Efficient Load Balancing for the VNF Deployment with Placement Constraints

Chaoqun You*, Le Min Li*
*University of Electronic Science and Technology of China

***Abstract*—The Virtual Network Function (VNF) deployment problem in Network Function Virtualization (NFV) is of broad theoretical and practical interests. As the traffic surges in data centers, VNF deployment requires load balancing across the servers to avoid possible congestions caused by the uneven distribution of VNFs. In addition, VNFs always specify placement constraints, restricting them to run only on part of the servers. Therefore, we study the load balancing problem for VNF deployment with placement constraints. Despite the rich bodies of recent work on the constrained load balancing problem using the network flow algorithms, they all suffer exponential complexities, leading to unbearable running time in practical executions. In this paper, we propose a new load balancing policy termed Constrained Min-max Placement (CMMP) that schedules VNFs in a way similar to the max-min allocation, where we try to assign the most possible VNFs to the poorest loaded server. The online scheduler for CMMP has a logarithmic time complexity and is simple enough to implement in practice. Trace-driven simulations show that the online CMMP speeds up at least two orders of magnitude of running time comparing to other network flow algorithms.**

***Index Terms*—load balancing, VNF deployment, placement constraints**

I. Introduction

Network Functions (NFs) are ubiquitous in today's data centers [1]. They are deployed to perform diverse processing functions, such as firewalls, proxies, and Intrusion Detection Systems (IDS), on the traffic flows passing through them. A recently developed approach, Network Function Virtualization (NFV), moves the traffic processing from dedicated hardware appliances to Virtual Network Functions (VNFs) running on general-purpose commodity servers [2]. NFV allows VNFs to duplicate and pick service locations from multiple servers, thus offering more flexibility to the traffic processing. Therefore, following a fundamental problem is to map the VNFs to the servers while achieving a specific objective, which we refer to as the VNF deployment problem.

Load Balancing is a main concern in deploying the VNFs. It is important because the uneven distribution of VNFs often leads to unstable traffic processing performance and make the NFV server prone to faults [3–5]. According to the studies shown in [5], some software implementations using 100% CPU with only $\sim$700Mbps incoming traffic suffer from more than 20% packet loss. This loss becomes more severe as the traffic surges in data centers [6].

In addition, most VNFs specify *placement constraints* [3, 4, 7, 8], restricting them to run on a particular class of servers. For example, the CUDA traffic must pass through servers with GPUs [9], while a DNS service can only deploy on servers with IP addresses. According to Google, over 50% of its jobs have simple, yet strict constraints about the servers they can run on [10].

Therefore, in this paper we study the load balancing problem for the deployment of VNFs with placement constraints. Our goal is to equalize the loads among the servers while satisfying constraints. A natural approach to achieve this goal is *min-max allocation*, where we try to assign the least possible VNFs to the most loaded server [11–13]. Min-max allocation can be easily transformed into an Integer Linear Programming (ILP) problem. Although there has been much work on finding the optimal solutions to the ILP problem, its solutions are *offline*. That is, whenever the data center environment changes (*e.g.*, a new flow arrives or a server shuts down), the ILP problem would be recomputed and reconfigure the allocation, and the flow that is being processed by a VNF on a server may be paused due to the reconfiguration of VNFs. In general, data-intensive frameworks do not support such pauses [14]. On the other hand, although it is possible to obtain online solutions using a network flow algorithm instead of an ILP, all the network flow algorithms suffer expensive computational costs with exponential time complexities [11, 13, 15], making it is too slow to deploy VNFs in a large-scale data center.

In this paper, we propose a simple, yet effective load balancing scheme, termed Constrained Min-Max Placement (CMMP), that takes a logarithmic time complexity $O(\log m)$ for the online deployment of VNFs with placement constraints, where m is the number of servers in the cluster. CMMP schedules VNFs in a way similar to the *max-min allocation* [14], where we try to assign the most possible VNFs to the poorest loaded server. We use max-min allocation because, unlike min-max allocation, max-min allocation can degenerate to the *progressive filling* (PF) algorithm [16], a greedy online algorithm with a low, logarithmic complexity. Therefore, we try to transform the min-max allocation problem into a max-min allocation problem to use PF for the online deployment of VNFs. However, min-max allocation and max-min allocation are two separate optimization problems with different objectives. Consequently, our challenge is to find the relationship between the optimal solutions of the two problems. To understand the relationship, we firstly introduce theorems rigorously describing under what conditions the min-max allocation solution is equal to the max-min allocation solution. Based on these theorems, CMMP successfully transforms the original min-max allocation into a max-min allocation problem. Like other max-min allocation

978-1-5386-8088-9/19/$31.00 ©2019 IEEE

solutions, the general approach we take is PF [14], making the CMMP implementations simple enough in practice. With PF, the online CMMP scheduler greedily deploys the next VNF to the server with the lowest current load whose placement constraints satisfies.

To summarize, our contributions in this work are three-fold:

- We introduce three theorems in Section III-B to rigorously describe under what conditions the optimal solution to the min-max allocation required by the constrained load balancing problem is equal to the optimal solution to the max-min allocation problem.
- Based on the theorems, we propose CMMP in Section III-C, to support the constrained load balancing VNF deployment. We then propose the offline and online schedulers for CMMP in Section IV, studying their time complexities, respectively.
- Through trace-driven simulations in Section V, CMMP is verified to efficiently balance the loads among the servers by using only several seconds in scheduling more than 10000 VNFs, showing a runtime two orders of magnitudes less than the other network work flow algorithms.

II. System Model and Problem Formulation

A. System Model

Modeling demands: Traffic flows are often required to pass through an ordered sequence of NFs, which is typically referred to as a Service Function Chain (SFC). For instance, a flow may be required to go through a firewall, then an IDS, and finally through a proxy [7]. Moreover, different users may require different SFCs [8]. Therefore in this paper, the input traffic is modeled by a set of flow demands $\mathcal{D} = \{d_1, d_2, \ldots, d_p\}$, where each flow $d_k \in \mathcal{D}$ is associated with a SFC $\texttt{sfc}(d_k)$ of length $l(d_k)$. A SFC $\texttt{sfc}(d_k)$ is an ordered VNF sequence in $\mathcal{F}$, where $\mathcal{F} = \{f_1, f_2, \ldots, f_n\}$ is the set of function types, *e.g.*, firewalls, IDS, *etc*. Let Q_i denote the total number of VNFs required by all the SFCs in $\mathcal{D}$ on function type f_i, and each function type f_i requires an execution time of t_i.

Modeling placement constraints: Consider n types of functions and m commodity servers for VNFs to deploy on. Let $\mathcal{V}$ denote the set of servers and let E represent the edge set, where $(f_i, v_r) \in E$ if $f_i \in \mathcal{F}$ can be assigned to server $v_r \in \mathcal{V}$. Thus, E captures the *placement constraints* of the network functions. An example of such a system can be represented by the bipartite graph $G = (\mathcal{F} \cup \mathcal{V}, E)$ shown in Fig. 1, where the dashed lines indicates that a server can be used by a particular type of function.

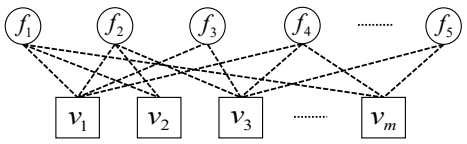


Fig. 1. Modeling placement constraints

Modeling server load: Throughout this paper, we assume that the number of demands p is much larger than the number of servers m in the system (*i.e.*, $p \gg m$). This indicates that if several VNFs are assigned to a server, then a VNF would be executed right after another VNF finishes without waiting for its former VNF from the same SFC to be finished. Let x_{ir} be the number of function f_i assigned to server v_r. Then the *load* on server v_r is denoted as $\sum_{i\in\mathcal{F}} x_{ir}t_{ir}$, which is the total execution time of all the VNFs assigned to server r. The heaviest load among all the servers, which is referred to as the *makespan*, can be denoted as $\max_{r\in\mathcal{V}} \sum_{i\in\mathcal{F}} x_{ir}t_i$.

B. Problem Formulation

Our goal is to balance the loads among the servers while satisfying the placement constraints. A natural way to achieve this goal is *min-max allocation*, where we try to give least possible VNFs to the most loaded server [11, 13]. Min-max allocation can be formalized as an optimization problem with the objective to minimize the maximal load among the servers, *i.e.*, to minimize the makespan, which we show as follows,

$$\min_{\{x_{ir}\}} \quad \max_{r\in\mathcal{V}} \sum_{i\in\mathcal{F}(r)} x_{ir}t_i, \tag{1a}$$

$$s.t. \quad \sum_{i\in\mathcal{F}(r)} x_{ir} \le \sum_{i\in\mathcal{F}(r)} Q_i, \quad (1 \le r \le m), \tag{1b}$$

$$\sum_{r\in\mathcal{V}} x_{ir} = Q_i, \quad (1 \le i \le n), \tag{1c}$$

$$x_{ir} \in \mathbb{Z}_+^N, \quad (1 \le i \le n, 1 \le r \le m). \tag{1d}$$

where $\mathcal{F}(r)$ represents the set of function types that can be placed on v_r. (1a) denotes the placement constraints of the functions, that the number of VNFs assigned to v_r is less than or equal to the total number of VNFs that can be deployed on this server, and (1b) denotes that the total number f_i allocated to all servers is equal to Q_i.

III. Constrained Min-Max Placement

A. Inefficiencies of the Conventional Solutions

The min-max allocation problem shown in (1) can be easily transformed into an *Integer Linear Programming* (ILP) problem [17] by replacing the original objective with an auxiliary variable M such that $\max_{r\in\mathcal{V}} \sum_{i\in\mathcal{F}} x_{ir}t_i \le M$. However, the optimal solutions to the ILP problem is offline. Whenever the system environment changes, such as a new traffic arrives or a server shuts down, the scheduler has to recompute the ILP problem and reconfigure the VNF deployment. A flow that is being processed by a VNF on a server may be paused due the reconfiguration. In general, this pause is not supported by data-intensive frameworks in datacenters [14].

Although it is possible to find online solutions using a network flow algorithm instead of an ILP [11, 13], these algorithms suffer exponential complexities with respect to the number of VNFs and the number of servers. For example, the CANCELALL has a time complexity of $O(Q^{1.5}m\log Q)$, where Q is the total number of VNFs [15]. Meanwhile, the latest network flow algorithm MEC^3 has a time complexity of $O(Q^2m^3)$ [13]. Such expensive computation costs do no apply to high rates of decision making. Therefore, for a data-intensive

compute cluster such as a data center where thousands of decisions are made per second [14], a simple enough algorithm with a low time complexity remains to be an open challenge.

B. The Relationship Between Min-max and Max-min Allocation

Max-min allocation, on the other hand, can also be transformed into an ILP problem [17]. However, unlike the min-max allocation, max-min allocation can degenerate to the *progressive filling* (PF) algorithm [16], whose complexity is much lower. Moreover, PF is a greedy algorithm that can be naturally extended to the case with integer solutions [14], making it is simple enough to be implemented in practice. Considering these nice properties possessed by the max-min allocation, in this section we study the relationship between the max-min and the min-max allocation to facilitate our design of the algorithm to achieve min-max allocation among the servers.

Initially, we give formal definitions of the max-min fair vector and the min-max fair vector, which are derived from the max-min and min-max allocation problems, respectively. Consider a set $\mathcal{X} \subset \mathbb{R}^N$, we define the max-min and min-max fair vectors with respect to $\mathcal{X}$ as follows,

Definition 1 (Max-min Fair Vector)**.** *An N-dimensional vector $X = \{x_1, x_2, \ldots, x_N\}$ is max-min fair on set $\mathcal{X}$, when it is impossible to increase the value of x_i without reducing the value of another element i' with a value $x_{i'} \leq x_i$.*

Definition 2 (Min-max Fair Vector)**.** *An N-dimensional vector $X = \{x_1, x_2, \ldots, x_N\}$ is min-max fair on set $\mathcal{X}$, when it is impossible to decrease the value of x_i without increasing the value of another element i' with a value $x_{i'} \geq x_i$.*

We then propose Theorem 1 to describe under what circumstance the max-min fair vector is equal to the min-max fair vector. By the equalization of two vectors X and Y defined on $\mathcal{X}$, we mean that the corresponding items of the two vectors are equal, *i.e.*, $x_i = y_i$ $(i = 1, 2, \ldots, N)$.

Theorem 1. *Given a set $\mathcal{X} \subset \mathbb{R}^N$ such that the sum of elements in any two N-dimensional vectors defined on $\mathcal{X}$ is the same. That is, if two N-dimensional vectors $X, Y \in \mathcal{X}$, then $\sum_{i=1}^{N} x_i = \sum_{i=1}^{N} y_i = Q$. If X and Y are the* max-min *and* min-max *fair vectors defined on $\mathcal{X}$, respectively, then we have $X = Y$.*

Proof: Since X is the max-min fair vector on $\mathcal{X}$, we have

$$x_1 = x_2 = \cdots = x_N.$$

Likewise, since Y is the min-max fair vector on $\mathcal{X}$, we have

$$y_1 = y_2 = \cdots = y_N.$$

Both results are not fresh and can be found in [12, 16]. Since the sum of the components of X and Y are the same, that is, $\sum_{i=1}^{N} x_i = \sum_{i=1}^{N} y_i = Q$, we will always have

$$x_i = y_i = \frac{Q}{N}, \qquad i = 1, 2, \ldots, N.$$

Therefore, we have $X = Y$. That is, the max-min fair vector and the min-max fair vector are exactly the same vectors as long as the sum of their elements are the same. ∎

Given the set of all flow demands, it is easy to compute their total loads, which is $\sum_{i \in \mathcal{F}} Q_i t_i$. Therefore, according to Theorem 1, if VNFs are infinitely divisible and have no placement constraints, then the load balancing problem can be regarded as a max-min fair allocation problem among the servers, and the load assigned to each server is the same, which is $\frac{\sum_{i \in \mathcal{F}} Q_i t_i}{m}$. However, VNFs are usually indivisible and have to be scheduled as whole entities in practice. Therefore, in Theorem 2 we turn our attentions to how the relationship between max-min fair vector and min-max fair vector is affected in the discrete scenario.

Theorem 2. *Given a set $\mathcal{X} \subset \mathbb{Z}_+^N$ such that the sum of elements in any two N-dimensional vectors defined on $\mathcal{X}$ is the same. That is, if two N-dimensional vectors $X, Y \in \mathcal{X}$, then $\sum_{i=1}^{N} x_i = \sum_{i=1}^{N} y_i$. If X and Y are the* max-min *and* min-max *fair vectors defined on $\mathcal{X}$, respectively, then we have $X = Y$.*

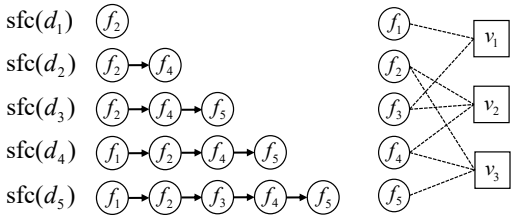
Proof: Since X is the max-min fair vector defined on $\mathcal{X}$ and all elements in X are integers, then the absolute value of the maximum difference between each pair of the elements in X is 1. That is, $|x_i - x_j| \leq 1$ $(i \neq j)$. Therefore, X can be rewrited as $\{x, x, \ldots, x, x+1, x+1, \ldots, x+1\}$, and the sum of all the elements in X is $Nx + a$, where a is a constant. Similarly, Y can be rewrited as $\{y, y, \ldots, y, y+1, y+1, \ldots, y+1\}$, and the sum of all the elements in Y is $Ny + b$, where b is a constant. Meanwhile, the sum of the elements in X and Y is the same, that is, $Nx + a = Ny + b = Q$. Therefore, we have $x = y$, $a = b$. At this point, obviously $X = Y$. ∎

From Theorem 2, we indicate that when VNFs are indivisible and have no placement constraints, the load balancing problem can also be regarded as a max-min fair allocation problem among the servers. Based on this theorem, we continue to consider the case when VNFs have placement constraints in Theorem 3.

Theorem 3. *Given a set $\mathcal{X} \subset \mathbb{Z}_+^N$ such that the sum of elements in any two N-dimensional vectors defined on $\mathcal{X}$ is the same. That is, if two N-dimensional vectors $X, Y \in \mathcal{X}$, then $\sum_{i=1}^{N} x_i = \sum_{i=1}^{N} y_i$. If $0 \leq x_i \leq a_i$ and $0 \leq y_i \leq b_i$, where both a_i and b_i are constants, $\sum_{i=1}^{N} a_i \geq Q$ and $\sum_{i=1}^{N} b_i \geq Q$. If X and Y are the* max-min *and* min-max *fair vectors defined on $\mathcal{X}$, respectively, then we have $X = Y$.*

Proof: For any N-dimensional vector $Z \in \mathcal{X}$, we firstly define the "order mapping" $\Gamma : \mathbb{Z}+^N \to \mathbb{Z}+^N$ as the mapping that sort Z in non-decreasing order, that is, $\Gamma(z_1, z_2, \ldots, z_N) = (z_{(1)}, z_{(2)}, \ldots, z_{(N)})$ such that $z_{(1)} \leq z_{(2)} \leq \cdots \leq z_{(N)}$ and for all i, $z_{(i)}$ is one of the z_js. Therefore, the ordering mappings of X and Y give us $x_{(1)} \leq x_{(2)} \leq \ldots x_{(N)}$ and $y_{(1)} \leq y_{(2)} \leq \cdots \leq y_{(N)}$.

In a proof by contradiction we assume that $X \neq Y$, then $\Gamma(X) \neq \Gamma(Y)$. Let $x_{(i)} = y_{(i)} + \delta_{(i)}$, then we have $\Gamma(X) = \{y_{(1)} + \delta_{(1)}, y_{(2)} + \delta_{(2)}, \ldots, y_{(N)} + \delta_{(N)}\}$, where $\sum_{i=1}^{N} \delta_{(i)} = 0$. Assume there is only one pair of elements in X that is different from those two elements in the corresponding positions in Y,



(a) Sample demands of function chains. (b) Placement constraints.

Fig. 2. Sample SFCs and their placement constraints.

then from $\Gamma(X)$ we have

$$y_{(1)} \leq \cdots \leq y_{(i)} + \delta_{(i)} \leq \cdots \leq y_{(j)} + \delta_{(j)} \leq \cdots \leq y_{(N)}, \quad (2)$$

where $i < j$. Since Y is the min-max fair vector defined on $\mathcal{X}$, then it is impossible to decrease the value of $y_{(j)}$ without increasing the value of $y_{(i)}$. Therefore, we have $\delta_{(i)} > 0$, $\delta_{(j)} < 0$. Combine (2) and $\Gamma(Y)$, we have

$$\begin{aligned} y_{(1)} \leq \cdots \leq y_{(i)} < y_{(i)} + \delta_{(i)} \leq \\ \ldots \leq y_{(j)} + \delta_{(j)} < y_{(j)} \leq \ldots y_{(N)}. \end{aligned} \quad (3)$$

This inequation is the same with

$$x_{(1)} \leq \cdots \leq y_{(i)} < x_{(i)} \leq \cdots \leq x_{(j)} < y_{(j)} \leq \ldots x_{(N)}. \quad (4)$$

Since X the max-min fair vector defined on $\mathcal{X}$, it is impossible to increase the value of $x_{(j)}$ without decreasing the value of $x_{(i)}$. However, this is not the case as it is shown in the inequation (4). The value of $x_{(j)}$ can increase to $y_{(j)}$ by decreasing the value of $x_{(i)}$ to $y_{(i)}$. Therefore, the assumption of $X \neq Y$ fails. Consequently, we have $X = Y$. ■

C. Constrained Min-max Placement

Now that the constrained load balancing problem can be regarded as the max-min allocation problem among the servers, our policy, *Constrained Min-max Placement* (CMMP), applies *max-min fair allocation with respect to the servers' loads*. That is, CMMP attempts to recursively maximize the allocation of the poorest loaded server, followed by the second poorest loaded server, and so on.

To understand CMMP, consider the example shown in Fig. 2. Fig. 2a is a sample case of SFC demands, while Fig. 2b shows the placement constraints of the VNFs in Fig. 2a. Assume that the execution time of all VNFs are the same in this example, then the load of a server can be simplified to be the total number of VNFs allocated to that server. The CMMP allocation is given by Fig. 3: $\mathcal{A}(v_1) = \{f_1, f_1, f_3\}$ (*i.e.*, the assignment of the VNFs to v_1), $\mathcal{A}(v_2) = \{f_2, f_2, f_2, f_4, f_4, f_4\}$, $\mathcal{A}(v_3) = \{f_2, f_4, f_2, f_5, f_5, f_5\}$. To see this is indeed a CMMP allocation, note that it is impossible to increase the minimal load across any subset of servers by reshuffling the loads between the servers in that subset.

IV. Offline and Online Allocations for CMMP

A. Computing CMMP Offline

We initially explore how to compute the CMMP allocations offline. We assume that there are fixed set of demands D with placements constraints on the servers, and that all the servers are currently idle. This represents the offline setting [14].

Like other max-min fair solutions, the general approach we take is PF. In the case of CMMP, PF starts by increasing all servers' loads equally until the maximum possible level M_1. Once M_1 is achieved, the scheduler continues to increases the loads equally of the servers that are still available for VNFs to be deployed on and then achieves a new maximal level M_2. This process repeats until all the VNFs the $\texttt{sfc}(d_k)$ ($d_k \in \mathcal{D}$) asks for have been placed in order subject to their placement constraints. Algorithm 1 shows the pseudo-code of the offline CMMP scheduler.

Algorithm 1: Offline CMMP scheduler

Procedure `OfflineSolver()`
- $c := 1$ ▷ Current round
- $M_0 := 0$, $S_0 := \emptyset$ ▷ Max level and servers stuck
- **while** *true* **do**
  - $(M_c, x_{i,r}) := \texttt{LP}(k, M_1, \ldots, M_{c-1}, S_1, \ldots, S_{c-1})$
  - $S_c := \texttt{Saturated}(c, M_1, \ldots, M_{c-1}, S_1, \ldots, S_{c-1})$
- **end**
- **if** $S_1 \cup \cdots \cup S_c = \mathcal{V}$ **then**
  - **return** $x_{i,r}$ ▷ Return matrix of allocations
- **end**
- $c := c + 1$

Procedure $\texttt{Saturated}(c, M_1, \ldots, M_{c-1}, S_1, \ldots, S_{c-1})$
- $U := \{1, \ldots, m\} \setminus (S_1 \cup \cdots \cup S_{c-1})$ ▷ Active servers
- $S := \emptyset$ ▷ Servers saturated in this round
- **for** $u \in U$ **do**
  - $S_c := U \setminus \{u\}$
  - $(M_{c+1}, x_{ir}) := \texttt{LP}(c+1, M_1, \ldots, M_c, S_1, \ldots, S_c)$
  - **if** $M_{c+1} = M_c$ **then**
    - $S := S \cup \{u\}$
  - **end**
  - **return** S
- **end**

Procedure $\texttt{LP}\,(c, M_1, \ldots, M_{c-1}, S_1, \ldots, S_{c-1})$

$$\begin{aligned} \max \quad & M_c, \\ s.t. \quad & \sum_{i \in \mathcal{F}(r)} x_{ir} \geq M_l, \quad (1 \leq l \leq c-1), \\ & \sum_{i \in \mathcal{F}(r)} x_{ir} \leq \sum_{i \in \mathcal{F}(r)} Q_i, \quad (1 \leq r \leq m), \\ & M_l, x_{i,r} \in \mathbb{Z} + . \end{aligned}$$

- **return** (M_c, x_{ir})

Example Fig. 3 shows how the offline CMMP works on the example shown in Fig. 2 with the solid lines indicating allocations, whereas dashed lines indicating that the VNF can be deployed on a particular server. In round 1 each server has a load constraint of the form $\sum_{i=1}^{5} x_{ir} \geq M_1$. The program attempts to maximize M_1. It finds an allocation shown in Fig. 3(a) after round 1, where each server is allocated 3 of the VNFs ($M_1 = 3$). Now a saturation is done on v_1 as there are no more VNFs available to be placed on it and only v_2 and v_3 are still active. Thus the program enters round 2, with the load constraints updated to $\sum_{i=1}^{5} x_{i1} \geq M_1$, $\sum_{i=1}^{5} x_{i2} \geq M_2$ and $\sum_{i=1}^{5} x_{i3} \geq M_2$. This time M_2 is maximized to 3 and the

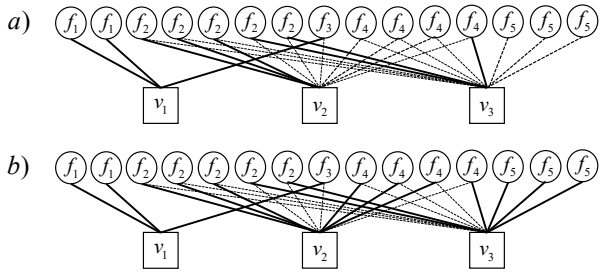


Fig. 3. (a) Allocation after round 1 of the offline CMMP algorithm, in which all servers are given 3 NFs. (b) Allocation after round 2, in which v_2 and v_3 are still active and each gets another 3 NFs.

program finds the allocation given by Fig. 3(b). Both v_2 and v_3 get another 3 VNFs and no more VNFs are left.

Complexity Since the number of servers is m, then the maximal possible round number is m. In each round, the offline CMMP solves a LP problem, leading its time complexity to be $O(LP(n, m))$. Therefore the total complexity of the offline CMMP scheduler is $O(mLP(n, m))$. In most practical cases there are solutions with polynomial complexities for such a LP scheduler [12].

B. Online Scheduling for CMMP

In practice, traffic flows arrive and depart dynamically and it is hard to predict the profile of the upcoming demands precisely. Therefore, an online CMMP scheduler is desirable to schedule the demands dynamically. In addition, in large-scale data-intensive frameworks such like data centers, thousands of scheduling decisions are made per second [14]. Thus, the online scheduler should also be simple enough to be scheduled at high speeds. Fortunately, the PF algorithm that achieves max-min fairness is a greedy algorithm and is simple enough to be implemented in practice. Our online CMMP scheduler, which is a natural extension of PF, can be described as follows,

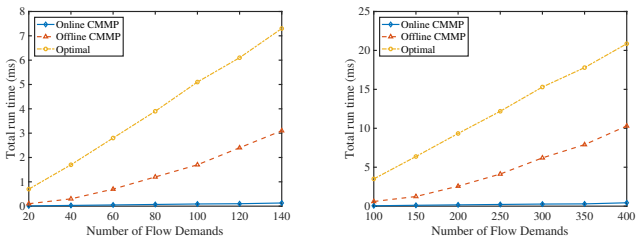
Assign the next VNF to the server with lowest current load whose placement constraints satisfies.

Complexity The complexity of online CMMP is the same with the PF algorithm, which is $O(\log m)$, as the online CMMP scheduler only need to sort the loads of the m servers and determine which one is the poorest loaded.

V. Trace-driven Simulations

In practical data centers where flows arrive dynamically and VNFs are deployed as whole entities, only the online CMMP scheduler is desirable for the VNF deployment. Therefore, we evaluate the online CMMP by comparing it to the ideal solutions (*i.e.*, the offline CMMP scheduler and the optimal scheduler) and the latest network flow solutions (*i.e.*, MEC[3] and CANCELL). The offline CMMP scheduler and the optimal scheduler shown in (1) are computed using IBM ILOG CPLEX.

Specifically, we would like to (i) confirm experimentally that the online CMMP runs at high speeds through runtime comparison. (ii) measure the average makespan to verify the similarities of the algorithms in providing minimal makespan.



(a) Internet2 (b) germany50

Fig. 4. Runtime comparison.

TABLE I
Running time of the schedulers with 1000 nodes and 10000 demands.

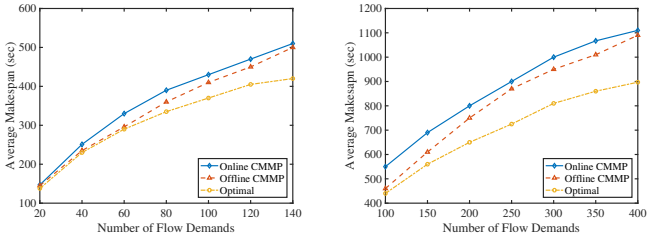
Scheduler	Total Runtime (s)
Online CMMP	1.5
Offline CMMP	234.7
CANCELALL	607.1
MEC[3]	589.6

(iii) investigate the VNF queueing delay to see if the low complexity of the online CMMP scheduler is achieved at the expense of its VNFs' queueing delays.

General Setup. We conduct simulations on two real-world datasets that contain the network topology and traffic traces measured at different times, and on random Erdös-Rényi graphs [18]. The first dataset is the `Internet2` [19] including a topology of 12 nodes, 30 links and traffic traces with 144 demands. The second dataset is the `germany50` [20] including 50 nodes, 88 links and traffic traces with 400 demands. The SFC of a demand is composed of 1 to 6 NFs uniformly chosen at random from a set of 30 functions. On the other hand, we assign placement constraints to NFs according to the model proposed at Google [10], where a task can be treated as a VNF in our case. In our experiments, we follow the same constraints' setting with [14] so as to constitute a nontrivial workload with a large fraction of highly constraint VNFs, that the average VNF could use 38% of the server nodes; 40% of the VNFs could use less than 20% of the nodes; and 30% of them could use less than 10% of the nodes.

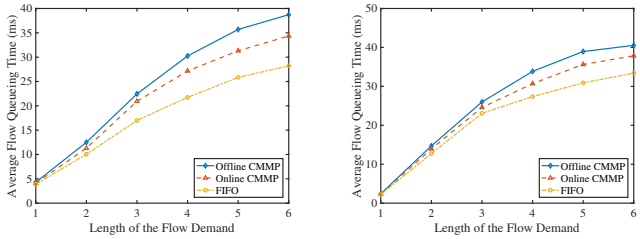
Total Run Time. Fig. 4 shows the runtime comparison using the two real-world datasets with respect to the number of flow demands. We observe that the online CMMP scheduler runs much faster than the other schedulers by showing a runtime one order of magnitude less than the others. Further, we use the random Erdös-Rényi graph to generate a large-scale cluster including 1000 nodes and 10000 flow demands. Table I lists the total runtime comparison between different schedulers. All the other schedulers are more than two orders of magnitude slower than the online CMMP scheduler, meaning that they would be too slow to be deployed in a real large-scale data center.

Average Makespan. Fig. 5 shows the average makespan experienced by the schedulers. Apparently, the average makespan of the optimal scheduler is the lower bound of the other schedulers since it is the ideal solution to the load balancing problem where VNFs are infinitely divisible and the demands



(a) Internet2 (b) germany50

Fig. 5. Average makespan comparison.



(a) Internet2 (b) germany50

Fig. 6. Average flow completion time comparison.

profile are given in advance. Meanwhile, the offline CMMP scheduler behaves better than the online CMMP as it is also an ideal solution assuming the demands profile is known ahead of time. As the number of flow demands increases, the gap between the optimal scheduler and the offline CMMP is also increasing. This is reasonable since the more flow demands, the more VNFs that may not obtain integer solutions in the optimal scheduler, thus leading a larger offset from the offline scheduler with integer solutions. On the other hand, we observe that the online CMMP scheduler approximates to the offline CMMP scheduler well. This approximation verifies that the online CMMP scheduler can efficiently balance the loads among servers, as the theorem indicates.

Average Queueing Time. Fig. 6 shows the average flow completion time of the schedulers. We compare the CMMP with the FIFO algorithm this time because there exits no queueing delay using the optimal scheduler. We observe that the average queueing time of FIFO is smaller than the other two. This indicates that the VNFs in CMMP usually have to wait longer than in FIFO to balance the loads among the servers. Further, the average VNF queueing time in the offline CMMP is larger than the online CMMP scheduler by up to 12%. This is reasonable as the online CMMP scheduler deploy VNFs in a greedy manner such that the VNF that arrives first would always assign to poorest loaded server and gets its service first on that server. Combine Fig. 4 and Fig. 6, we indicate that the significant improvement in the running speeds of the online CMMP scheduler is achieved at the expense of some latency.

VI. Acknowledgements

This work is partially supported by NSFC Fund (1671130, 61301153, 612711656, 61671124), 973 Program (2013CB329103), Program for Changjiang Scholars and Innovative Research Team (PCSIRT) in University and the 111 Project B14039.

VII. Conclusions

In this paper, we studied the VNF deployment problem in data centers with placement constraints. We proposed a new load balancing policy, CMMP, that deployed the VNFs in a way similar to the max-min allocation. In particular, CMMP greedily deployed the next VNF to the current poorest server with the lowest load whose placement constraints satisfied. The online scheduler of CMMP had a logarithmic time complexity $O(\log m)$, as opposed to the exponential complexities suffered by the network flow algorithms that can also solve the online constrained load balancing problem. Trace-driven simulations showed CMMP could speed up the deployment by at least two orders of magnitude of running time compared to the network flow algorithms.

References

[1] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar, "Making middleboxes someone else's problem: network processing as a cloud service," *ACM SIGCOMM CCR*, vol. 42, no. 4, pp. 13–24, 2012.

[2] P. Quinn and T. Nadeau, "Problem statement for service function chaining," *IETF SFC Tech. Rep.*, 2015.

[3] F. Carpio, S. Dhahri, and A. Jukan, "VNF placement with replication for load balancing in NFV networks," in *IEEE ICC*, 2017.

[4] T.-M. Pham, S. Fdida, H. T. T. Binh *et al.*, "Online load balancing for network functions virtualization," in *IEEE ICC*, 2017.

[5] T. Wang, H. Xu, and F. Liu, "Multi-resource load balancing for virtual network functions," in *IEEE ICDCS*, 2017.

[6] M. Honda, Y. Nishida, C. Raiciu, A. Greenhalgh, M. Handley, and H. Tokuda, "Is it still possible to extend TCP?" in *ACM SIGCOMM*, 2011.

[7] F. Bari, S. R. Chowdhury, R. Ahmed, R. Boutaba, and O. C. M. B. Duarte, "Orchestrating virtualized network functions," *IEEE Transactions on Network and Service Management*, vol. 13, no. 4, pp. 725–739, 2016.

[8] A. Tomassilli, F. Giroire, N. Huin, and S. Pérennes, "Provably efficient algorithms for placement of service function chains with ordering constraints," in *IEEE INFOCOM*, 2018.

[9] "CUDA," https://developer.nvidia.com/cuda-zone.

[10] B. Sharma, V. Chudnovsky, J. L. Hellerstein, R. Rifaat, and C. R. Das, "Modeling and synthesizing task placement constraints in google compute clusters," in *ACM SoCC*, 2011.

[11] N. J. Harvey, R. E. Ladner, L. Lovász, and T. Tamir, "Semi-matchings for bipartite graphs and load balancing," *Journal of Algorithms*, vol. 59, no. 1, pp. 53–78, 2006.

[12] B. Radunovic and J.-Y. Le Boudec, "A unified framework for max-min and min-max fairness with applications," *IEEE/ACM Transactions on networking*, vol. 15, no. 5, pp. 1073–1083, 2007.

[13] J. P. Champati and B. Liang, "Efficient minimization of sum and differential costs on machines with job placement constraints," in *IEEE INFOCOM*, 2017.

[14] A. Ghodsi, M. Zaharia, S. Shenker, and I. Stoica, "Choosy: Max-min fair sharing for datacenter jobs with constraints," in *ACM EuroSys*, 2013.

[15] J. Fakcharoenphol, B. Laekhanukit, and D. Nanongkai, "Faster algorithms for semi-matching problems," *ACM Transactions on Algorithms (TALG)*, vol. 10, no. 3, p. 14, 2014.

[16] D. P. Bertsekas, R. G. Gallager, and P. Humblet, *Data networks*. Prentice-Hall International New Jersey, 1992, vol. 2.

[17] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[18] B. Bollobás, "Random graphs," in *Modern graph theory*. Springer, 1998, pp. 215–252.

[19] "Internet2 research network topology and traffic matrix," http://www.cs.utexas.edu/yzhang/research/AbileneTM/.

[20] S. Orlowski, R. Wessäly, M. Pióro, and A. Tomaszewski, "Sndlib 1.0survivable network design library," *Networks: An International Journal*, vol. 55, no. 3, pp. 276–286, 2010.