathcal{P}_x^{\texttt{PL}}}\max\{\min_{\alpha\in\mathcal{R}} e(p_y,v_{i,j+1},\alpha),0\}, \tag{10}$$

where $0\le\varepsilon_1,\varepsilon_2\le 1$ and $\varepsilon_1+\varepsilon_2=1$. Moreover, $e(p_x,v_{i,j},\alpha)$ is the proportion of residual type-α resources for p_x after serving $v_{i,j}$, as calculated by $e(p_x,v_{i,j},\alpha)=1-((u_x^\alpha+\theta_{i,j}^\alpha)/\Theta_x^\alpha)$. In the case that $v_{i,j}$ is the last VNF of its SFC, we set $\varepsilon_1=1$ and $\varepsilon_2=0$. The idea behind Eq. (10) is as follows: If p_x can have more resources left after serving $v_{i,j}$ and there are more choices of PMs in $\mathcal{P}_x^{\texttt{PL}}$ to place the next VNF $v_{i,j+1}$, then the value of $f(p_x,v_{i,j})$ becomes larger. In this case, p_x is a better candidate to place $v_{i,j}$.

Given each VNF $v_{i,j}$ (following the sequence) of an SFC $s_i\in\mathcal{S}$, Algorithm 1 gives the pseudocode of selecting a PM p_y from $\mathcal{P}$ to serve $v_{i,j}$. Let us define two sets of PMs: $\mathcal{P}_{\texttt{IDL}}=\{p_x\in\mathcal{P}\mid u_x^\alpha=0,\forall\alpha\in\mathcal{R}\}$ and $\mathcal{P}_{\texttt{AVL}}=\{p_x\in\mathcal{P}\mid u_x^\alpha>0,\exists\alpha\in\mathcal{R}$ and $\Theta_x^\alpha-u_x^\alpha\ge\theta_{i,j}^\alpha,\forall\alpha\in\mathcal{R}\}$. Specifically, $\mathcal{P}_{\texttt{IDL}}$ represents the set of idle PMs, and $\mathcal{P}_{\texttt{AVL}}$ denotes the set of non-idle PMs that have enough resources to meet $v_{i,j}$'s request.

In Algorithm 1, the code in lines 1–8 handles s_i's head VNF (i.e., $j=1$ for $v_{i,j}$). Let $\mathcal{P}_{\texttt{CAN}}$ be the set of candidate PMs. To balance loads among PMs, we shall prioritize idle PMs to take

Algorithm 2: VNF Movement

Data: p_b (overloaded PM in terms of type-β resources)
Result: $v_{\mathtt{mv}}$ (p_b's VNF to be moved) and p_t (target PM to serve $v_{\mathtt{mv}}$)

1 $v_{\mathtt{mv}} \leftarrow$ null, $p_t \leftarrow$ null, $C = \infty$;
2 $\mathcal{P}_{\mathtt{CAN}} \leftarrow \mathcal{P}_{\mathtt{NOV}} \cap \mathcal{P}_b^{\mathtt{MV}}$;
3 **if** $\mathcal{P}_{\mathtt{CAN}} = \emptyset$ **then**
4 **return** null;
5 $(p_y, q_\beta) \leftarrow \max_{p_x \in \mathcal{P}_{\mathtt{CAN}}} \delta_\beta \Theta_x^\beta - u_x^\beta$;
6 $q_{\beta'} \leftarrow \delta_{\beta'} \Theta_y^{\beta'} - u_y^{\beta'}$;
7 SORT(Ω_b, β);
8 **foreach** $v_{i,j} \in \Omega_b$ **do**
9 **if** $\theta_{i,j}^\beta \le q_\beta$ *and* $\theta_{i,j}^{\beta'} \le q_{\beta'}$ **then**
10 $v_{\mathtt{mv}} \leftarrow v_{i,j}$ and break;
11 **if** $v_{\mathtt{mv}} =$ *null* **then**
12 **return** null;
13 **foreach** $p_x \in \mathcal{P}_{\mathtt{CAN}}$ **do**
14 **if** $\delta_\beta \Theta_x^\beta - u_x^\beta < \theta_{\mathtt{mv}}^\beta$ *or* $\delta_{\beta'} \Theta_x^{\beta'} - u_x^{\beta'} < \theta_{\mathtt{mv}}^{\beta'}$ **then**
15 continue;
16 **if** $p_t =$ *null or* $\psi(p_b, p_x) < C$ **then**
17 $p_t \leftarrow p_x$, $C \leftarrow \psi(p_b, p_x)$;
18 $u_b^\alpha \leftarrow u_b^\alpha - \theta_{\mathtt{mv}}^\alpha$, $u_t^\alpha \leftarrow u_t^\alpha + \theta_{\mathtt{mv}}^\alpha$, $\forall \alpha \in \mathcal{R}$;
19 **return** $v_{\mathtt{mv}}$ and p_t;

charge of $v_{i,j}$. In the case of no idle PM, the candidate set will be $\mathcal{P}_{\mathtt{AVL}}$. However, if $\mathcal{P}_{\mathtt{AVL}}$ is empty, meaning that no PM has enough resources to serve $v_{i,j}$, the PM selection algorithm will return null. The code is shown in lines 2–7. Then, among all candidates in $\mathcal{P}_{\mathtt{CAN}}$, line 8 picks out the one whose $f(p_x, v_{i,j})$ value is the maximum (i.e., with the most suitability).

On the other hand, the code in lines 9–19 deals with s_i's other VNFs (i.e., $j > 1$ for $v_{i,j}$). In this situation, since $v_{i,j}$'s previous VNF $v_{i,j-1}$ has been placed on PM p_o, we can obtain the set $\mathcal{P}_o^{\mathtt{PL}}$. Like lines 2–7, the code in lines 10–15 finds the candidate set $\mathcal{P}_{\mathtt{CAN}}$. The difference is that we need to consider the intersection of $\mathcal{P}_{\mathtt{IDL}}$ (or $\mathcal{P}_{\mathtt{AVL}}$) and $\mathcal{P}_o^{\mathtt{PL}}$. Then, the for-loop in lines 17–19 chooses a PM from $\mathcal{P}_{\mathtt{CAN}}$ such that the PM has a larger $f(p_x, v_{i,j})$ value while its RLC to p_o can be smaller (to reduce s_i's total link cost). To do so, we use a variable F to store the $f(p_x, v_{i,j})$ value of the currently selected PM in $\mathcal{P}_{\mathtt{CAN}}$ (i.e., p_y). Initially, we have $F = 0$ and $p_y =$ null (thereby, $\psi(p_o, p_y) = \infty$). There are two cases to make us replace p_y by p_x and update F's value accordingly: 1) p_x has obviously larger suitability than p_y (i.e., $f(p_x, v_{i,j}) \ge F + \Delta$, where Δ is a small value), and 2) the suitability difference between p_x and p_y is insignificant and p_x has a smaller RLC to p_o than p_y to p_o (i.e., $f(p_x, v_{i,j}) \ge F - \Delta$ and $\psi(p_o, p_x) < \psi(p_o, p_y)$). Finally, line 20 updates the amount of resource usage for the selected PM p_y (i.e., u_y^α), and line 21 returns p_y.

If a PM p_b is overloaded in terms of type-β resources (i.e., $u_b^\beta > \delta_\beta \Theta_b^\beta$), Algorithm 2 helps select one of its serving VNFs (i.e., $v_{\mathtt{mv}}$) and move $v_{\mathtt{mv}}$ to another PM p_t. Let $\mathcal{P}_{\mathtt{NOV}}$ be the set of non-overloaded PMs: $\mathcal{P}_{\mathtt{NOV}} = \{p_x \in \mathcal{P} \mid u_x^\alpha / \Theta_x^\alpha < \delta_\alpha, \forall \alpha \in \mathcal{R}\}$. Evidently, the candidate set $\mathcal{P}_{\mathtt{CAN}}$ will be the intersection of $\mathcal{P}_{\mathtt{NOV}}$ and $\mathcal{P}_b^{\mathtt{MV}}$, as presented in line 2. If $\mathcal{P}_{\mathtt{CAN}}$ is not empty, line 5 finds the PM (i.e., p_y) in $\mathcal{P}_{\mathtt{CAN}}$ such that it has the most supportable type-β resources (i.e., $\delta_\beta \Theta_x^\beta - u_x^\beta$, and this value is stored in q_β). Then, line 6 stores the number of supportable type-β' resources of p_y, where $\beta' \in \mathcal{R} \setminus \{\beta\}$.

Let Ω_b denote the set of VNFs currently served by p_b. The code in lines 7–12 picks out a VNF from Ω_b whose request for type-β resources can be "best fit" to q_β. More concretely, line 7 sorts VNFs in Ω_b decreasingly based on their numbers of requested type-β resources. The for-loop in lines 8–10 picks VNF $v_{\mathtt{mv}}$ such that its requested number of type-β resources is the closest to (but not above) q_β and that of type-β' resources does not exceed $q_{\beta'}$. This guarantees that there exist PMs in $\mathcal{P}_{\mathtt{CAN}}$ with enough resources to serve $v_{\mathtt{mv}}$. If no such VNF can be found, Algorithm 2 returns null, as shown in lines 11–12.

The for-loop in lines 13–17 chooses the target PM (i.e., p_t) to take over $v_{\mathtt{mv}}$. The if-statement in lines 14–15 excludes those PMs without sufficient supportable resources to serve $v_{\mathtt{mv}}$ (that is, once such a PM serves $v_{\mathtt{mv}}$, the PM will become overloaded or even run out of resources). Among other PMs in $\mathcal{P}_{\mathtt{CAN}}$, we pick the one that has the smallest RLC to p_b (i.e., $\psi(p_b, p_x)$ is the minimum). Then, line 18 updates resource usage of both p_b and p_t (as $v_{\mathtt{mv}}$ is moved from p_b to p_t).

IV. Performance Evaluation

In the simulation, we consider a network with 99 PMs. Each PM has 32 and 64 units of CPU and memory resources. These PMs are arranged into two topologies popularly used in large networks (e.g., data center networks): *fat tree* and *jellyfish*. In a fat tree, PMs form a three-layer tree structure. Any two leaf PMs have multiple shortest paths between them [13]. Jellyfish considers a random regular graph. Non-leaf PMs are randomly connected and have the same degree [14]. Every link is given a cost arbitrarily chosen from $[1, 20]$. There are 25 SFCs, each with 10∼12 VNFs. A VNF requires $[5, 7]$ and $[9, 11]$ units of CPU and memory resources, respectively. The total number of VNFs is set to 275.

Three methods are chosen for comparison:

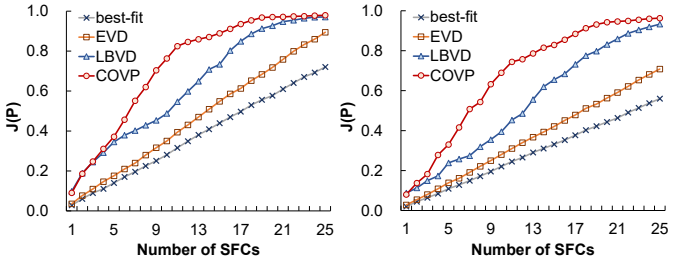
- Best-fit [4]: Find a PM with the best-fit resources to serve VNFs of an SFC.
- Efficient VNF deployment (EVD) [10]: Move VNFs from overloaded PMs to lightly loaded PMs.
- Load-balanced VNF deployment (LBVD) [11]: Consider placing VNFs on PMs with high scores and move VNFs for load-balancing purposes.

In EVD, LBVD, and COVP, δ_α is set to 0.8, $\forall \alpha \in \mathcal{R}$. Besides, we set $\varphi_{\mathtt{PL}} = 20$ (i.e., the maximum link cost) and $\varphi_{\mathtt{MV}} = 40$.

Table I compares the number of served VNFs, the fairness index $J(\mathcal{P})$ on PMs' loads, and the total link cost to run SFCs. As can be seen, each method serves a total of 275 VNFs in $\mathcal{S}$. Regarding fairness, all methods perform better in the fat-tree topology, where PMs form a well-structured tree. The best-fit method places VNFs of an SFC on the same PM, which results

TABLE I
PERFORMANCE COMPARISON (LEFT: FAT TREE, RIGHT: JELLYFISH).

method	VNFs	$J(\mathcal{P})$	link cost	VNFs	$J(\mathcal{P})$	link cost
best-fit	275	0.73	1404	275	0.56	1977
EVD	275	0.89	3334	275	0.71	4324
LBVD	275	0.97	2869	275	0.93	3145
COVP	275	0.98	1730	275	0.96	2424



(a) fat tree (b) jellyfish

Fig. 2. Effect of the number of SFCs on $J(\mathcal{P})$.

in the worst fairness. EVD can move VNFs from overloaded PMs to other PMs, which improves fairness. Both LBVD and COVP choose PMs with more resources to serve VNFs first. This way, they can significantly raise the $J(\mathcal{P})$ value. For the total link cost, due to the nature of a random regular graph, all methods perform worse in the jellyfish topology. The best-fit method has the lowest link cost, as it prefers placing VNFs of each SFC on the same PM. EVD may move some VNFs of an SFC to those PMs far away from each other. Doing so greatly raises the total link cost. LBVD considers hop counts between PMs when placing and moving VNFs of each SFC, thereby reducing the total link cost. Instead of simply considering hop counts, COVP takes account of the real costs of links. Hence, it can substantially save the total link cost compared to LBVD. To sum up, our COVP method performs the best among all methods in terms of improving fairness and total link cost.

We also evaluate the effect of the number of SFCs on $J(\mathcal{P})$ (i.e., fairness). To do so, starting from 1 SFC, we iteratively add one SFC to the network until there are 25 SFCs. Fig. 2 shows the experimental result. EVD outperforms the best-fit method since it can move VNFs for load sharing among PMs. LBVD considers resource usage of PMs when placing VNFs, so it can significantly increase the $J(\mathcal{P})$ value. Compared to LBVD, COVP is capable of finding out more suitable VNFs of overloaded PMs to be moved, which helps further improve the fairness. The result in Fig. 2 demonstrates that our COVP scheme can efficiently achieve load balance among PMs when there are different numbers of SFCs in the network.

V. CONCLUSION

To facilitate resource and service management in a network, NFV carries out the abstraction of network functions as VNFs. The VNF placement problem, which determines how to assign PMs to serve the VNFs of each SFC, has a significant impact on NFV's performance. Consequently, this paper proposes the COVP method with the objectives of serving as many VNFs as possible, balancing loads among PMs, and reducing the total link cost to run SFCs. It assesses the suitability of placing a VNF on each PM and also distinguishes the placement of the head VNF from others in each SFC. Once a PM is overloaded, COVP finds its appropriate VNFs to be moved to other PMs in the vicinity. Simulation results reveal that COVP can improve fairness while reducing the total link cost as compared to the best-fit, EVD, and LBVD methods.

ACKNOWLEDGMENT

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

REFERENCES

[1] M. Zoure, T. Ahmed, and L. Reveillere, "Network services anomalies in NFV: survey, taxonomy, and verification methods," *IEEE Trans. Network and Service Management*, vol. 19, no. 2, pp. 1567–1584, 2022.

[2] J. Sun *et al.*, "A survey on the placement of virtual network functions," *Journal of Network & Computer Applications*, vol. 202, pp. 1–37, 2022.

[3] 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.

[4] C. Pham *et al.*, "Traffic-aware and energy-efficient VNF placement for service chaining: joint sampling and matching approach," *IEEE Trans. Services Computing*, vol. 13, no. 1, pp. 172–185, 2020.

[5] S. Qi *et al.*, "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.

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

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

[8] D. H. P. Nguyen *el al.*, "Virtual network function placement for serving weighted services in NFV-enabled networks," *IEEE Systems Journal*, vol. 17, no. 4, pp. 5648–5659, 2023.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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 Trans. Network and Service Management*, vol. 15, no. 4, pp. 1422–1434, 2018.

[14] 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.