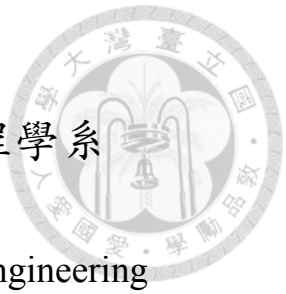


國立臺灣大學電機資訊學院資訊工程學系
碩士論文
Department of Computer Science and Information Engineering
College of Electrical Engineering and Computer Science
National Taiwan University
Master Thesis

利用工作分配來提升數據中心網路中服務鏈的部署
On Improving Service-chain Deployment with Job Dispatching in the Data Center Networks

許俊淵
Chun-Yuan Hsu

指導教授：周承復博士
Advisor: Cheng-Fu Chou, Ph.D.

中華民國 106 年 7 月
July 2017

doi:10.6342/NTU201703605

國立臺灣大學碩士學位論文
口試委員會審定書

利用工作分配來提升數據中心網路中服務鏈的部署

On Improving Service-Chain Deployment with Job Dispatching in the Data Center Networks

本論文係許俊淵君（學號 R04922136）在國立臺灣大學資訊工程學系完成之碩士學位論文，於民國 106 年 7 月 27 日承下列考試委員審查通過及口試及格，特此證明

口試委員：

（指導教授）

系 主 任 趙坤茂

誌謝

首先，我想感謝我的指導教授周承復教授，謝謝老師在我研究所就讀期間一直很有耐心的指導我，在研究過程中遇到困難時給予我許多建議，指引我正確的研究方向，我才能順利完成這篇論文，並且讓我在思考問題的能力上能有所成長。另外也要感謝我的口試委員，吳曉光教授、蔡子傑教授、廖婉君教授、林俊宏教授，能撥空前來參加我的口試，並且給予我的研究一些建議，讓我的論文可以更完整。

同時也要感謝大煒學長與銘宏學長，總是熱心的回答我許多問題，並且給予我研究上許多建議與幫助。接著要感謝實驗室的夥伴們，凱文、欣鈺、鈞謙以及榜榜，這兩年來受到你們許多幫助，研究的過程很開心有你們的陪伴。也謝謝這一路上遇到的所有人，讓我學習到許多自身不足的地方，並且充實了我研究所生活。

最後感謝我的家人，總是在背後支持我，讓我無後顧之憂，才能順利完成碩士學位。

doi:10.6342/NTU201703605

摘要

Middlebox 在現今網路中扮演著非常重要的角色，像是防火牆 (firewall)、代理伺服器 (Proxy)、網路地址轉換 (Network Address Translation) 等。然而往往我們需要封包依序經過這些 middlebox，這樣的需求我們稱為服務鏈。傳統的 middlebox 大多是高成本並且位置固定的硬體設備，這使得在部署上有所限制，並且導致浪費許多頻寬。網路功能虛擬化能夠將這些 middlebox 轉換成軟體並且能在一般伺服器上執行，這使得在部署上更有彈性。問題在於如何去部署服務鏈才會更好。由於在資料中心網路或是企業網路中往往有許多的 job，我們應該同時分配這些 job 來做服務鏈的部署才能得到最佳解。並且過去的研究皆並未考慮到使用的伺服器數目，這可能導致電力耗費提高，因此我們尋找出頻寬與運算資源以及使用的伺服器數目之間的關聯性，並且將整個問題公式化成最佳化問題，因為這個最佳化問題十分困難，因此我們提出一個 rack aware 的演算法有效率地解決此問題。經由模擬實驗的結果證明加入 job 分配能有效提升服務鏈部署的表現，並且我們提出的演算法能大幅減少運算時間，同時也能有效地找到接近最佳解的解。

關鍵字：Middlebox、網路功能虛擬化、服務鏈

doi:10.6342/NTU201703605

Abstract

Middleboxes, such as firewall, proxy and NAT, play an important role in the existing network. However, we usually need traffic go through these middleboxes in specific order. We call these requests Service-chain. Most of middleboxes are expensive hardware-based appliances with fixed placement. It is not flexible to deploy and might cause we waste lots of bandwidth. Network function virtualization transforms these middleboxes to software to be executed on general-purpose servers. It helps us do deployment more flexible. The question is how to do service chain deployment would be better. Due to there have many jobs in the data center networks and enterprise networks, we should do service chain deployment and job dispatching together so as to find the global optimal solution. Moreover, most prior works do not aware the number of used servers and cause the higher electricity cost. Therefore, we discover the relation among the bandwidth, CPU resource and number of used servers. And we formulate the service chain deployment and job dispatching problem as an integer linear programming model. However, it is very difficult to solve this optimization model. Hence, we propose a Rack-aware Service-chain deployment and Job dispatching (RSJ) algorithm to find the solution effectively and efficiently. The simulation result shows that we can improve the service chain deployment by doing job dispatching together. Our proposed algorithm significantly reduces the processing time and also can find a solution close to the optimal solution.

Keywords: Middlebox, Network function virtualization, service chain

doi:10.6342/NTU201703605

Contents

doi:10.6342/NTU201703605

doi:10.6342/NTU201703605

List of Figures

doi:10.6342/NTU201703605

List of Tables

doi:10.6342/NTU201703605

Chapter 1

Introduction

1.1 Middleboxes

Middleboxes, or network functions, provide many functions other than packet forwarding between source and destination in the networks. These functions primarily deploy for security and performance or other benefits. In the aspect of security, the network functions, such as firewalls, Instruction Detection Systems(IDS) and Instruction Prevention Systems(IPS), filter packets based on security rules or detecting the malicious activities. In the aspect of performance, the network functions, such as proxy and WAN optimizer, provide some function for accessing the data faster or increasing the data transfer efficiency. Some network functions provide benefits other than security or performance such as Network Address Translation(NAT) which modifies packet header for ease of rerouting traffic without readdressing each host. Nowadays, middleboxes already play an importance role in the network. Figure 1.1 [1] shows the different types middleboxes and number of these middleboxes deployed in an enterprise network. It also shows that middleboxes are already commonly used in the network.

However, there still have many requests that need network traffic traverse an ordered

doi:10.6342/NTU201703605

Appliance type	Number
Firewalls	166
NIDS	127
Conferencing/Media gateways	110
Load balancers	67
Proxy caches	66
VPN devices	45
WAN optimizers	44
Voice gateways	11
Middleboxes total	636
Routers	≈ 900

Figure 1.1: Middleboxes in the enterprise network [1]

set of network functions. For example, we might need a flow go through firewall first and then go through IDS before proxy so as to filter suspicious requests. These requests are called Service Function Chain or Service Chain [2].

Middleboxes or network functions are implemented on special-purpose hardware-based appliances. However, there would have several disadvantages when we use legacy middleboxes. The first one is the price. Generally, the price of special-purpose hardware-based appliances is very expensive. Second, we need to choose a fixed location to place each middlebox. The fixed placement might cause a flow should traverse a longer path to use middlebox and waste bandwidth. Also, it might very difficult to maintain these middleboxes. For instance, we might have a period time can not use service provided by middleboxes when middleboxes failed and need to be upgraded or moved to another location.

doi:10.6342/NTU201703605

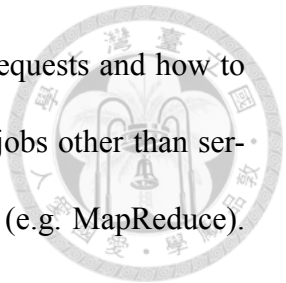
1.2 Network Function Virtualization

To cope the disadvantages of the special-purpose hardware-based appliances, Network Function Virtualization(NFV) [3] has been proposed recently to transform the commodity hardware-base appliances to software layer. NFV allows network functions to be executed on the virtual machines hosted on the general-purpose servers. After middleboxes turn to software layer, we call it Virtual Network Functions(VNF). NFV reduces the cost of expensive hardware-base appliances and makes the network functions deployment can be more flexible. We could elastically deploy network functions at the location which we want. NFV also makes these VNF easy to maintain and upgrade, we just install a new VNF to another server without suspending the service.

1.3 Motivation

NFV helps us deploy VNFs more flexible. However, the question is how to do service chain deployment using NFV would be better? First, it is very important to route service chain request carefully because we should make sure this request go through the request service in a specific order. If we use a bad deployment, this flow might waste lots of bandwidth and might also increase the latency. When a flow request a VNF, we can install the required VNF on the shortest path of the flow using NFV to save the bandwidth. On the other hand, if the CPU resources is very limited, we might route this flow go through a longer path to reuse VNF and then save CPU resources. We can see there has a competition between bandwidth and CPU resources. But with this property, we have an opportunity based on the network condition to make network better. There already have many researches on the service chain deployment. But most researches only focus

doi:10.6342/NTU201703605



on how to use available network resources to deploy service chain requests and how to tradeoff bandwidth and CPU resources. However, there have many jobs other than service chain requests in data center networks or the enterprise networks (e.g. MapReduce). These jobs might be computational-intensive jobs or bandwidth-intensive jobs. These researches only based on fixed job dispatching to deploy the service chain request. But if we only do service chain deployment based on fixed job dispatching, it might not be global optimal because service chain requests and jobs in the network share the network resources. Therefore, we should do service chain deployment with dispatching these jobs together to make network better. Another issue is that many previous researches try to tradeoff bandwidth and CPU resources. However, they did not aware the number of used servers and cause higher electricity cost.

Hence, we should do service chain deployment and job dispatching together to find the global optimal solution. For the aspect of the tradeoff between bandwidth and CPU resources, we should also aware the number of used server so that we can reduce the electricity cost. To achieve these goals, we find out the relation among the bandwidth, CPU resources and the number of used servers first. And then we formulate the service chain deployment and job dispatching problem as an integer linear programming model. Finally, we propose Rack-aware Service-chain deployment and Job dispatching (RSJ) algorithm to solve this problem effectively and efficiently.

doi:10.6342/NTU201703605

Chapter 2

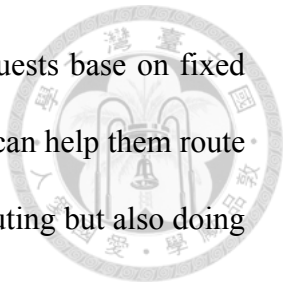
Related Work

In this chapter, we will introduce several related works. We classify previous works into three categories, service chain routing, virtual network functions placement, and service chain deployment.

2.1 Service Chain Routing

SIMPLE [4] and Adaptive service-chain routing [5] are be proposed to find a routing path for service chain request to make sure the traffic would go through the desired sequence of network functions. SIMPLE focus on policy enforcement for efficient traffic steering using SDN under fixed middlebox placement. SIMPLE also takes account the middlebox load balancing problem in the constraint of limited TCAM table space in switches. Adaptive service-chain routing focus on efficient traffic steering using SDN and NFV based on fixed virtual network function placement. This research translates service chain routing problem into a simple shortest path problem so that this problem can be solved by the conventional shortest path algorithms, such as Dijkstra's algorithm. Also, it takes account the latency and model a delay model to estimate. Hence, this work might choose different path for same service chain request based on latency.

doi:10.6342/NTU201703605



However, these works only focus on route the service chain requests base on fixed network function placement but there might have a better placement can help them route these requests much better. We do not only focus on service chain routing but also doing the VNF placement.

2.2 VNF Placement

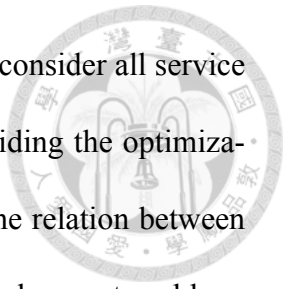
For the single VNF placement, EVNFP [6] provides a placement and allocation solution for single VNF requests. This work does not only consider the resource consumption but also the elasticity. The elasticity means that we can elastically allocate and release resources, but it incurs some costs, such as the time for installing or removing or reassigning the resources. It presents a model for minimizing the operational costs in providing VNF services including the elasticity overhead. And it also proposes a heuristic algorithm to do VNF placement.

Our work focus on service chain deployment because there usually has many request need to traverse an ordered set of VNFs. However, EVNFP did not consider the execution of the service chain, and thus it can not handle the service chain requests well.

2.3 Service Chain Deployment

There have many prior researches focus on service chain deployment. In [7], the service chain placement and chaining problem are formulated as a mixed integer quadratically constrained program. However, this work can only deploy one service chain each time using this model so that it can not consider all service chain requests together to optimize the service chain deployment. In [8], this work formulates the service chain

doi:10.6342/NTU201703605



deployment problem as an integer linear programming model, which consider all service chain requests together. Also, [8] proposed a heuristic approach guiding the optimization model to find a feasible solution efficiently. [9] discover that the relation between bandwidth and CPU resources play a crucial role in service chain deployment problem. Hence, [9] tradeoff the bandwidth and CPU resources by computing a path constraint for each service chain request. And [9] deploy service chain request with this path constraint incrementally so as to maximize the served traffic.

As we mentioned before, there have many jobs in the data center networks or enterprise networks. However, these works do service chain deployment with fixed job dispatching. These works only focus on the remaining available resources to deploy the service chain requests. But we should consider service chain deployment and the job dispatching problem together so as to get a global optimal solution. Moreover, prior works only focus on how to proper tradeoff the bandwidth and CPU resources but did not aware the number of used servers and cause the higher electricity cost. Therefore, our work would do service chain deployment and job dispatching together. We also discover the relation among the bandwidth, CPU resources and number of used servers, and then we tradeoff these competitive resources.

doi:10.6342/NTU201703605

Chapter 3

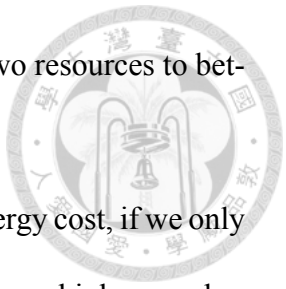
Design

In this chapter, we will describe our goal and design. First, we will show the relation among bandwidth, CPU resources and the number of used servers. Second, we will briefly describe our goal and design. Third, we model the service chain deployment and job dispatching problem as an integer linear programming model. Finally, we will introduce the proposed heuristic algorithm to solve this problem effectively and efficiently.

3.1 Relation between Multiple Resources

The bandwidth resources are a very precious resource in data center network. Hence, we need to minimize the used bandwidth. However, if we only minimize the used bandwidth, then we might use more CPU resources so that we might not have sufficient CPU resources to serve more other requests as much as possible. There has a competition between bandwidth and CPU resources. When the flow traverses shortest path so as to save the bandwidth, the opportunity of the desired VNF which already be installed on this path is not high. Therefore, there might need to install a new VNF on VM to serve this flow and might cause we use more CPU resource. On the other hand, if there has limited CPU resources, then we should route the flow go a longer path to reuse the VNF and might

doi:10.6342/NTU201703605



cause we use more bandwidth. Therefore, we need to tradeoff there two resources to better utilize and serve more requests.

The electricity cost is also a large cost in the data center. For the energy cost, if we only focus on how to tradeoff bandwidth and CPU resource, then we might use higher number of servers and might casue higher electricity cost. Hence, we also need to improve server utilization to make fewer working servers so that we can save the power and reduce the electricity cost.

The question is what is the relation among bandwidth, CPU resource, and the number of used servers? We do simulation to find out the relation. First, we generate several service chain demands and each time we use the optimization model to minimize the total used bandwidth with constraining the number of used servers of these demands. The result shows on Figure 3.1. Figure 3.1 shows that the total used bandwidth is higher when we use fewer servers. With using more servers, the total used bandwidth becomes lower. At the same time, we can see the number of VMs also become higher. From figure 3.1, we find that there has competition between bandwidth and number of used servers and it also prove that the bandwidth and CPU resources are actually two competitive resources. The fact is that when we limit the number of used servers, we also limit the CPU resources. Hence, when we tradeoff bandwidth and number of used servers, we also achieve tradeoff bandwidth and CPU resources. Moreover, we also can achieve reduce the electricity cost.

doi:10.6342/NTU201703605

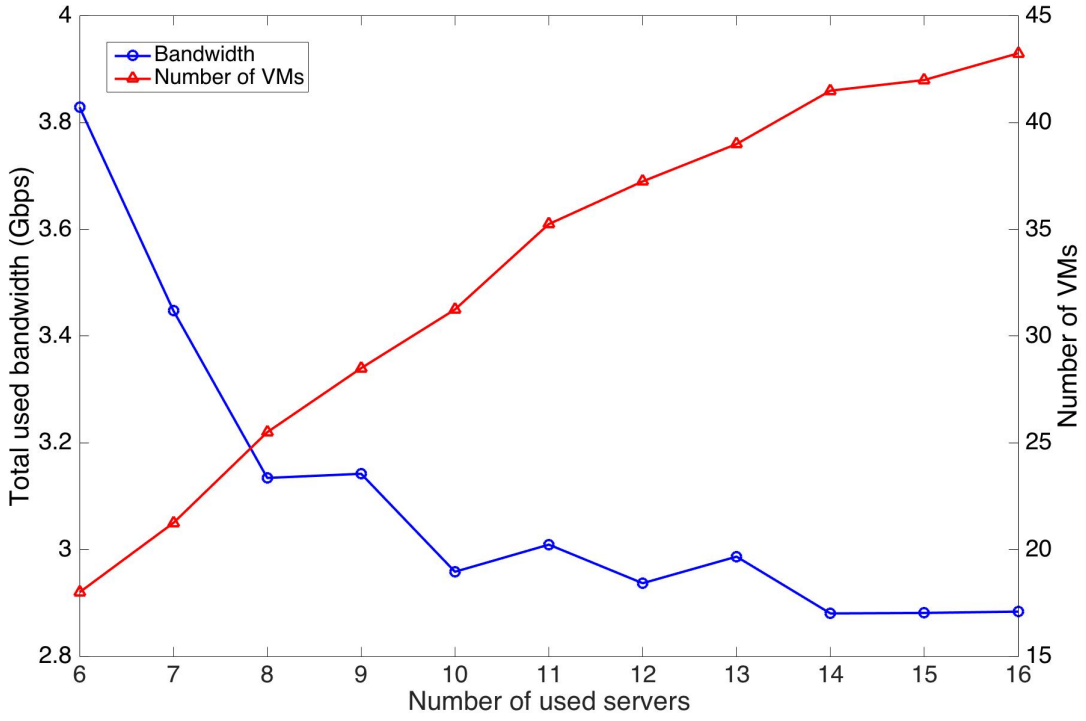


Figure 3.1: Competition between bandwidth and number of used servers

3.2 Overview

We have two goals in this paper. The first one is we need to do service chain deployment and job dispatching together. If we only deploy service chain and do not care the other jobs in the network, then we might not have a global optimal solution. As we mentioned before, we need to tradeoff two competitive resources bandwidth and CPU so as to serve more requests. Also, only tradeoff bandwidth and CPU is not enough, we need to reduce the number of used servers as much as possible so as to save the electricity cost. We already know that bandwidth and the number of used servers are competitive resources so that we need to tradeoff these two resources to better utilize. After we find out the relation among bandwidth, CPU resources and the number of used servers, we know that we only need to tradeoff bandwidth and number of used servers so that we can also achieve tradeoff bandwidth and CPU resources. Hence, the second goal is to minimize the bandwidth

doi:10.6342/NTU201703605

and the number of used servers. First, we formulate the service chain deployment and job dispatching problem as an integer linear programming model. However, this optimization model is very difficult to solve in a reasonable time. Therefore, we proposed a Rack-aware Service-chain deployment and Job dispatching (RSJ) algorithm to find the solution of this problem effectively and efficiently.

3.3 Problem Formulation

3.3.1 Problem Definition

We model a network as an directed graph $G = (V, E)$. For each edge $e = (u, v) \in E$ and $v \in V$ in the graph, $C_e = C(u, v)$ is the link capacity and C_v is the server CPU capacity. V is the vertex set and we also define $V = V_s \cup V_h$, where V_s is the set of switches and V_h is the set of servers. The CPU capacity of switch is 0 and the CPU capacity of server is greater than 0. The virtual network function set is $F = \{f_1, f_2, ...\}$. $D = D_S \cup D_J$ denotes the set of demands, and D_S denotes the set of service chain demands and D_J denotes the set of general data center job which does not have service chain request. For each demand $d \in D$, the source of the demand is represented by src_d and the destination is represented by dst_d. $S_d = (s_{d,1}, s_{d,2}, ...)$ is the service chain of the demand d and $|S_d|$ denotes number of services or virtual network functions in the service chain. For ease of modeling the problem, we set src_d, dst_d to be $s_{d,0}, s_{d,|S_d|+1}$ respectively and $s_{d,0}, s_{d,|S_d|+1} \notin S$. The traffic rate of the demand d is R_d. Also, the demand d would consume CPU cost at destination server is represented by dst_cpu_d. We define the CPU cost of VNF f with traffic rate R is $L(f, R)$ and each VM m has the limited CPU capacity $C_m(f)$. Moreover, we have several assumptions. We assume each VM can run at most one

doi:10.6342/NTU201703605

VNF $f \in F$. We also assume that these has not same service request in a service chain demands. And the same service request in different service chain demands can share a VM which host the service if the VM still has sufficient capacity.

The notations used in the formulation are sumarized in table 3.1.

Table 3.1: Notation used in our model

Notation	Description
G	Network topology as a directed graph
E	Set of all links
V	Set of all nodes
V_s	Set of all switch nodes
V_h	Set of all server nodes
C_e	Link capacity
C_v	CPU capacity of node v
F	Set of virtual network functions
D	Set of demands include SFC demands and data center jobs
src_d	Source of demand d
dst_d	Destination of demand d
S_d	Service chain of demand d
R	Traffic rate
dst_cpu_d	The CPU cost of demand d at the destination server
$L(f, R)$	The CPU cost of VNF f with traffic rate R
$C_m(f)$	The VM capacity for VNF f

doi:10.6342/NTU201703605

We have several variables in our model. The variable $m^I_{f,v} \in \mathbb{N}$ is the number of VNF f should install on server v. And the variable $m^V_{d,s_i,v} \in \{0,1\}$ indicates whether the service s_i of demand d should map to server v or not, that is, if $m^V_{d,s_i,v} = 1$, we would route demand d go through server v and the service s_i would be served in server v. Note that s_0 and $s_{|S_d|+1}$ are source and destination of demand d and also use variables $m^V_{d,s_0,v}$ and $m^V_{d,s_{d,|S_d|+1},v}$ to decide the source and destination, so we need to setup $s_{d,0}, s_{d,|S_d|+1}$ at the first. The variable $m^E_{d,s_i,s_{i+1},u,v}$ indicates whether the link (u,v) is the part of path between two consecutive services (s_i, s_{i+1}) in demand d. Because service chain is that the request need to go through several service functions in a specific order, so each time we consider two consecutive services and we need to find a path between these two service functions. And the variable m^S_v indicates whether the server is used or not.

The variables used in our integer linear programming model is list in table 3.2.

Table 3.2: Variables used in our model

Variable	Description
$m^I_{f,v}$	Number of VNF f install on server v
$m^V_{d,s_i,v}$	Indicator variable for service s_i of demand d map to server v
$m^E_{d,s_i,s_{i+1},u,v}$	Indicator variable for the link (u,v) is a part of path between S_i, s_{i+1}
m^S_v	Indicator variable for the server v is used or not

doi:10.6342/NTU201703605

3.3.2 Integer Linear Programming Model

We formulate the service chains deployment and job dispatching problem as an integer linear programming model for placing virtual network functions and routing the service chain requests and also dispatching data center jobs together to minimize the cost of network resource bandwidth and the number of used servers when the requirements of demands could still be satisfied.

The optimization problem of service chain deployment and job dispatching is formulated as following:

Mimimize :

$$\alpha * \frac{usedBW}{\sum_{(u,v)\in E} C(u,v)} + (1-\alpha) * \frac{usedServer}{|V_s|} \tag{3.1}$$

subject to:

$$\forall v \in V_h, \quad \sum_{f\in F} m^I_{f,v} * C_m(f) + \sum_{d\in D} m^V_{d,s_{d,|S_d|+1},v} * dst\_cpu_d \leq C_v \tag{3.2}$$

$\forall v \in V_h,\ f \in F$:

$$(m^I_{f,v} - 1) * C_m(f) < \sum_{d\in D, s_i \in S_d : s_i = f} m^V_{d,s_i,v} * L(s_i, R_d) \leq m^I_{f,v} * C_m(f) \tag{3.3}$$

$$\forall (u,v) \in E, \quad \sum_{d\in D} \sum_{i=0}^{|S_d|} m^E_{d,s_i,s_{i+1},u,v} * R_d \leq C(u,v) \tag{3.4}$$

doi:10.6342/NTU201703605

$$\forall d \in D,\ v \in V,\ s_{d,i}, where\ 0 \leq i \leq |S_d| :$$

$$\sum_{\forall u \in V:(v,u) \in E} m^E_{d,s_i,s_{i+1},v,u} - \sum_{\forall u \in V:(u,v) \in E} m^E_{d,s_i,s_{i+1},u,v} = m^V_{d,s_i,v} - m^V_{d,s_{i+1},v} \tag{3.5}$$

$$\forall d \in D,\ s_i \in S_d \cup \{s_{d,|S_d|+1}\}, \quad \sum_{v \in V_h} m^V_{d,s_i,v} = 1 \tag{3.6}$$

$$\forall v \in V_h,\ d \in D,\ s_i \in S_d \cup \{s_{d,0},\ s_{d,|S_d|+1}\}, \quad m^V_{d,s_i,v} \leq m^S_v \tag{3.7}$$

$$\forall d \in D,\ \forall v \in V, \quad m^V_{d,s_{d,0},v} = \begin{cases} 0, v \neq src_d \\ 1, v = src_d \end{cases} \tag{3.8}$$

$$\forall d \in D,\ dst_d \neq None,\ \forall v \in V, \quad m^V_{d,s_{d,|S_d|+1},v} = \begin{cases} 0, v \neq dst_d \\ 1, v = dst_d \end{cases} \tag{3.9}$$

$$\forall d \in D,\ dst_d = None,\ \forall v \in V_s, \quad m^V_{d,s_{d,|S_d|+1},v} = 0 \tag{3.10}$$

$$\forall d \in D,\ dst_d = None,\ v = src_d, \quad m^V_{d,s_{d,|S_d|+1},v} = 0 \tag{3.11}$$

doi:10.6342/NTU201703605

The objective function 3.1 of our model aims for minimizing a weighted combination of network resources bandwidth and number of used servers. The equation of the used bandwidth and the number of used servers is formulated as 3.12 and 3.13.

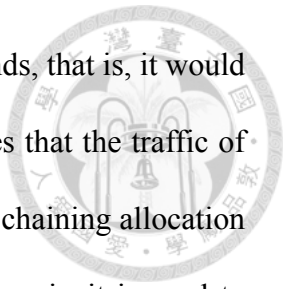
$$usedBW = \sum_{e \in E} \sum_{d \in D} \sum_{i=0}^{|S_d|} m^E_{d,s_i,s_{i+1},u,v} * R_d \tag{3.12}$$

$$usedServer = \sum_{v \in V_h} m^S_v \tag{3.13}$$

We use a weighted factor $\alpha \in [0, 1]$ for network administrator to decide how to tradeoff bandwidth and number of used servers. However, even network administrator can decide α to tradeoff these two resources, it is still difficult to intuitively quantify our objective function only using how many Gbps of bandwidth and number of used servers because we do not know how important between 1 Gbps and 1 server. Hence, for ease of understanding how to choose α, we use the combination of bandwidth utilization and the percentage of used servers. When $\alpha = 0$, the objective function is to minimize the number of used servers in data center. Instead, if $\alpha = 1$, the objective function would become minimizing network bandwidth. If we want to save bandwidth, then we should route the demands to the nearest available server which may not be used before. On the other hand, if we want to use fewer servers to save the energy cost, then we should make the demands served by the server which is already used before. It also means that demands should reuse VM as much as possible so that it would traverse a longer path and use more bandwidth.

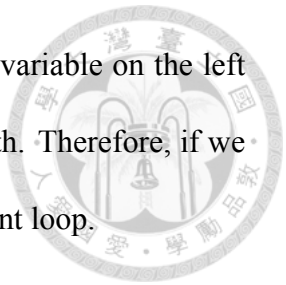
Constraint 3.2 limits the CPU cost of all VMs and all the jobs on the server v not greater than the server CPU capacity. Constraint 3.3 ensures that the CPU cost of the demands served by the VM which is for f on the server v does not exceed the VM capacity. And

doi:10.6342/NTU201703605



it also ensures that the number of VM is minimum to serve the demands, that is, it would not open additional VM which would not use. Constraint 3.4 ensures that the traffic of demands on the link will less than the link capacity. Constraint 3.5 is chaining allocation and routing demands constraint. For each consecutive two services pair, it is used to decide both which link belongs to the path and which server should install VM and serve the service. Left side of constraint 3.5 is the difference of the outdegree of d of network node v and the indegree of d of network node v and then we would get a value 1, 0 or -1, which implies the network node v is the source or on the desired path or the destination. And the right side of constraint 3.5 determines the source or the destination of the path. By this constraint, we make sure there should have a path between these to consecutive services pair. For example, consider if we want to setup service pair (s_i, s_{i+1}) and also want to find a path between (s_i, s_{i+1}). There will have several cases. If $m^V_{d,s_i,v} = 1$ and $m^V_{d,s_{i+1},v} = 0$, then the left side of the equation must be 1, that means the network node v will serve s_i and also be the source of path of (s_i, s_{i+1}). If $m^V_{d,s_i,v} = 0$ and $m^V_{d,s_{i+1},v} = 1$, the left side of equation must be 1, which means the network node v will serve s_{i+1} and also be the destination of path of (s_i, s_{i+1}). If $m^V_{d,s_i,v} = 0$ and $m^V_{d,s_{i+1},v} = 0$, then the right side of equation must be 0 and both of two variables on the left side of equation must be 1 because there already setup a source node for a demand so that if the next service is not assigned to the source node then there must exist a network node that the next service should traverse to and lead to a outdegree. Therefore, there has a network node that does not serve the service and has a indegree so that the outdegree of this node must be 1. The last case is $m^V_{d,s_i,v} = 1$ and $m^V_{d,s_{i+1},v} = 1$, this means these two consecutive services pair are assigned to the network node v and both of link indicator variable on the left side will be 0 because one of our goal is minimize bandwidth cost. But when $\alpha = 0$, that means we

doi:10.6342/NTU201703605



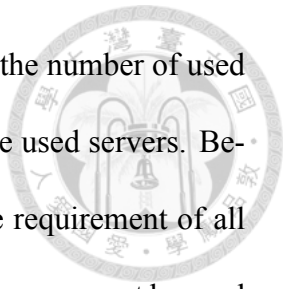
do not need to minimize bandwidth cost, then both of link indicator variable on the left side may be 1. This situation would cause a loop and waste bandwidth. Therefore, if we want to set $\alpha = 0$, then we need to add a new constraint 3.14 to prevent loop.

$$\forall d \in D, v \in V, s_{d,i}, where\ 0 \leq i \leq |S_d| :$$

$$\sum_{\forall u \in V:(v,u) \in E} m^E_{d,s_i,s_{i+1},v,u} + \sum_{\forall u \in V:(u,v) \in E} m^E_{d,s_i,s_{i+1},u,v} \leq 2 \tag{3.14}$$

Constraint 3.6 ensures that each service will be assigned to exact one server. And constraint 3.6 also guarantees that there must assign a server for unknown destination demand. Constraint 3.7 is used to indicate whether the server is used or not. This constraint help us to count number of used servers. If there has a service be assigned to the server or the source or destination of the demand on the server, then the server indicator variable must be 1. Constraint 3.8 and Constraint 3.9 setup the source and the known destination of demands. Constraint 3.10 makes sure the unknown destination of demand will not be assigned to switch nodes. If we do not add this constraint in our model, our model still can work due to we set the CPU capacity of switch nodes to 0. But we can intuitively know we should not dispatch the jobs to a switch but a server. By doing so, it can help this model reduce finding unnecessary solutions. Constraint 3.11 guarantees the unknown destination demands would not dispatch to the source. Because the jobs that we want to dispatch are like MapReduce jobs or network applications, this means these jobs may need some resources or some goal want to finish on other servers but not on the original server. Hence, we need to avoid dispatch these jobs to their source servers.

doi:10.6342/NTU201703605



In addition, although we limit the CPU resource with minimizing the number of used servers, it also means we could install VMs as much as possible in the used servers. Because we also dispatch the data center jobs, so we already satisfy the requirement of all requests and jobs. Hence, it means there might have additional CPU resources not be used so that we might install more VMs on used servers to save bandwidth.

We model the service chain deployment and job dispatching problem as an integer linear programming model. We can use this model to optimize the utilized resources in the data center network. However, our model is very difficult to solve and get a good solution in a reasonable time. Because our model need to examine all possible servers whether install VNFs or not and how much VNFs and decide each service of each service chain requests whether map to this server to be served and find the desired paths between these servers for all service chain requests and all jobs. Actually, this problem could be reduced to edge-disjoint paths(EDP) problem [10], which is a NP-hard problem. In [9], this paper already proved that the path selection and VNF placement problem is a NP-hard problem. We would extend this provement to our problem, which is not only take account the path selection and VNF placement but also consider job dispatching. Given a graph G with every link capacity is R in the EDP problem, there has three nodes pair(o_1, t_1), (o_2, t_2), (o_3, t_3) and the EDP problem wants to find three dis-joint paths to connect o_i to t_i for i in $\{1, 2, 3\}$. We construct a graph $G' = G$ in our problem. We add a node r, which has the only one VM in the G', to the graph G' and add the directed links (t_1, r), (r, o_2) with link capacity R. If we have a service chain demand with traffic rate R want to go through the only VM in node r and a job with traffic rate R from o_3 to t_3, then there has a feasible solution in our problem if and only if the EDP problem has a feasible solution. Therefore, we reduce our problem to EDP problem and prove that it is NP-hard to find the

doi:10.6342/NTU201703605

feasible solution of our problem.

Since it is difficult to find the solution for all demands as a whole. Instead, we could find the solution for a demand each time. Therefore, we propose a Rack-aware Service-chain deployment and Job dispatching (RSJ) algorithm to find the solution effectively and efficiently.

doi:10.6342/NTU201703605

3.4 Rack-Aware Service-chain Deployment and Job Dispatching

Our work proposed Rack-aware Service-chain deployment and Job dispatching (RSJ) algorithm to find the solution of the service chain deployment and job dispatching problem effectively and efficiently. In the section, we describe our proposed RSJ algorithm which is based on choosing one or more proper racks to deploy whole service chain or part of service chain each time for a demand and also dispatch.

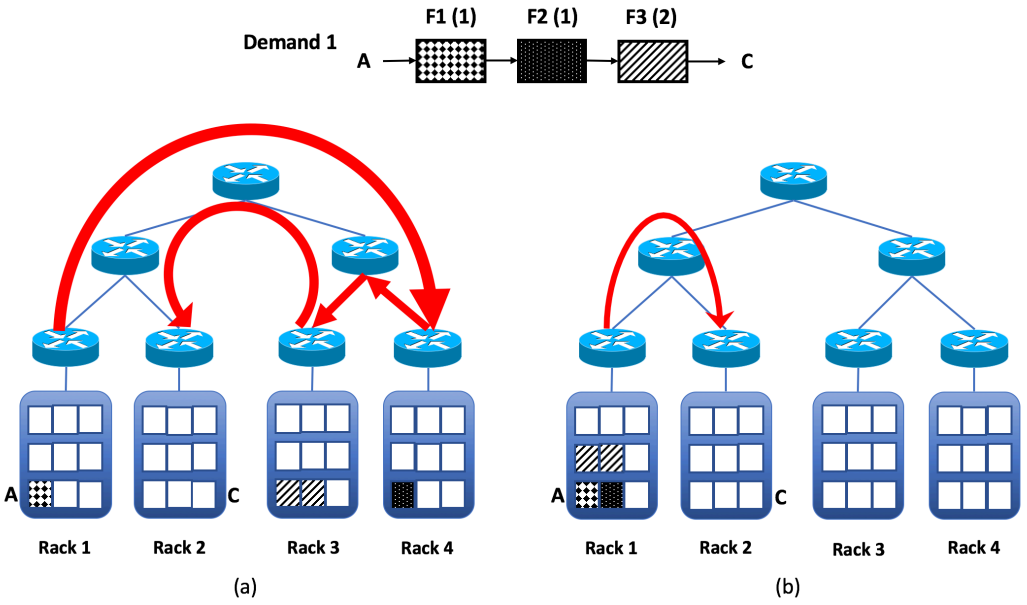
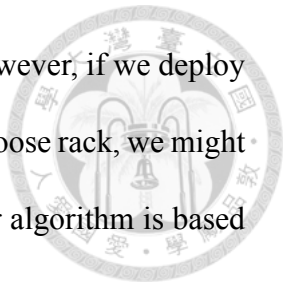


Figure 3.2: Example of service chain deployment

Nowaday, the servers are packed to a rack in the data center. If the traffic traverses across racks, it might use the precious core level bandwidth. For example, figure 3.2 shows two examples of service chain deployment for a demand which need to traverse VNF f1, f2, f3 in order from server A to server C. Figure 3.2(a) separate the service chain across rack and figure 3.2(b) makes the service chain stay in a rack. We can see the figure 3.2(a) use lots of bandwidth compare to figure 3.2(b). Moreover, if we separate the service chain to several racks, there might not only increase the used bandwidth but also have possibility

doi:10.6342/NTU201703605



of higher number of used servers depend on how to choose racks. However, if we deploy service chain in a rack and also aware the number of used servers to choose rack, we might reduce the bandwidth and used servers. Therefore, the concept of our algorithm is based on how to choose an appropriate rack to deploy service chain.

How to choose rack for a service chain demand is very important in our algorithm. First, we should recall the objective function 3.1 in our ILP model. We can see there has a weighted factor α in our objective function. If the network administrator does not care about the bandwidth cost, then the weighted factor α would be small. It means we should more care about the used servers while choosing rack. Therefore, we need to depend on α to choose rack. Our goal is to minimize the objective function. We could use the objective function as our score function to choose rack which affects the objective value is smallest, that means we prefer choosing the smaller score of rack has the higher priority. However, we do not know the total used bandwidth and the total number of used servers. We need to estimate these two metrics. Suppose the rack of source of demand is r_i and we pick rack r_j and try to calculate the score. We should not only calculate the path between rack r_i and r_j but also need to add the path between rack r_j and the rack of destination of demand to estimate the used bandwidth of this service chain demand. The expectation used bandwidth is formulated as equation 3.15. For the number of used servers, we need to calculate the additional servers which would open after we deploy service chain in the rack. If the rack has not sufficient CPU resources for the service chain demand, and although the exceeded part of service chain could not open new server on this rack, it still would cause we open a new server on other racks. Because we want the rack can place the whole service chain, so if the rack has not sufficient CPU resources, then we should give this rack higher scores. Therefore, the expectation of the number of used servers

doi:10.6342/NTU201703605

is equation 3.16, which we predict the number of servers would be open after deploying the service chain S_d in the rack r_j. And then we get the rack score function which is formulated as equation 3.17.

$$expectBW = R_d * (hop(r_i, r_j) + hop(r_j, r_dst)) \tag{3.15}$$

$$expectServer = predictServer(S_d, r_j) \tag{3.16}$$

$$Rack\ Score = \alpha * \frac{expectBW}{TotalBW} + (1 - \alpha) * \frac{expectServer}{TotalServer} \tag{3.17}$$

Algorithm 1 presents the pseudo code of RSJ algorithm. The concept of RSJ algorithm is based on rack score 3.17 find one or more racks to deploy service chain for a demands each time. For job we can dispatch, we choose the destination server based on finding the minimal shortest path on opened servers first. Finally, we compute the shortest path between these servers, which are this demand should traverse to and use the required VNF.

In RSJ algorithm, the first thing is to decide the execution order (line 1). We should choose the higher cost demand first because if we deploy the higher cost demand later, then there might no rack has sufficient CPU resource to serve whole service chain and cause this service chain should separate to several racks and then affect our objective value to be worse. Hence, we would like to process the higher cost demand first. We sort the demand set by $R_d * |S_d| * CPU\_COST_{aver}$. Longer service chain might use more bandwidth an CPU resources and larger traffic size also means the bandwidth cost might higher. Also, we calculate the average CPU cost in service chain and required CPU cost in destination server. This calculation implies this demand would use how much resources. In addition,

doi:10.6342/NTU201703605

Algorithm 1: Rack-aware Service chain deployment and Job dispatching Algorithm

Input: $G = (V, E), D$

Output: Variables $m^I_{f,v}, m^V_{d,S_i,v}, m^E_{d,S_i,S_{i+1},u,v}$

Sort D by $R_d * |S_d| * CPU\_COST_{aver}$ with descending order and then move demands with unknown dst to the end;

$m^I,\ m^V,\ m^E \leftarrow \emptyset$;

for $d \in D$ **do**

  $m^V_d \leftarrow \emptyset,\ excludeSet \leftarrow \emptyset$;

  $dstPi \leftarrow |S_d|$

  for $r \in RackSet$ **do** // calculate the rack scores

    $M \leftarrow M \cup \{(r, \alpha * (expectBW)/totalBW + (1-\alpha) * (expectServer(S_d, r))/TotalServer)\}$;

  $preRack,\ prePi \leftarrow rack(src_d),\ 0$;

  while $M \neq \emptyset$ **do**

    $r,\ minM \leftarrow pop(min(M))$;

    $isDst = False$;

    if $r == rack(dst_d)$ **then** // set flag to put max subset SFC to dst rack

      $isDst = True$;

    if $placeSFCinRack(G, R_d, S_d, prePi, r, m^V_d, dstPi)$ **then**

      break;

    else

      $excludeSet \leftarrow excludeSet \cup \{r\}$;

      $Pi \leftarrow placeMaxSubsetSFCinRack(G, R_d, S, prePi, dstPi, r, m^V_d, isDst)$;

      if $Pi \neq -1$ **then**

        if $isDst == True$ **then**

          $dstPi = Pi$;

        else

          Update M; // use remaining part of SFC

        $preRack = r,\ prePi = Pi$;

  else

    Reject d;

    continue;

  Sort m^V_d by move servers in the destination rack to the end;

  if $dst_d == None$ **then**

    $dst_d \leftarrow$ find destination in opened servers with the minimal shortest path , if can not find, then find in the remaining servers;

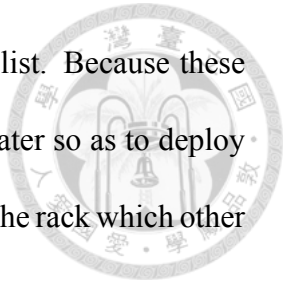
    $m^V_d \leftarrow m^V_d \cup \{dst_d\}$;

  $m^E_d \leftarrow m^V_d$ records demand d should traverse which servers to ensure the requirement. We find the shortest path between these servers;

$m^I \leftarrow$ When we find rack and place service chain in rack, we also record the request VNF would be installed on which servers and store result in G, we take the result from G;

return $m^I,\ m^V,\ m^E$;

doi:10.6342/NTU201703605

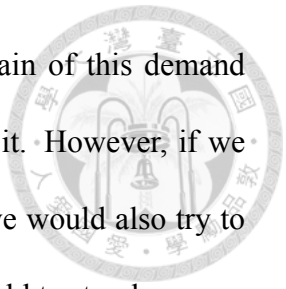


we move the unknown destination demands to the end of the sorted list. Because these demands mean we can dispatch it, we should deploy these demands later so as to deploy these demands on proper racks, instead of deploying these demands to the rack which other rack might not be chosen by other demands and might cause we open unnecessary servers. Therefore, after we move this kind of demands to the last, we could deploy service chain and dispatch these demand on opened servers to reduce the bandwidth and CPU cost.

Before we start our algorithm, we could allocate some resources first since there have some resources which must use for demands. In the calculation, we would allocate the CPU resources to the source and the already known destination servers of demands at the first, and then we would know which servers must open and help us choosing right rack to improve the rack utilization. Also, by doing so, we would not allocate too much CPU resources of a server for other demands and cause some demands, whose destination is that server, could not be served. We also allocate the bandwidth out from source server and into the destination for these demand in the calculation. If we find an allocation could not be done, then we reject this demand.

After sorting the demands and the prepared works, we pop a demand each time from the sorted list. And then we calculate the rack score for all racks and choose the smallest one to deploy (line 3-27). We would check that the bandwidth into and out from this rack is sufficient if it needs. And then we try deploy the whole service chain to this rack. Algorithm 2 details how to deploy the service chain in the rack and we will describe later. Algorithm 2 would try to find a feasible solution. If we can find solution of deploying whole service chain into chosen rack, then we finish finding the rack (line 14-15). But if this rack could not deploy this service chain, then we deploy the max consecutive subset of this service chain from the front of chain to the chosen rack and then recalculate the

doi:10.6342/NTU201703605



rack score excluding the chosen racks for the remaining service chain of this demand until there has no rack could choose (line 9-24), then we will reject it. However, if we choose a rack which has destination server when still finding rack, we would also try to place whole remaining part of service chain. But if it can not, we would try to place max consecutive subset of chain from the end of service chain to this rack. By doing so, we could make sure the rack, which has destination server, would be the last rack be traversed for required service of demand in specific order. Moreover, if there has same rack score of racks. For first α% demands, we would prefer the rack with lower $expectBW$. If there still have racks which has same $expectBW$, then we choose the rack with larger number of reuse VMs. And for the remaining demands, we would prefer the rack with larger number of reuse VMs. If there still have racks which has same value, then we choose rack with lower $expectBW$. Because when the rack scores are same, it means the influence to objective value is same. Therefore, how to choose proper rack is an opportunity for future demands need. However, we know the demands ahead in sorted list means the required resources is larger. The kind of demands might cause larger used bandwidth. Therefore, we should deploy them more prefer lower used bandwidth and choosing install new VM to save bandwidth. At the same time, this behavior also make an opportunity for future demands to reuse VMs. And we also use α, which is the bandwidth importance for network administrator, to achieve the goal which network administrator wants to tradeoff.

After we have done choosing racks to deploy service chain, when algorithm 2 find a feasible solution of deploying whole or part of service chain, that means the service chain can be deployed into the chosen rack, we would allocate CPU resources and install the VNFs on the servers based on the solution. Note that, because we might choose rack which has destination server first and then the other rack. Hence, we should move the

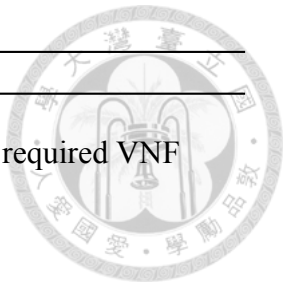
doi:10.6342/NTU201703605

server, which is in the rack which has destination server, to the end of m_d^V and we will get the right order of the server we want to route demand traverse to (line 28). If the demand is that the job we can dispatch, then we choose a server from opened servers by selecting server with the minimal path from the server, which should traverse to for last service of demand, to each opened server (line 29-31). Also, with this solution, we would know that each service in service chain of the demand should go through which server to use the required VNF and even the destination server we dispatch the demand to. Therefore, we compute the shortest path between these servers to route the demand (line 32). After we deployed all demands, then we take the information, which are which VNF should be installed to which server and should be opened how many VMs for the VNF, from the graph $G = (V, E)$ (line 33). We store each server would install which VNF and should open how many VMs for the VNF when each time we choose a rack and deploy whole or max subset of service chain (line 14,18). Finally, we finish deploy service chain and do job dispatching.

Next, we will describe how we deploy whole or part of service chain in a rack. Algorithm 2 presents the pseudo code of this algorithm. Suppose we want to deploy a service chain in to a rack. Each time we choose a server in the rack for one service to traverse to use. We start the procedure from the first service in service chain to the last one. In algorithm 2, there has two main parts, the first one is trying to find the server which has the required VNF which could be reused, the second part is that there has no server has the required VNF which could be reused so that we need to choose a server to install VNF and has the the minimal influence on objective value.

Since we already choose appropriate racks to deploy service chain based on α, and the deployment in the rack should more focus on number of used servers, this is because

doi:10.6342/NTU201703605



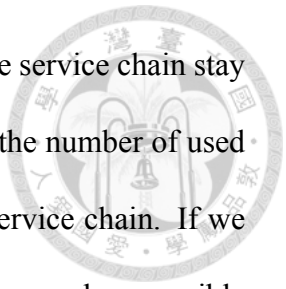
Algorithm 2: Deploy Service-chain in Rack

Input: $G = (V, E), rack, d, S$

Output: sol :record demand d should go which server to find the required VNF

```
1  i ← 0, sol ← ∅;
2  vlist ← rack;
3  while i < |S| do
4      s ← S[i];
5      Sort vlist with 1. if s can also reuse in previous server 2. number of the
         consecutive reusable VM from s in descending order 3. choose source 4. put
         destination to last;
6      for v ∈ vlist do                          // choose server which can reuse s
7          if sol ≠ ∅ and α == 1 then
8              continue;
9          if s can be reused in v then
10             if v is previous chosen server then
11                 Add s to set of v in sol;
12             else
13                 if in-link and out-link capacity of v is enouth then
14                     Update in-link and out-link capacity of v;
15                     sol ← sol ∪ {(v, {s})};
16                 else
17                     continue;
18             break;
19     else                          // no server can reuse s, re-choose server to install s
20         Sort vlist with 1. ServerScore in ascending order 2. put destination to last
             3. remainingLoad in descending order;
21         for v ∈ vlist do
22             sLoad ← calculate the expected CPU cost on v;
23             if remainingLoad_v < sLoad then
24                 continue;
25             if v is previous chosen server then
26                 Add s to set of v in sol;
27                 Update remainingLoad_v and VM load in v
28             else
29                 if in-link and out-link capacity of v is enouth then
30                     Update in-link and out-link capacity of v;
31                     decision ← sol ∪ {(v, {s})};
32                     Update the remainingLoad_v;
33                 else
34                     continue;
35         else
36             return ∅;
37     i+ = 1;
38 return sol;
```

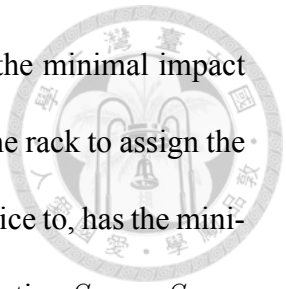

doi:10.6342/NTU201703605



the bandwidth cost between servers in rack is small and if we make the service chain stay in a server so that we could not only save bandwidth but also reduce the number of used servers. However, the question is how we choose server to deploy service chain. If we only minimize the number of used servers, we might try to reuse VM as much as possible and cause that we might route demand out from a server to reuse a VM from another server and back to the server to reuse another VM. This situation might cause use more bandwidth but minimize the CPU resources so that it can serve more demands. On the other hand, if we want to minimize the bandwidth, we should not try to reuse VM as much as possible, instead, we should make services stay in same server to reduce path hops. However, as we mentioned before, the bandwidth cost in rack would not use too much to affect the objective value too much. Therefore, minimizing number of used servers and trying to reduce path hop is our strategy.

In the first part (line 6-18), we need to decide the servers order for assigning the service and the order we store in $vlist$ first. Then we sort $vlist$ based on a) if this service could find a reusable VNF in previous chosen server b) number of consecutive services, which could find reusable VNFs in the server, from the service we want to assign to server at this time c) source server d) put destination server to the last (line 5). Note that, the sorting is first use (a) to sort, if there still has same condition, then do (b) and so on. After we do this sorting on $vlist$, we would get the order we want. Then, we choose server following this order and we would check whether the service can find reusable VNF in the server or not. If it can, then we done assignment of the service (line 6-18). If not, we would try all servers in $vlist$ until no server can assign to. Moreover, when $\alpha = 1$, which means we do not care the number of used servers, so if it is not first service, we would not try to reuse VM and leave the assignment to the second part (line 7-8).

doi:10.6342/NTU201703605



In the second part (line 19-36), it would choose server based on the minimal impact on objective function. We also need to decide the order of servers in the rack to assign the service. However, we need to find out which server, we assign the service to, has the minimal impact on the objective value. Therefore, we also use the score function $ServerScore$ as same as rack score equation 3.17. For $expectBW$, we calculate the expectation used bandwidth for each server. If we choose the server which is previous chosen server, then the $expectBW$ would be 0. On the other hand, if we choose another server, then the $expectBW$ would be $2R$ because we need to route to the top-of-rack switch and then route to the chosen server. For $expectSever$, if the chosen server already opened VM of the required VNF but has not sufficient capacity to be reused, the chosen server should open new VM for installing the required VNF because there has no reusable required VNF of the service in the rack after the first part. However, although we still need open new VM, there might still has some remaining capacity could use so that we might have more capacity for future need if we choose the lower loading of VNF. Hence, if the chosen server has the required VNF, then the $expectServer$ would be $(Load\ of\ VNF\ f\ on\ v)/C_m(f)$. If not, then $expectServer$ would be $C_m(f)/C_v$ if server v already opened, otherwise, $expectServer$ would be 1. Therefore, we would sort $vlist$ by a) score in ascending order b) put the destination server to the end of list c) remaining server capacity in descending order (line 20). After sorting, then we calculate the CPU cost of the service in the chosen server (line 22). If the remaining server capacity is not sufficient to serve the service, then we choose the next server in $vlist$. If it has sufficient capacity to serve, then if the chosen server is the previous chosen server, we assign the service to the chosen server and add to the solution, and then we finish the assignment of this service (line 25-27). Otherwise, we would try all servers in $vlist$ until no server can assign to. If we still could not find a

doi:10.6342/NTU201703605

server for the service, then we could not find a feasible solution and return $\emptyset$.

3.5 Complexity Analysis

In this section, we find out the complexity of RSJ algorithm. Each time we try to place whole or max subset of service chain would use algorithm 2 to find the deployment in a rack. Algorithm 2 would find a server for each service. Hence, the complexity of the algorithm 2 is $O(|S| * |V_r|)$, where V_r denotes the server set in the rack. Therefore, the complexity of placing whole or max subset of services in a rack is $O(|S| * |V_r|)$. RSJ algorithm would find one or more racks to deploy the service chain. Hence, the finding rack procedure would repeat $O(|R|)$, where R denotes the rack set. Therefore, the complexity of finding rack procedure is $O(|R| * |S| * |V_r|)$ .After finishing the finding rack procedure, we would dispatch demand to find a destination for it. The dispatching step would find the minimal shortest path among servers, which means it would repeat $O(|V|)$. And we use the Dijkstra's algorithm to find the shortest path, and complexity of this algorithm is $O(|V|+|E|+|E|\log|E|)$. Then, we get the complexity of the dispatching step is $O(|V| * (|V| + |E| + |E|\log|E|))$ if job ratio is not equal to 0. For the routing step, we would find the shortest path between the chosen servers and each service would assign to one of these servers. Thus, the complexity of routing step is $O(|S| * (|V| + |E| + |E|\log|E|)$. And we would repeat $O(|D|)$. Finally, we could get the complexity of RSJ algorithm is $O(|D| * ((|R| * |S| * |V_r|) + (|S| + |V|) * (|V| + |E| + |E|\log|E|)))$ if job ratio is not equal to 0. If job ratio is equal to 0, that means we do not need to dispatch demands, then the complexity of RSJ algorithm would be $O(|D| * ((|R| * |S| * |V_r|) + |S| * (|V| + |E| + |E|\log|E|)))$.

doi:10.6342/NTU201703605

Chapter 4

Evaluation

In this section, we show how we evaluate our design and the results. First, we list our simulation configurations and all methods we will compare. In our evaluation, we focus on objective value which is a combination of bandwidth utilization and the percentage of used servers. It can show how much resources cost and also implies the tradeoff which network administrator wants. We also show the used bandwidth and number of used servers and we observe the variety. Second, we show that our methods can improve service chain deployment in different weighted factor α. Third, we show the improvement of our methods in different job ratio. Finally, we show the improvement of our methods in different size of demand set and also compare the processing time between optimal method and our proposed algorithm.

4.1 Simulation Setup

4.1.1 Configurations

We do the simulation to show the performance of our design in a data center networks, a 8-ary Fat-tree [11]. For Fat-tree, we set the link capacity between core switch and aggre-

doi:10.6342/NTU201703605

gation switch to 4 Gbps, the link capacity between aggregation switch and edge switch to 2 Gbps and edge switch to server to 1 Gbps. For each server, we set the server capacity to 40000 [9]. For each virtual machine, we set the VM capacity to 10000 [9]. We generate a VNF set F, which has 15 distinct VNFs. For each VNF $f \in F$, we set the CPU cost of VNF f with traffic rate R to $L(f, R) = 10 * R$ and $L(f, R) = 10 * R * ln(R)$ with probability 90% and 10% respectively [9]. We use IBM CPLEX Optimization Studio 12.7 [12] as the integer linear programming solver. All simulations were performed on a computer with Intel Core i7-4790 processor and 16 GB RAM using operating system Ubuntu 16.04.

4.1.2 Traffic Patterns

We use the power law distribution [13] to generate the traffic size with the minimum demand size 10 Mbps [9]. The length of service chain is uniformly chosen from 0 to 8. Each service chain is sequentially constructed by uniformly choosing from VNF set F with length of the service chain. The CPU cost at the destination is chosen from 2000 to 5000 uniformly if the length of service chain is 0 and normal distribution with mean = 1000 and variance = 500 if the length of service chain is greater than 0. Because there might have many computational intensive jobs in data center and these jobs might not need to use VNF, so we give these jobs higher CPU cost. For fixed job dispatching, we make the jobs which we can dispatch use the probability 40-90% would choose the destination outside the rack [14].

doi:10.6342/NTU201703605

4.1.3 Methods

We comapre our proposed algorithm with different methods.

Original Optimal: It uses our optimization model with fixed job dispatching.

RSJ: Our proposed Rack-aware Service-chain deployment and Job dispatching algorithm.

Optimal: It uses our optimization model with job dispatching.

4.2 Impact of Weigthed Factor α

Fig 4.1 shows the objective value with different weighted factor α. We can see that the optimal has the best performance. However, our proposed algorithm is very close to the optimal. It also shows that the optimal and our proposed algorithm are both outperform than the original optimal which is fixed job dispatching. With increasing of α, the objective value decreases in different methods, this is because the bandwidth resources are plentiful so that the bandwidth utilization would more smaller than the percentage of used servers. Therefore, with the growth of α, the objective value would be smaller because the weighted value of the percentage of used servers is smaller relatively. Although there three methods would very close when $\alpha = 1$, however, the optimal actually still best and our proposed algorithm also very close to the optimal. And the original optimal is still worse than the optimal and our proposed algorithm. Figure 4.2 shows the number of used servers in different weighted factor α. The optimal and our proposed algorithm still better than the original optimal. However, when $\alpha = 1$, the optimal and original optimal have the same number of used servers. Because $\alpha = 1$ means we do not care the number of used servers, so it would use more CPU resources to save the used bandwidth and cause use all

doi:10.6342/NTU201703605

of servers. And Our proposed algorithm would not use all of servers due to our deploying service chain method would still try to stay in same server and would cause we would not use all of servers but the number of used server would still higher than $\alpha \neq 1$. Figure 4.3 shows the used bandwidth in different weighted factor α. When $\alpha = 0$, that means we do not care the bandwidth cost, so it would cause these methods use more bandwidth to save the CPU resources and then minimize the number of used servers. However, the optimal can dispatch job so that it might use more bandwidth to minimize the used servers. This is why the use bandwidth is slightly greater than the original optimal. Our algorithm used the minimal bandwidth because we deploy the service chain into rack. With the characteristic of rack, we still can use less bandwidth even though we do not care the used bandwidth.

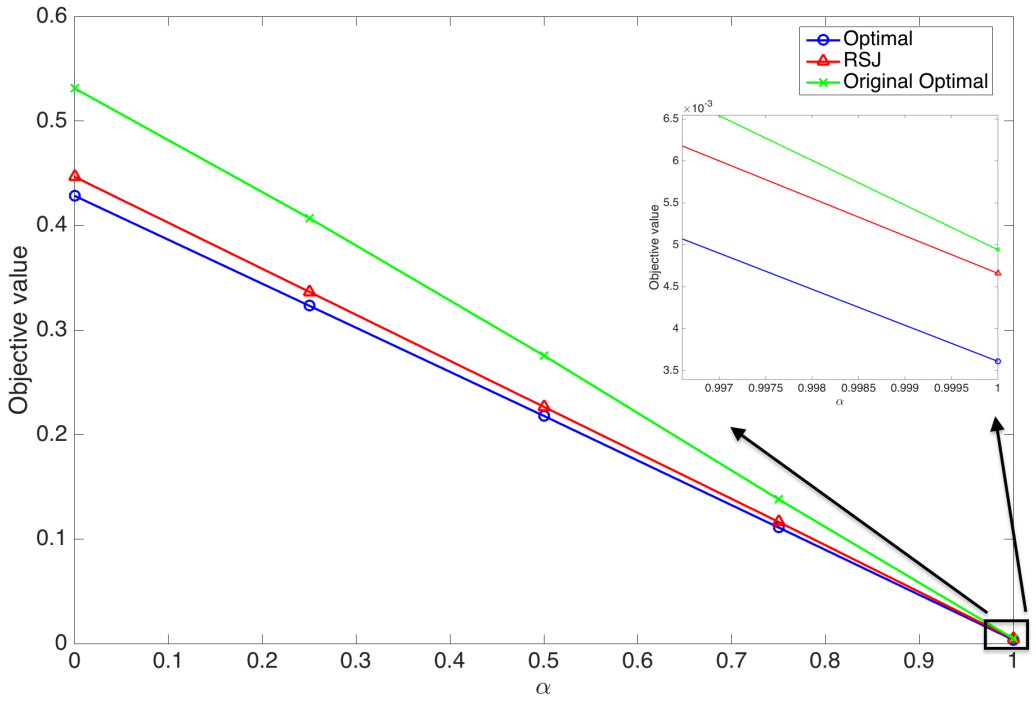


Figure 4.1: Impact of weighted factor α (Objective value)

doi:10.6342/NTU201703605

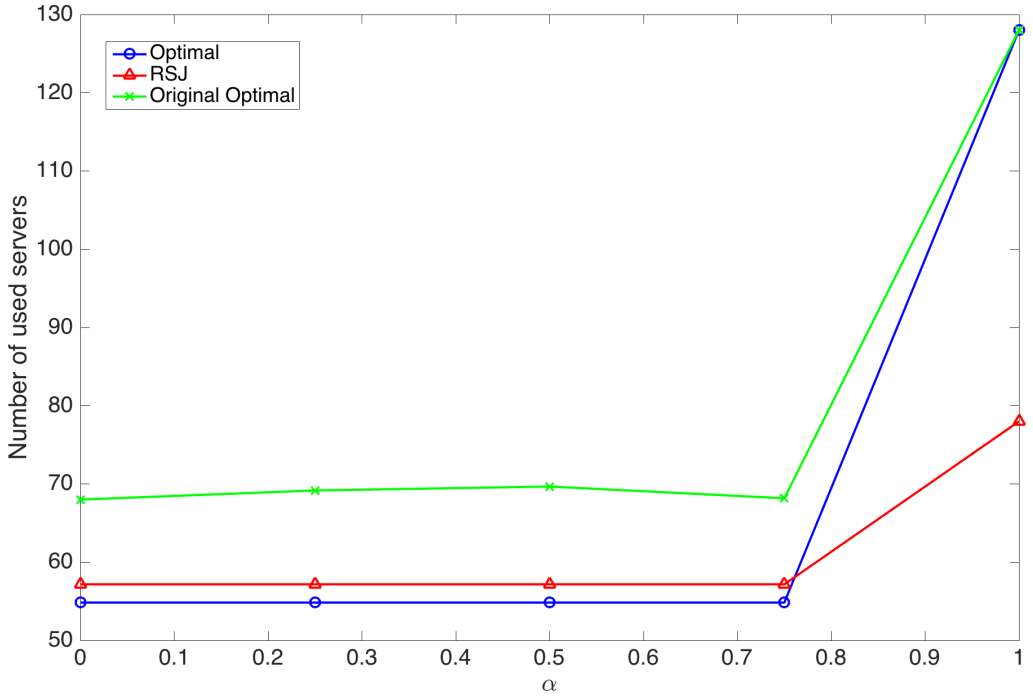


Figure 4.2: Impact of weighted factor α (Number of used servers)

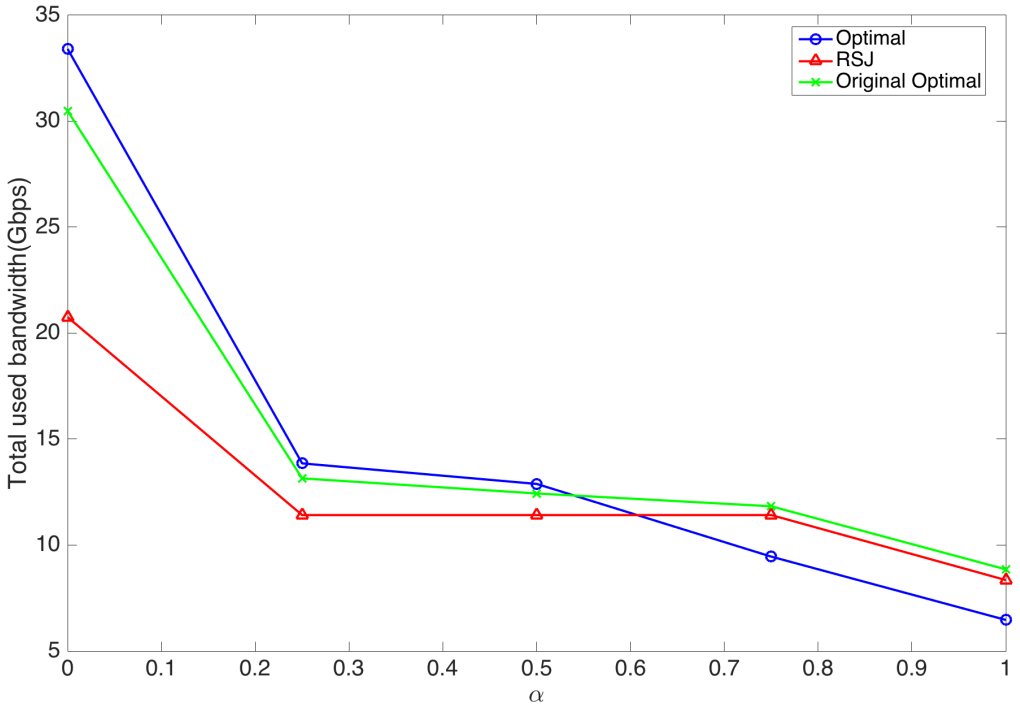
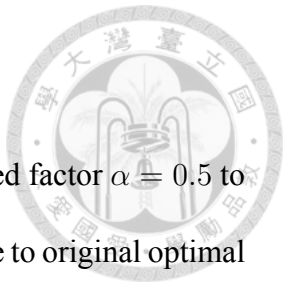


Figure 4.3: Impact of weighted factor α (Total used bandwidth)

doi:10.6342/NTU201703605

4.3 Impact of Job Ratio



We also do simulation with different job ratio. We use the weighted factor $\alpha = 0.5$ to do this simulation. We want to know how much improvement compare to original optimal in different job ratio. As the result shown in figure 4.4, the optimal would degrade to the original optimal when job ratio is 0. At this time, our algorithm is worse than the original optimal because it is optimal when job ratio is 0. With the growth of job ratio, the optimal and our algorithm would outperform than the original optimal. Due to the original optimal is fixed job dispatching, so we expect to the objective value should in the same level no matter the growth of the job ratio and the result in figure 4.4 also confirm our expectation. Figure 4.5 shows that the number of used servers in different job ratio. We can see this figure is similar to figure 4.4 because the percentage of used servers is very larger than the bandwidth utilization. Therefore, the number of used servers dominate the objective value. Even though we have this situation, however, our algorithm still can base on α to tradeoff two competitive resources and approach the optimal. Figure 4.6 shows the used bandwidth in different job ratio. The total used bandwidth is not stable because there might has some heavy flow in the demand set. And as we mentioned before, the number of used servers dominate the objective value, hence, although the optimal or our algorithm might not better than the original optimal in used bandwidth, the objective value still better than the original optimal. But when job ratio is small, the used bandwidth might not easy to compare which method is better. However, with increasing of job ratio, the performance of the optimal and our algorithm still better than the original optimal because we can dispatch more jobs to improve the performance. Also, our algorithm also can approach the optimal and outperform than the original optimal in used bandwidth.

doi:10.6342/NTU201703605

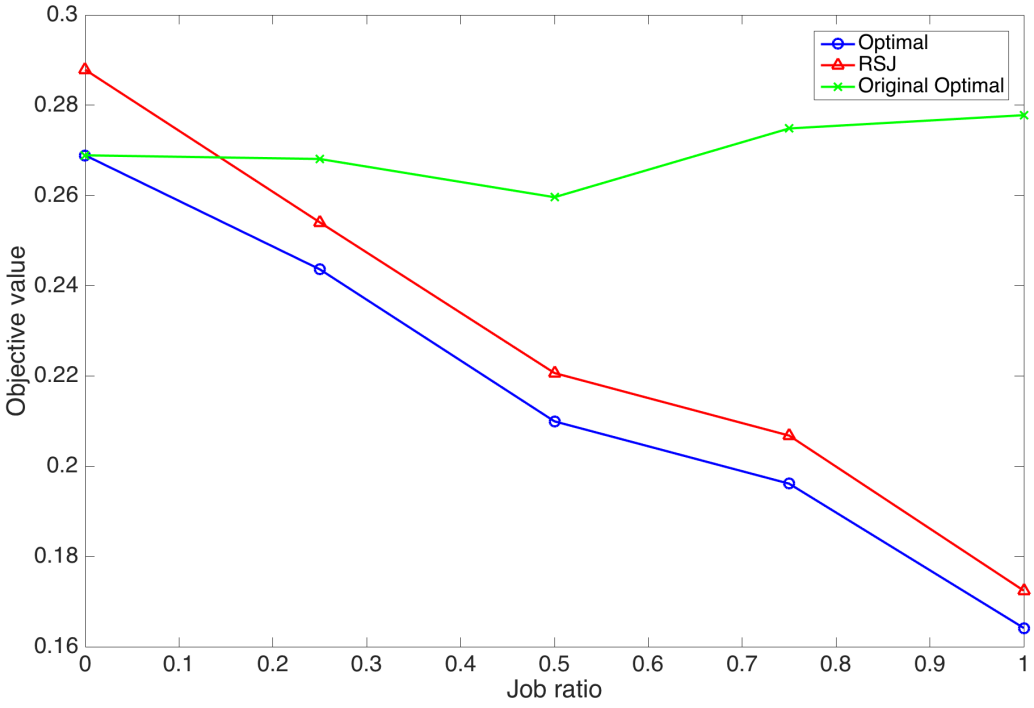


Figure 4.4: Impact of job ratio (Objective value)

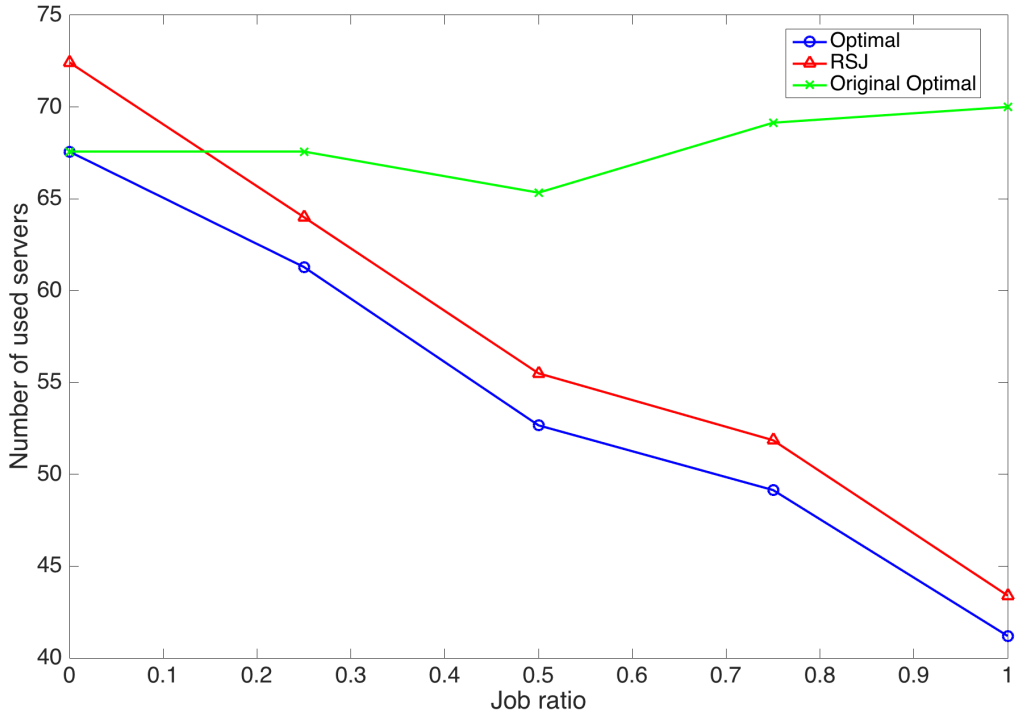


Figure 4.5: Impact of job ratio (Number of used servers)

doi:10.6342/NTU201703605

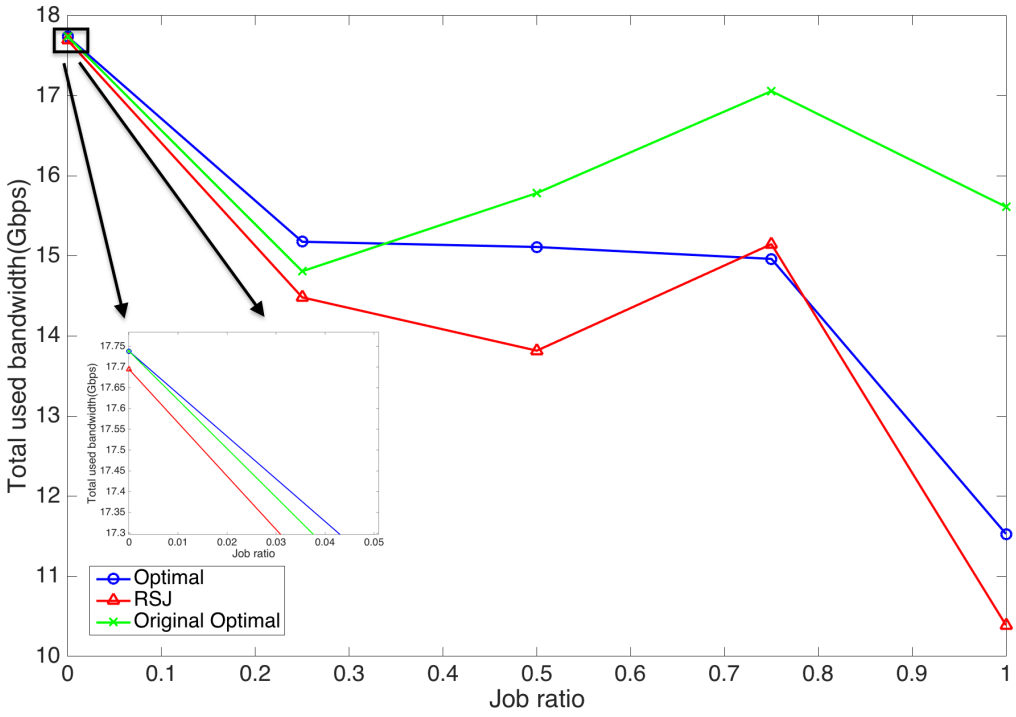


Figure 4.6: Impact of job ratio (Total used bandwidth)

4.4 Impact of Number of the Demands

We also do simulation on different number of demands. We use $\alpha = 0.5$ and job ratio $= 0.5$ to do this simulation. As the result shown as figure 4.7, we can see that the optimal is the best and our proposed algorithm still can very close to the optimal. And these two methods are both much better than the original optimal. Figure 4.8 shows the number of used server in different number of demands. The result is similar to the figure 4.7 and the reason is same as the figure 4.5. And figure 4.9 shows that the used bandwidth in different number of demands. With the growth of the number of demands, the optimal still best and our algorithm also can approach the optimal and better than the original optimal.

doi:10.6342/NTU201703605

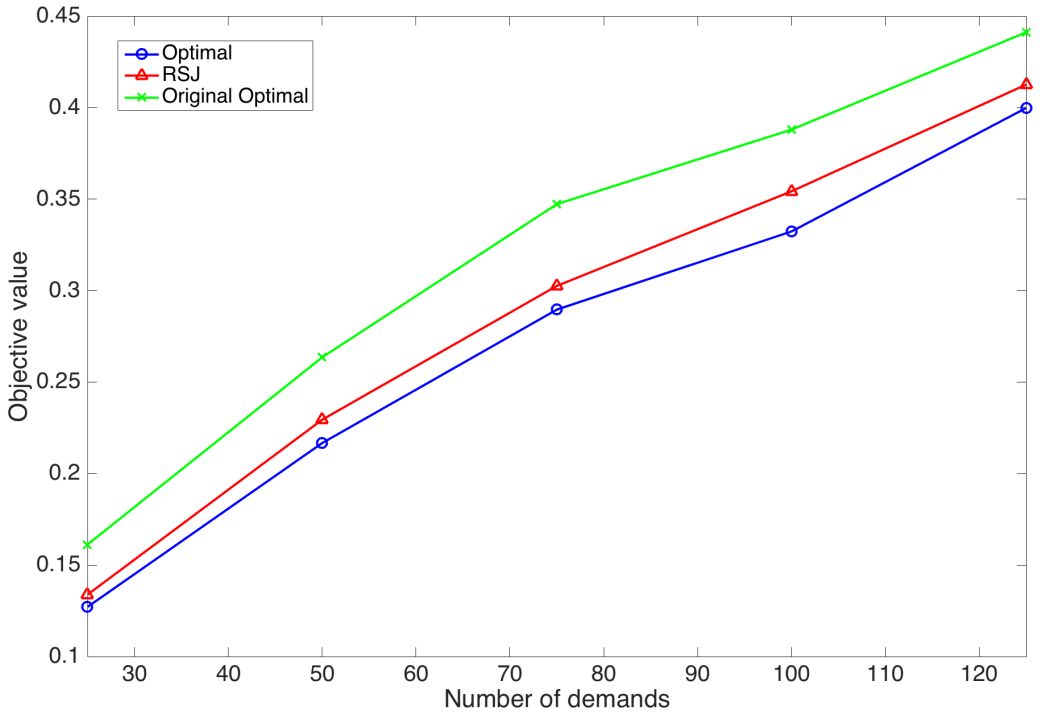


Figure 4.7: Impact of size of the demand set (Objective value)

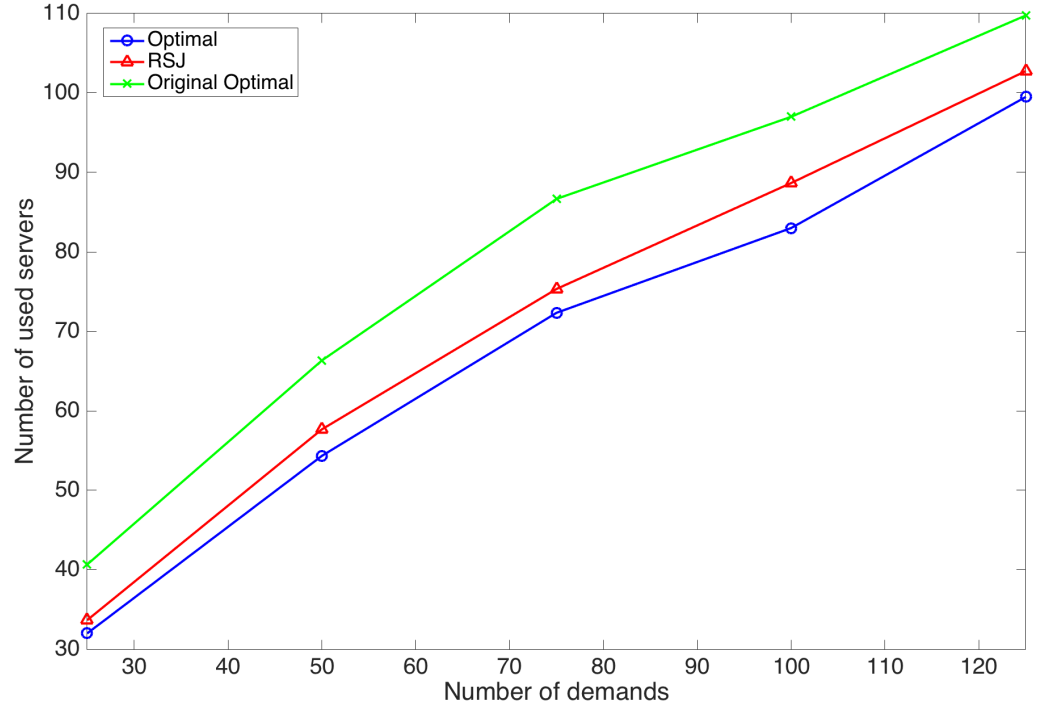


Figure 4.8: Impact of size of the demand set (Number of used servers)

doi:10.6342/NTU201703605

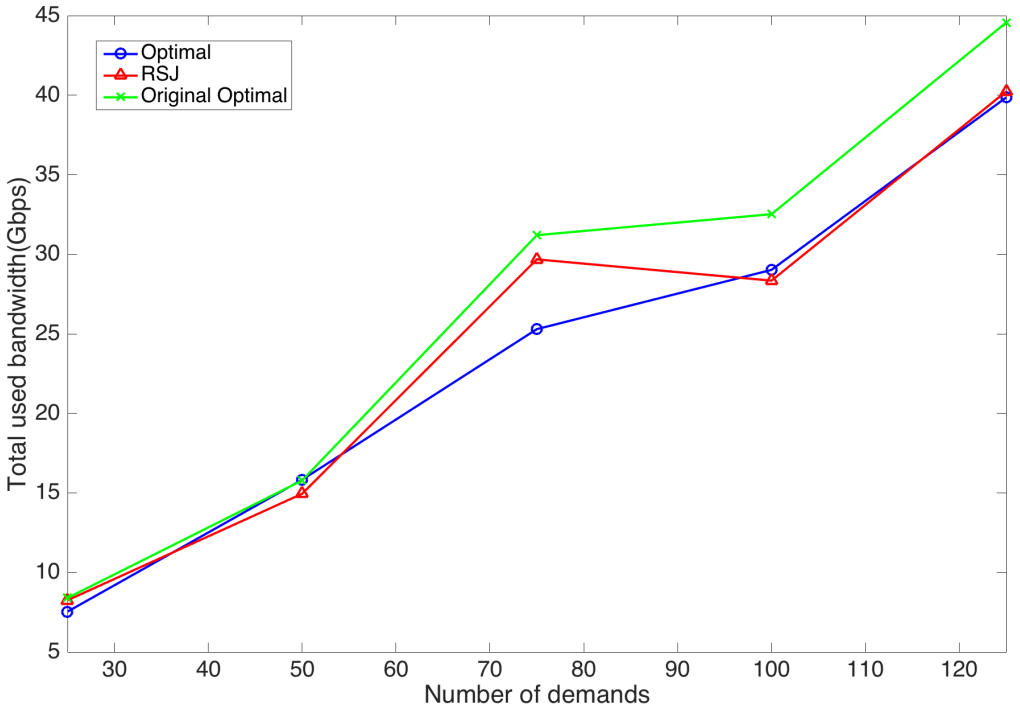


Figure 4.9: Impact of size of the demand set (Total used bandwidth)

4.5 Processing Time

We also compare the processing time between our proposed algorithm and the optimal. Figure 4.10 shows the results when the number of demands is 125. Our algorithm significantly faster than the optimal. This shows that our algorithm can find a good solution efficiently. And it also shows that the optimal need to process so much time when the number of demands is only 125. Hence, it is necessary to reduce the processing time.

doi:10.6342/NTU201703605

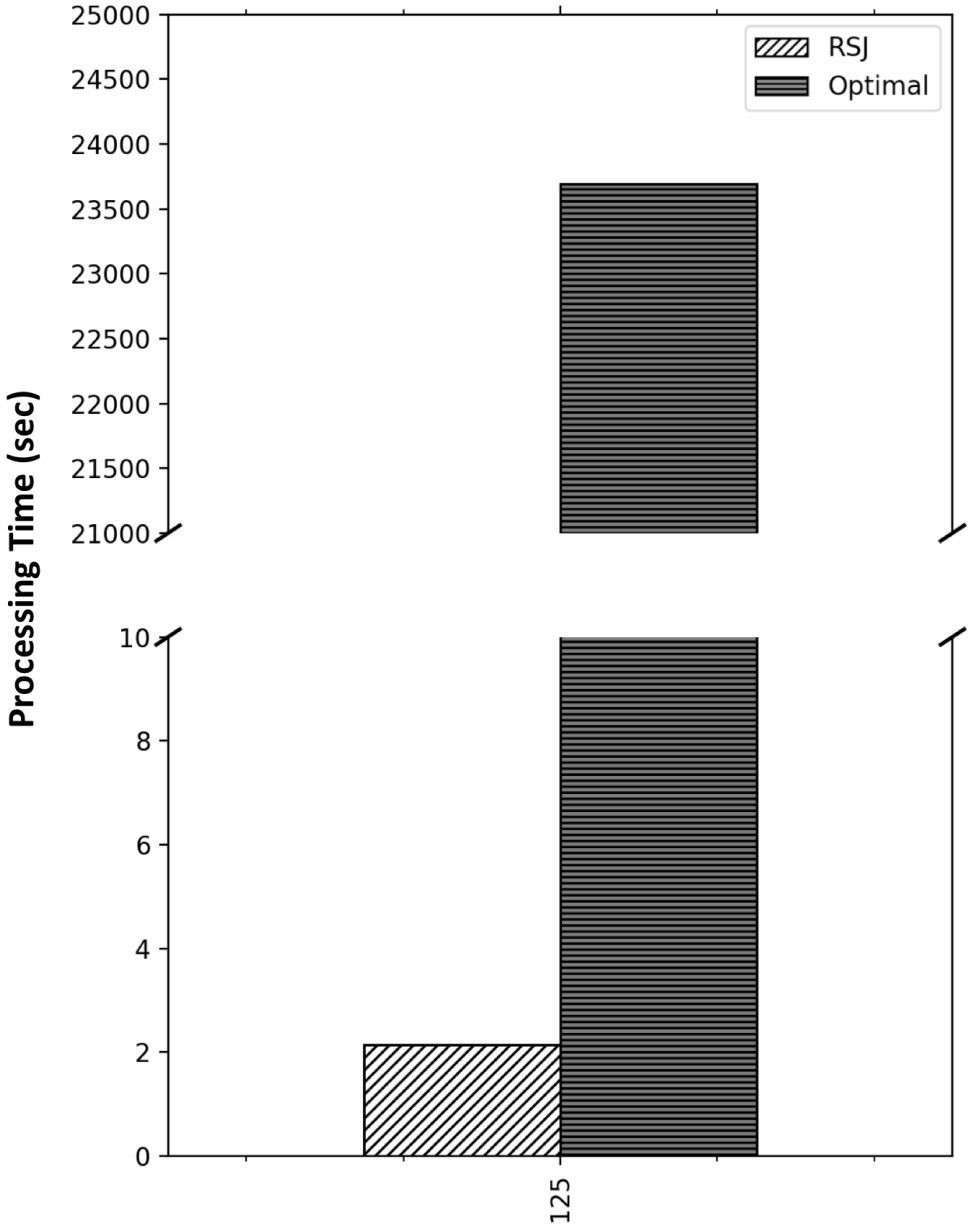


Figure 4.10: Processing time

doi:10.6342/NTU201703605

Chapter 5

Conclusion

In this thesis, we discover the relation among the bandwidth, CPU resources and the number of used servers. And we achieve tradeoff these resources. We also consider the job dispatching in the data center networks or enterprise networks to improve the service chain deployment. We formulate the service chain deployment and job dispatching problem as an integer linear programming model. However, this optimization model is very difficult to solve. Therefore, we propose a Rack-aware Service-chain deployment and Job dispatching (RSJ) algorithm to solve this problem effectively and efficiently. In our simulation, we evaluate the optimal method and our RSJ algorithm and the original optimal which is optimal method under fixed job dispatching. The simulation result shows that there has a significantly improvement with doing job dispatching together compare to fixed job dispatching. Moreover, the result also shows that our proposed RSJ algorithm can find a good solution very close to the optimal and significantly reduce the processing time.

doi:10.6342/NTU201703605

Bibliography

[1] Vyas Sekar, Sylvia Ratnasamy, Michael K Reiter, Norbert Egi, and Guangyu Shi. The middlebox manifesto: enabling innovation in middlebox deployment. In *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*, page 21. ACM, 2011.

[2] Paul Quinn and Tom Nadeau. Problem statement for service function chaining. 2015.

[3] Margaret Chiosi, Don Clarke, Peter Willis, Andy Reid, James Feger, Michael Bugenhagen, Waqar Khan, Michael Fargano, Chunfeng Cui, Hui Deng, et al. Network functions virtualisation: An introduction, benefits, enablers, challenges and call for action. In *SDN and OpenFlow World Congress*, pages 22–24, 2012.

[4] Zafar Ayyub Qazi, Cheng-Chun Tu, Luis Chiang, Rui Miao, Vyas Sekar, and Minlan Yu. Simple-fying middlebox policy enforcement using sdn. *ACM SIGCOMM computer communication review*, 43(4):27–38, 2013.

[5] Abhishek Dwaraki and Tilman Wolf. Adaptive service-chain routing for virtual network functions in software-defined networks. In *Proceedings of the 2016 workshop on Hot topics in Middleboxes and Network Function Virtualization*, pages 32–37. ACM, 2016.

[6] Milad Ghaznavi, Aimal Khan, Nashid Shahriar, Khalid Alsubhi, Reaz Ahmed, and Raouf Boutaba. Elastic virtual network function placement. In *Cloud Networking (CloudNet), 2015 IEEE 4th International Conference on*, pages 255–260. IEEE, 2015.

doi:10.6342/NTU201703605

[7] Sevil Mehraghdam, Matthias Keller, and Holger Karl. Specifying and placing chains of virtual network functions. In *Cloud Networking (CloudNet), 2014 IEEE 3rd International Conference on*, pages 7–13. IEEE, 2014.

[8] Marcelo Caggiani Luizelli, Leonardo Richter Bays, Luciana Salete Buriol, Marinho Pilla Barcellos, and Luciano Paschoal Gaspary. Piecing together the nfv provisioning puzzle: Efficient placement and chaining of virtual network functions. In *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*, pages 98–106. IEEE, 2015.

[9] Tung-Wei Kuo, Bang-Heng Liou, Kate Ching-Ju Lin, and Ming-Jer Tsai. Deploying chains of virtual network functions: On the relation between link and server usage. In *Computer Communications, IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on*, pages 1–9. IEEE, 2016.

[10] Matthew Andrews, Julia Chuzhoy, Venkatesan Guruswami, Sanjeev Khanna, Kunal Talwar, and Lisa Zhang. Inapproximability of edge-disjoint paths and low congestion routing on undirected graphs. *Combinatorica*, 30(5):485–520, 2010.

[11] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. A scalable, commodity data center network architecture. In *ACM SIGCOMM Computer Communication Review*, volume 38, pages 63–74. ACM, 2008.

[12] CPLEX Optimization Studio 12.7. https://www.ibm.com/bs-en/marketplace/ibm-ilog-cplex.

[13] Xin Li and Chen Qian. Low-complexity multi-resource packet scheduling for network function virtualization. In *Computer Communications (INFOCOM), 2015 IEEE Conference on*, pages 1400–1408. IEEE, 2015.

[14] Theophilus Benson, Aditya Akella, and David A Maltz. Network traffic characteristics of data centers in the wild. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pages 267–280. ACM, 2010.

doi:10.6342/NTU201703605