Computer Networks 80 (2015) 109–123

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Joint VM placement and topology optimization for traffic scalability in dynamic datacenter networks

Yangming Zhao [a,b], Yifan Huang [b], Kai Chen [b], Minlan Yu [c], Sheng Wang [a,*], DongSheng Li [d]

[a] Key Lab of Optical Fiber Sensing and Communication, Education Ministry of China, University of Electronic Science and Technology of China, Chengdu, PR China
[b] Hong Kong University of Science and Technology, Hong Kong, China
[c] Computer Science Department at University of Southern California, LA, CA, USA
[d] School of Computer, National University of Defense Technology, PR China

ARTICLE INFO

Article history:
Received 17 April 2014
Received in revised form 9 October 2014
Accepted 17 December 2014
Available online 4 February 2015

Keywords:
VM placement
Dynamic datacenter networks
Joint optimization
Lagrange's relaxation decomposition

ABSTRACT

In dynamic datacenter networks (DDNs), there are two ways to handle growing traffic: adjusting the network topology according to the traffic and placing virtual machines (VMs) to change the workload according to the topology. While previous work only focused on one of these two approaches, in this paper, we jointly optimize both virtual machine placement and topology design to achieve higher traffic scalability. We formulate this joint optimization problem to be a mixed integer linear programming (MILP) model and design an efficient heuristic based on Lagrange's relaxation decomposition. To handle traffic dynamics, we introduce an online algorithm that can balance algorithm performance and overhead. Our extensive simulation with various network settings and traffic patterns shows that compared with randomly placing VMs in fixed datacenter networks, our algorithm can reduce up to 58.78% of the traffic in the network, and completely avoid traffic overflow in most cases. Furthermore, our online algorithm greatly reduces network cost without sacrificing too much network stability.

1. Introduction

As cloud computing becomes a reality and offers an attractive pay-as-you-go charge model [1,2], users have been increasingly outsourcing their computing demands and services to clouds by renting virtual machines (VMs). This leads to significant traffic growth in cloud datacenters. Therefore, how to handle such a growing amount of workload in a scalable manner and how to optimize networks to accommodate traffic growth are important research issues [3–11]. In this paper, we term this *traffic scalability* problem. In addition to statically provision a uniformly high bandwidth with rich network connectivity such as Fattree [5], VL2 [6], BCube [7], DCell [8], there are two attractive approaches to track the challenges in such a traffic scalability problem in oversubscribe networks.

One approach is to adapt the datacenter network topology to traffic. This is realized by dynamically changing the network topology to fit the underlying traffic. Along this line of thought, researchers have exploited wireless communication technologies (e.g., Flyways [12] and 3D beamforming [13]) or optical switching technologies (e.g., OSA [3], Helios [4] and c-Through [14]) to build dynamic datacenter networks (DDNs). Among all these existing schemes, OSA is the state-of-the-art work that provides the most flexibility because it can dynamically adapt its topology as well as link capacity to the traffic, thus improving traffic scalability. In this paper, we leverage the OSA-based DDN model to control the network topology in our design. OSA-based DDN model is detailedly reviewed in Section 2.1.

* Corresponding author.
E-mail address: wsh_keylab@uestc.edu.cn (S. Wang).

http://dx.doi.org/10.1016/j.comnet.2014.12.014
1389-1286/

The other approach is to adapt the traffic to fit the datacenter network. This is achieved by applying intelligent VM placement algorithms to allocate VMs with high traffic demands to proximate hosts with high bandwidth connectivity. Such intelligent VM placement already exists in practice, e.g., Amazon EC2's dynamic creation and deletion of VM instances. There are a lot of recent research efforts [13,9,10,15] on VM placement and shuffling for diverse goals, and among which the VM placement problem (VMPP) [10] is a representative that addresses traffic scalability issues in datacenters. More specifically, it studies how to map each VM to a host in order to minimize the network cost, e.g., the total delay or the total traffic in the network. We review VMPP in detail in Section 2.2.

However, VMPP [10] assumes a fixed topology and shifts the workload, whereas DDNs focuse on optimizing network topology given fixed workload. These two approaches are the twofolds of improving network traffic scalability. Intuitively, jointly optimizing VM placement and network topology would yield a better solution as it provides a much larger optimization space although it would be quite challenging.

When VM placement and topology are jointly optimized, their interaction can be shown in Fig. 1. The output of VM placement (i.e., traffic demands between racks) is the input of topology optimization, while the output of topology optimization (i.e., interconnections between racks) is also part of the input of VM placement. Accordingly, we cannot directly adopt algorithms in OSA [3] or VMPP [10] to solve joint optimization due to the lack of input for either algorithm.

This paper makes the following main contributions:

- We are among the first to explore the optimization space of jointly optimization VM placement and topology to improve traffic scalability in DDNs, and formulate this problem as a mixed integer linear programming (MILP) model (Section 3).
- Since the MILP model is intractable at scale, we propose an efficient heuristic to solve this problem based on the Lagrange's relaxation decomposition, and we also analyze the approximation ratio of the proposed heuristic (Section 4).
- To handle dynamic traffic changes, we further propose an online optimization algorithm to the problem at runtime which can balance the algorithm performance and overhead (Section 5).

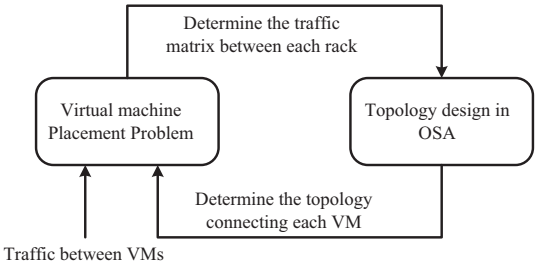


Fig. 1. Interaction between VM Placement and topology optimization.

Our simulation results suggest that our algorithm of joint optimization improves traffic scalability more significantly than only optimizing either VM placement or topology, and our online algorithm greatly reduces network cost without frequently reconfiguring topology.

For clarity, we list all the notations that will be used in this paper in Table 1.

2. Background and motivation

As mentioned, there are two ways to improve traffic scalability. One is to adapt topology to traffic in DDNs, in which OSA [3] is a representative. The other is placing VMs to adapt traffic to network topology, in which VMPP [10] is a representative. We will review these two approaches in Section 2.1 and 2.2, respectively. After that, we will discuss why we need to jointly optimize them in Section 2.3.

2.1. OSA-based DDNs

OSA-based DDN is such a network model that can dynamically change its topology and link capacity to fit traffic.

The architecture of OSA-based DDNs is shown in Fig. 2. It achieves dynamic topology via exploiting the reconfigurability of Micro-Electro-Mechanical Switch (MEMS). MEMS is a bipartite $N \times N$ matrix, in which any input port can be connected to any one of the output ports. If we connect each of $\frac{N}{d}$ racks to d ports of MEMS (assume N is divisible by d), each rack is directly connecting with d racks. By this connection, the DDN model can form all the topologies satisfying

$$\sum_j e_{ij} \leqslant d \qquad for\ all\ i \tag{1}$$

and

$$e_{ij} = e_{ji} \qquad for\ all\ i,\ j \tag{2}$$

where e_{ij} indicates whether rack i and rack j are directly connected via MEMS. (2) is due to the fact that the connection between each rack pair is mutual.

Another advantage of the OSA-based DDN model is that it can flexibly modify the capacity of each link by utilizing Wavelength Selective Switching (WSS). If we are to create a link with capacity that is k times of a single wavelength between rack i and rack j, k wavelengths can be assigned to this link. To this end, it first multiplexes all the wavelengths from rack i into a single fiber connecting a WSS module. Then the WSS module can split the expected k wavelengths to the appropriate port with circuit connecting to rack j. In this way, operator can easily change the capacity of each link by choosing different k and the traffic on each link only needs to satisfy

$$f^{uv} \leqslant C_w \sum_k w_{uv}^k \qquad for\ all\ u, v \tag{3}$$

where f^{uv} is the traffic rate directly traversing from rack u to rack v, i.e., on link (i,j), C_w is the capacity per wavelength

Table 1
Notation used in our work.

Notation	Description
x_{ki}	Binary variable. It takes 1 if VM k is connected to rack i, and 0 otherwise
e_{ij}	Binary variable. It takes 1 if rack i and rand j are connected directly by DDNs, and 0 otherwise
f_{ij}	Real variable. It denotes the traffic between rack i and rack j
f^{uv}	Real variable. It denotes the traffic rate directly traverse from rack u to rack v. If rack u and rack v are not connected directly, there must be $f^{uv}=0$
f_{ij}^{uv}	Real variable. It denotes the bandwidth used by the demand between rack i and rack j on link (u, v)
z_{kl}^{ij}	Binary variable. It takes 1 if the traffic between VM k is under rack i and VM l is under rack j, and 0 otherwise
v_{kl}	Real constant. It is the traffic rate between VM k and VM l
w_{uv}^{k}	Binary variable. It takes 1 if wavelength k is used to directly carry traffic from rack u to rack v
C_w	Real constant. It is the capacity per wavelength
d	Integer constant. It is the maximum nodal degree of each vertex, which is determined by DDNs' parameter
$f_{ij}^{(r)}$	Real variable. The required traffic rate from rack i to rack j which is determined by VM placement
$f_{ij}^{(s)}$	Real variable. The traffic rate from rack i to rack j that is served by DDNs
$C_{ij}^{(O)}$	Real variable. Network cost corresponding to over flow and incurred by traffic from rack i to rack j
$C_{ij}^{(C)}$	Real variable. Network cost corresponding to network capacity usage and incurred by traffic from rack i to rack j
C_{ij}	Real variable. Total network cost incurred by traffic from rack i to rack j, which is equal to the sum of $C_{ij}^{(O)}$ and $C_{ij}^{(C)}$
T	Real constant. It is a large number that is larger than the sum of all the traffic rate between each VMs in the networks
H_i	Integer constant. It is the number of host under rack i
R	Integer constant. It is the number of racks in the network

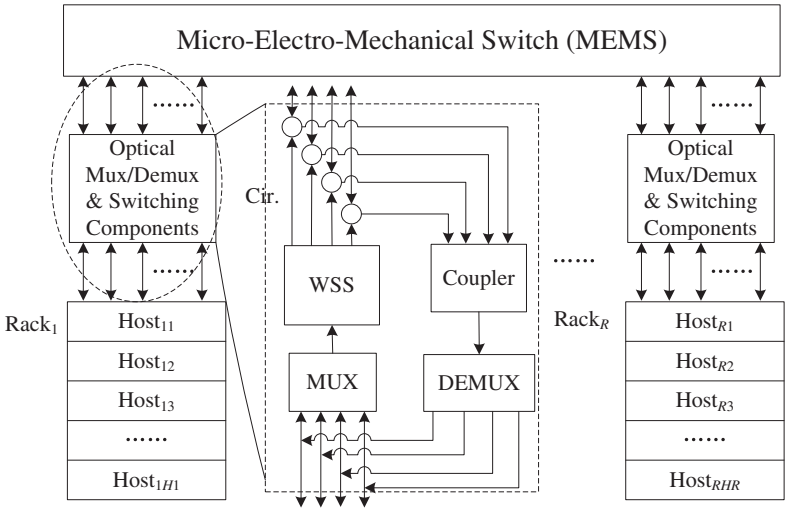


Fig. 2. Architecture of OSA-based DDNs.

and w_{uv}^{k} indicates whether wavelength k is used on link (u, v).

It also should be noted that a fiber cannot carry more than one channel over the same wavelength. Accordingly, each rack port should be assigned special wavelengths across its rack, i.e., the following constraint should be satisfied,

$$\sum_{v} w_{uv}^{k} \leqslant 1 \qquad for\ all\ u,\ k \tag{4}$$

2.2. VM placement problem

In a datacenter, CPU/memory resources are provided by physical hosts. Operators should assign these resources to VMs, i.e., determine where each VM to place to serve requests. Without loss of generality, VMPP [10] assumes there are n VMs and n hosts in the network and each host can serve one VM (If one host can support multiple VMs, we can introduce multiple virtual hosts that can support only one VM to substitute the physical host, such that our solution can work after the substitution.). In this case, the question becomes how to map n VMs to n hosts in order to reduce the total network cost. In other words, suppose C_{ij} denotes the communication cost from host i to host j for each unit of traffic, and D_{ij} denotes the traffic rate from VM i to VM j. The goal of VMPP is to find a mapping $\pi : [1, \ldots, n] \rightarrow [1, \ldots, n]$, such that the objective

$$NetworkCost = \sum_{i,j} C_{\pi(i)\pi(j)} D_{ij} \tag{5}$$

is minimized. Usually, C_{ij} is defined as the number of hops on the path from VM i to VM j. By this setting, the network cost refers to the total used capacity in the network. Furthermore, given all hosts under the same rack have identical status in terms of the traffic contribution to the network, the overall network cost will not be affected by VM placement within a certain rack. Accordingly, we only focus on under which rack each VM is placed, i.e., if we assume that there are H_i hosts under rack i, we only should find x_{ki}, such that

$$\sum_k x_{ki} = H_i \qquad for\ all\ i \tag{6}$$

to minimize (5), where x_{ki} is used to indicate whether VM k is placed under rack i.

2.3. Why jointly optimizing VM placement and topology

OSA-based DDNs [3] dynamically optimize the topology and link capacity according to a given traffic distribution, while VM placement [10] intelligently places VMs in order to optimize the traffic distribution for given network topology. In joint optimization, topology can be configured according to the traffic distribution, and hence VMs can be placed in such a way that the resulting traffic distribution can better feed the topology.

Furthermore, it should be noted that the original design in VMPP [10] has not considered the link traffic overflow caused by greedy, unbalanced VM placement (i.e., the desired bandwidth on a particular link exceeds its capacity). We believe this shortcoming can be overcome by joint optimization as dynamic link capacity can handle this overflow by assigning more bandwidth to the links carrying more traffic.

In addition, though it is proved in [10] that VMPP for a fixed topology is the hardest NP-hard problem, i.e., we cannot even find a σ-approximation solution in polynomial time, it may not be the case whether we can change the network topology. In joint optimization, topology constraint in VM placement may get relaxed, and hence it may be simplified. Therefore, we focus on jointly optimizing VM placement and topology in this paper to seek additional optimization space.

3. Problem formulation

3.1. Network model

The network model studied in this paper can be described in Fig. 2. In such network, each rack contains several hosts and all the racks are connected through DDNs. With the traffic between each VM pair as input, two problems should be solved to jointly optimize VM placement and topology. The first one is the configuration of MEMS and the wavelength assignment on each link, i.e., the topology design. The other one is how to map VMs to hosts in the network. In this paper, we assume that no traffic blocking will occur under the same rack.

3.2. Cost model

In this paper, network cost consists of two parts. One is the capacity used in the network (called *capacity cost*), and the other part is the penalty incurred by traffic overflow (called *overflow cost*). Since there is no traffic overflow within the same rack, the network cost only counts the traffic between racks.

Similar to the previous work [10], we define the capacity cost, $C_{ij}^{(C)}$, where the superscript C means *capacity*, incurred by the traffic between rack i and rack j to be

$$C_{ij}^{(C)} = f_{ij}^{(s)} h_{ij} \tag{7}$$

where $f_{ij}^{(s)}$ is the served traffic rate between rack i and j, while h_{ij} is the hop number of the path between these two racks. Overflow cost incurred by the traffic between rack i and rack j, $C_{ij}^{(O)}$ (the superscript O means *overflow*), is defined as

$$C_{ij}^{(O)} = \gamma(f_{ij}^{(r)} - f_{ij}^{(s)}) \tag{8}$$

where γ is a large number as a penalty factor and $f_{ij}^{(r)}$ is the required traffic rate between rack i and rack j. Obviously,

$$f_{ij}^{(s)} \leqslant f_{ij}^{(r)} \qquad for\ all\ i,\ j \tag{9}$$

and $f_{ij}^{(r)}$ is calculated as

$$f_{ij}^{(r)} = \sum_{k,l} v_{kl} z_{kl}^{ij} \tag{10}$$

where v_{ij} denotes the required traffic rate from VM i to VM j, and z_{kl}^{ij} indicates whether VM k is connecting to rack i and VM l is connecting to VM j. In this case, cost incurred by traffic from rack i to rack j is

$$C_{ij} = C_{ij}^{(C)} + C_{ij}^{(O)} \tag{11}$$

Network cost is the sum of cost incurred by traffic between all rack pairs, i.e.,

$$\begin{aligned} C &= \sum_{i,j:i\neq j} C_{ij} = \sum_{i,j:i\neq j} [f_{ij} h_{ij} + \gamma(f_{ij}^{(r)} - f_{ij}^{(s)})] \\ &= \sum_{(u,v)\in E} f^{uv} + \sum_{i,j:i\neq j} \gamma(f_{ij}^{(r)} - f_{ij}^{(s)}) \end{aligned} \tag{12}$$

3.3. MILP formulation

From the definition of z_{kl}^{ij}, it takes 1 if and only if $x_{ki} = 1$ and $x_{lj} = 1$, i.e., z_{kl}^{ij} satisfies

$$\frac{x_{ki} + x_{lj} - 1}{2} \leqslant z_{kl}^{ij} \leqslant \frac{x_{ki} + x_{lj}}{2} \qquad for\ all\ i,\ j,\ k,\ l \tag{13}$$

In DDNs, if rack u and rack v are not directly connected, no traffic can be carried by link (u, v), i.e.

$$f_{ij}^{uv} \leqslant T e_{uv} \qquad for\ all\ i,\ j,\ u,\ v \tag{14}$$

where T is a large number which can be set as the sum of all the traffic existing in the network, and f_{ij}^{uv} is the traffic

rate from rack i to j and carried by link (u, v). Then the traffic on link (u, v) can be calculated by

$$f^{uv} = \sum_{i,j} f_{ij}^{uv} \qquad for\ all\ u,\ v \tag{15}$$

Also, flow conservation constraints should be satisfied,

$$\sum_{v} f_{ij}^{uv} - \sum_{v} f_{ij}^{vu} = \begin{cases} f_{ij}^{(s)} & \text{if } u = i \\ -f_{ij}^{(s)} & \text{if } u = j \\ 0 & \text{otherwise} \end{cases} \tag{16}$$

In addition, all the structure constraints of VM placement and OSA-based DDNs discussed in Section 2.1 and 2.2 should also be treated as constraints in the model. In summary, the jointly optimizing VM placement and topology to minimize network cost can be formulated as

Jointly optimizing VM placement and topology (JOVT):

minimize: (12)

subject to:

Topology constraints: (1)–(4),

Maximum VM number under each rack: (6),

Interaction between VM placement and topology design: (9), (10), (13)–(16).

3.4. Why jointly optimizing VM placement and topology is a challenge

The MILP formulated in previous subsection is an NP-complete problem [16] and intractable in large-size networks, so that an efficient heuristic is required. Though the heuristics to only design topology or place VM are efficient, they cannot be transplanted to joint optimization problem. It is not only because that the input of topology design is the output of VM placement and vice versa, but also due to the probability that these heuristics may not be efficient in joint optimization. On the one hand, VM placement algorithm proposed in [10] leverages the hierarchical datacenter topology to cluster VMs, it may not be applicable in OSA-based DDNs which is a flat architecture. On the other hand, the topology formed by OSA-based DDNs [3] is not necessary to be a connected graph since we can change traffic matrix by placing VMs.

Another challenge in jointly optimizing VM placement and topology is to handle time-varying traffic in realistic networks. Topology design can be solved in a short time scale, but it may incur route oscillation if it is triggered too frequently. Though doing VM placement will not suffer from route oscillation, it takes too much time to get a solution and is not suitable to time-varying traffic scenario.

4. Algorithm design

4.1. Problem analysis

Since **JOVT** formulated in Section 3.3 is intractable in large-size networks, efficient heuristic is required. To design an efficient heuristic, we first analyze the structure of **JOVT** and expect to get some valuable insights.

It can be noted that all the variables in (6), (10) and (13) do not appear in other constraints of **JOVT** except (9). Accordingly, if constraint (9) can be relaxed, **JOVT** can be decomposed into two subproblems and the problem complexity can be degraded. In other words, Lagrange's relaxation decomposition works in **JOVT** model. Accordingly, we first relax constraint (9) in **JOVT** by introducing Lagrange multiplexer $\lambda_{ij} \geqslant 0$ for all i, j. Then, we obtain,

Lagrange relaxed JOVT (L-JOVT):

$$\text{minimize}: \quad \sum_{u,v} f^{uv} + \sum_{i,j} (\gamma - \lambda_{ij})(f_{ij}^{(r)} - f_{ij}^{(s)}) \tag{17}$$

subject to: constraints in **JOVT** except (9)

Then **L-JOVT** can be decomposed as

SubProblem 1:

$$\text{minimize}: \quad S_1(\lambda_{ij}) = \sum_{i,j} (\gamma - \lambda_{ij}) f_{ij}^{(r)} \tag{18}$$

subject to: 6, 10, 13

and

SubProblem 2:

$$\text{minimize}: \quad S_2(\lambda_{ij}) = \sum_{i,j} [f^{ij} - (\gamma - \lambda_{ij}) f_{ij}^{(s)}] \tag{19}$$

subject to: (1)—(4), (14)—(16)

Now, the lower bound of network cost can be calculated by

$$\max_{\lambda_{ij}} \quad S_1(\lambda_{ij}) + S_2(\lambda_{ij}) \tag{20}$$

Due to the rotation symmetry of i, j in $f_{ij}^{(r)}$ and $f_{ij}^{(s)}$ of **L-JOVT**, $\lambda_{ij} = \lambda$ for all i, j. If γ is large enough, i.e. $\gamma - \lambda$ is positive, SubProblem 1 can be degraded to minimize $\sum_{i,j} f_{ij}^{(r)}$.

Consider a weighted full mesh graph $\boldsymbol{G} = \langle \boldsymbol{V}, \boldsymbol{E} \rangle$, each vertex in $\boldsymbol{G}$ denotes one VM in the network and the weight on each edge is the required traffic rate between the VMs corresponding to its two end vertexes. In this case, SubProblem 1 is equivalent to find the minimum k cut in $\boldsymbol{G}$ to divide it into R groups, such that all the nodes in a group correspond to the VMs should be placed under the same rack. After we determine under which rack each VM should be placed, the traffic matrix in the DDN is easy to yield. Then, the remaining question is how to configure topology.

Consider the two items in the objective of SubProblem 2, the first one is exactly the capacity cost in the network, while the second is the served traffic rate. Since $\gamma - \lambda_{ij}$ is a large number, we should first guarantee that network can serve as much traffic as possible, and then minimize the hop number of each traffic to reduce capacity cost. In the following two subsections, we will introduce our algorithms to place VMs and configure DDN topology, respectively.

4.2. Virtual machine clustering

From previous analysis, we can use minimum k cut algorithm introduced in [17] to divide all the VMs into R groups with required group size. The algorithm is shown in Algorithm 1. There is an issue in this algorithm at Line 5, where to check whether $\boldsymbol{G}$ is divided into two parts such

that one of them contains exactly b_i VMs. Since more than two parts may remain when j cuts are removed from $\boldsymbol{G}$, the question can be described as:

Algorithm 1. Minimum k-cut algorithm.

Require: Weighted graph $\boldsymbol{G} = \langle \boldsymbol{V}, \boldsymbol{E} \rangle$ and the size of each group $\{b_1, b_2, \ldots, b_k\}$
Ensure: Group set $\boldsymbol{S} = \{s_1, s_2, \ldots, s_k\}$, where s_i is a set of vertex in V with size b_i, such that $\cup s_i = V$ and $s_i \cap s_j = \Phi$
1: Compute the vertex number in each group $l = \frac{n}{k}$, where n is the size of G
2: Compute Gomory–Hu tree [17] for G and obtain $n-1$ cuts $\{g_i\}$
3: Sort $\{g_i\}$ by increasing order
4: **for** i = 1 to k **do**
5: Finding minimal j such that $\{g_1, g_2, \ldots, g_j\}$ divide $\boldsymbol{G}$ into two parts such that one of them is with size b_i. Suppose the part with size b_i is c
6: $s_i \leftarrow c$
7: $\boldsymbol{G} \leftarrow \boldsymbol{G}/c$
8: **end for**
9: **return** $\boldsymbol{S}$

Question 1. Suppose there are j positive integers $a_1, a_2, \ldots, a_j$, how to pick out some of these integers such that the sum of them is a given integer s.

Question 1 is NP-hard since it can be reduced to an Knapsack problem [18]. Say there are j items whose weight and value are both $a_1, a_2, \ldots, a_j$, respectively, and the total weight limit is s. In this case, the answer of Question 1 is corresponding to the solution of this constructed Knapsack problem. If the optimal solution of this Knapsack problem is exactly s, the solution is derived. Otherwise, Question 1 is infeasible. To quickly separate l vertexes from $\boldsymbol{G}$, we first prove following lemma:

Lemma 1. *Suppose $a_1, a_2, \ldots, a_l$ are l positive integers, such that*

$$\sum_{i=1}^{l} a_i = n$$

where $l = \frac{n}{2} + 1$ if n is even and $l = \lceil \frac{n}{2} \rceil$ if n is odd. For arbitrary integer s $(0 < s < n)$, there must be some integers from $a_1, a_2, \ldots, a_l$ such that the sum of them is s.

Proof. Without loss of generality, we assume a_i is sorted in non-increasing order and let $\boldsymbol{A} = \{a_1, a_2, \ldots, a_l\}$. For arbitrary k, once any integer in $[1, a_k - 1]$ can be obtained by summing up some numbers from $\{a_m, m > k\}$, Lemma 1 is correct. Obviously, if $a_1 = 2$, there must be $a_2 = a_3 = \cdots = a_{\frac{n}{2}-1} = 2$, $a_{\frac{n}{2}} = a_{\frac{n}{2}+1} = 1$ when n is even and $a_2 = a_3 = \cdots = a_{\lfloor \frac{n}{2} \rfloor - 1} = 2$, $a_{\lfloor \frac{n}{2} \rfloor} = 1$, when n is odd. In general, if any number larger than 1 increasing by 1, there must be one more "1" appearing in $\boldsymbol{A}$. Accordingly, if there is an integer $a_k, (a_k > 2)$ in $\boldsymbol{A}$, there must be at least $(a_k - 2) + 1 = a_k - 1$ "1"s in $\boldsymbol{A}$. With these "1"s, we can get all the integers in $[1, a_k - 1]$. □

Based on Lemma 1, we can design Algorithm 2 to obtain each group of vertexes for Line 5 in Algorithm 1. The key idea of this algorithm is to divide graph into $\lfloor \frac{n}{2} \rfloor + 1$ blocks at first, and then pick out some blocks to form a group with l vertexes. When we pick out blocks to form group, the larger blocks have priority to be selected than the smaller ones since traffic rate between VMs in such blocks may be large and they should be placed under the same rack.

Algorithm 2. Algorithm to get a group of vertexes

Require: A Gomory–Hu tree and all the cuts on this tree $\{g_i\}$ (These cuts have already been sorted in increasing order)
The vertex number in a group l.
Ensure: A group of vertexes s with size l
1: $n \leftarrow$ size of $\boldsymbol{G}$, $s \leftarrow \Phi$, $N \leftarrow 0$, $i \leftarrow 1$
2: remove the $\lfloor \frac{n}{2} \rfloor$ smallest cuts from G, such that G is divided into $\lfloor \frac{n}{2} \rfloor + 1$ parts $\{p_i\}$
3: Sort $\{p_i\}$ in non-increasing order in terms of its size
4: **while** $N \neq l$ **do**
5: $m \leftarrow$ size of p_i
6: **if** $m + N < l$ **then**
7: $s \leftarrow s \cup p_i, N \leftarrow m + N$
8: **end if**
9: $i \leftarrow i + 1$
10: **end while**
11: **return** s

4.3. Link provisioning

From Algorithm 1, we can divide all VMs into R groups and place VMs in the same group to the hosts under one rack according to the group size. Consequently, the traffic between each pair of racks can be easily obtained. With such traffic information, a heuristic can be proposed to design topology. For clarity, we say the traffic between a rack pair as a "demand" in this subsection. Before we present the algorithm to design DDN topology in detail, we first review a classic inequality.

Lemma 2 (*rearrangement inequality [19]*). *For $a_1 < a_2 < \cdots < a_n$ and $b_1 < b_2 < \cdots < b_n$, if τ is a one-to-one mapping from $S = \{1, \ldots, n\}$ to $B = \{1, \ldots, n\}$, the following inequation must be held*

$$\sum_{i=1}^{n} a_i b_{n+i-1} \leqslant \sum_{i=1}^{n} a_i b_{\tau(i)} \leqslant \sum_{i=1}^{n} a_i b_i$$

Motivated by rearrangement inequality shown in Lemma 2, the demand with larger traffic rate should be offered a shorter route. To this end, we first construct a full-mesh graph with the purpose to keep all the routes being available for demands, and then route all the demands on the constructed graph in the non-increasing order in terms of their traffic rate. Once an edge without any corresponding concrete link in physical topology, say edge e, is used by

any demand, one concrete link, say link l, should be set up for this edge (Line 6). In this way, when all the demands are routed in the constructed graph, the network topology is also determined.

Algorithm 3. Algorithm to design topology

Require: Traffic matrix between each pair of racks $f_{ij}^{(r)}$
Ensure: Concrete link set $\boldsymbol{L}$ and Wavelength on each link $\boldsymbol{W}_l,\ l \in \boldsymbol{L}$
1: Initialize $G = \langle V, E\rangle$ as a full-mesh graph and each vertex denote a rack; f_{ij} in non-increasing order.
Suppose $\boldsymbol{F}$ is the set of $f_{ij}^{(r)}$ for all $i,\ j$
2: **for** each $f_{ij}^{(r)}$ in $\boldsymbol{F}$ **do**
3: Set weight on $\boldsymbol{G}$
4: Find a shortest path on $\boldsymbol{G}$ from vertex i to vertex j
5: **if** a path for $f_{ij}^{(r)}$ is found **then**
6: Concrete new link, say $\boldsymbol{L}'$ is the new links $\boldsymbol{L} \leftarrow \boldsymbol{L} \cup \boldsymbol{L}'$,
7: Wavelength w_l is the new wavelengths on link $', \boldsymbol{W}_l \leftarrow \boldsymbol{W}_l \cup w_l$
8: Update used capacity of links used by $f_{ij}^{(r)}, \boldsymbol{F} \leftarrow \boldsymbol{F} - f_{ij}^{(r)}$
9: **else**
10: **if** a concrete link can be selected to delete **then**
11: Delete the selected link.
12: Suppose the set of all the traffic on this selected links is $\boldsymbol{T}, \boldsymbol{F} \leftarrow \boldsymbol{T} \cup \boldsymbol{F}$
13: **else**
14: $f_{ij}^{(r)}$ is failed to route $F \leftarrow F - f_{ij}^{(r)}$
15: **end if**
16: **end if**
17: **end for**
18: **return** $\boldsymbol{L},\ \boldsymbol{W}_l$

The main architecture of the algorithm to design DDN topology is shown in Algorithm 3. Before each demand being routed, we should set weight to all edges on the graph for this demand (Line 3), with the purpose of leading demand to the route incurring low network cost. The flow-chart of setting weight to each edge is shown in Fig. 3. Since the cost of demand is the product of its traffic rate and the hop number on its route, we set 1 as the weight to the edges that will not block the on-routing demand while set infinite to the edges that cannot be used to carry on-routing demand (e.g. due to the maximum nodal degree constraint). If the link has not enough capacity to carry on-routing demand, its weight is set to be

$$Weight = wavelength\ capacity - remaining\ capacity \quad (21)$$

The key spirit is to use the link with more capacity so as to reduce the overflow demand quantity. When checking whether more wavelengths can be assigned to link l, we should find common wavelengths unused for both racks connected by link l. As to check whether a concrete link can be set up, we should not only check whether there remains at least one common wavelength that can be used by both end racks, but also see whether the nodal degree constraint is still satisfied after a new link is added into the network.

When one demand (say demand d from rack i to rack j) fails to be routed, *link adjusting* procedure will be triggered (Line 10). The only case resulting in demand routing failure is that the two ends are in two disconnected blocks in the network and no link can be added to connect these two blocks. There are two reasons that can prevent link being added. One is maximum nodal degree constraint that is violated after adding more links. In this case, we release one of rack i's adjacent links carrying minimum traffic (say link l_{ri}), and release one of rack j's adjacent links that uses one of the wavelengths used by l_{ri}. In this case, a link can be set up between rack i and rack j to carry demand d. If this link adjusting procedure is triggered, the demands carried by the released links should be rerouted at once, i.e. before all other un-routed demands. The other reason preventing link being added is no more wavelength remains. In this case, demand d will fail to be served due to the shortage of network resource.

4.4. Algorithm analysis

In this subsection, we analyze the convergence and approximation ratio of algorithms proposed in previous subsections through some proofs of theorem.

Lemma 3. *If there are enough wavelengths in the network, the link adjusting procedure will be triggered at most* $\left\lfloor \frac{R}{d+1} \right\rfloor + 1$ *times.*

Proof. Assume rack i and rack j are in two blocks, say block A and block B respectively. If link adjusting is triggered when a demand between these two blocks is routed, at least one of these two blocks is d-regular graph, who contains at least $d+1$ vertexes. Without loss of generality, we assume B is the d-regular block and C is the block formed by connecting A and B through link adjusting. If link adjusting should be triggered again to connect C and another block D, one of the following conditions must be satisfied:

1. D is a d-regular graph
2. C becomes a d-regular graph by adding edges and connecting different blocks

If condition 1 is the case, at least $d+1$ vertexes are merged into the new block. If condition 2 is the case, we can add at most one link in C/B to form a d-regular graph without introducing a new vertex into C. Therefore, there are at least $2d+2$ vertexes in C. It means that at least $d+1$ vertexes are merged by previous edge adjusting. Accordingly, at most $\left\lfloor \frac{R}{d+1} \right\rfloor + 1$ link adjustings are required to form a connected graph. □

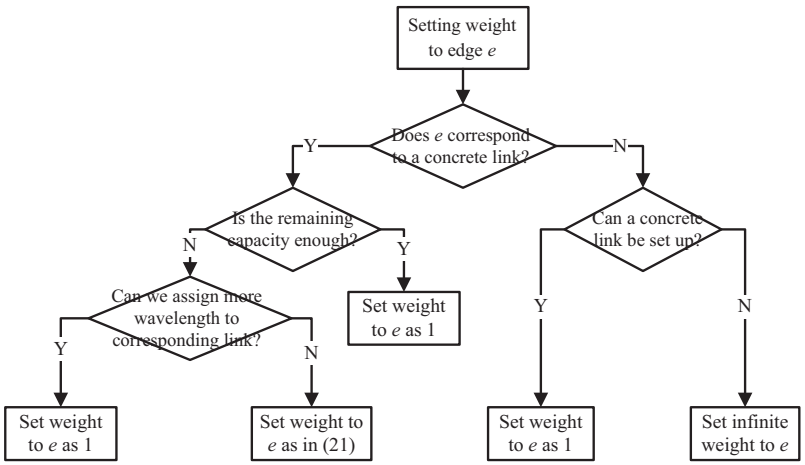


Fig. 3. Flowchart to set weight to edge e.

Lemma 3 guarantees the convergence of Algorithm 3. Another important issue of our algorithm is the approximation ratio. It can be answered by following lemma and theorem. Since the minimum k cut algorithm has no bound unless all the groups have the same size, all the following analyses are under the assumption that $H_i = H$ for all i. These results are valuable since it would be this case once the operator uses the identical access switch in its datacenter network (DCN) and fully utilizes all the input ports of each access switch.

Lemma 4. *If wavelength is enough and no link adjusting occurs, the upper bound of approximation ratio of our algorithm is*

$$H(R-1)\left[\sum_{i=1}^{M}\sum_{j=1}^{d}\frac{i}{(M-1)d+j}+\sum_{i=Md+1}^{R(R-1)/2}\frac{M+1}{i}\right] \tag{22}$$

where H is the number of VMs under each rack, d is the maximal nodal degree of each rack, R is the number of racks in the network and $M = \left\lfloor \frac{R(R-1)}{2d} \right\rfloor$

Proof. Without loss of generality, we assume $f_1 > f_2 > \cdots > f_{\frac{R(R-1)}{2}}$ is the traffic rate of each demand. In this case, it is obvious that

$$f_i < \frac{f}{i} \text{ for all } i = 1 \cdots \frac{R(R-1)}{2}$$

where

$$f = \sum_{i=1}^{\frac{R(R-1)}{2}} f_i$$

When the first d demands are routed, it is clear that direct link can be constructed for them and these demands are routed on a path with only 1 hop. Therefore, the network cost incurred by these d demands is

$$\sum_{i=1}^{d} f_i < f\sum_{i=1}^{d}\frac{1}{i}$$

When the second d demands are routed, they can construct a direct link (and be routed with 1 hop path) or be relayed by the links for the first d demands (in this case, the demand has 2-hops path). Accordingly, the network upper bound associating with these d demands is

$$\sum_{i=d+1}^{2d} f_i < 2f\sum_{i=d+1}^{2d}\frac{1}{i}$$

so on and so forth, the network cost upper bound incurred by the Mth d demands is

$$Mf\sum_{i=(M-1)d+1}^{Md}\frac{1}{i}$$

while the network cost that can be brought by the remaining demands is at most

$$(M+1)f\sum_{i=Md+1}^{R(R-1)/2}\frac{1}{i}$$

Accordingly, the upper bound of networks cost is

$$f\left(\sum_{i=1}^{M}\sum_{j=1}^{d}\frac{i}{(M-1)d+j}+\sum_{i=Md+1}^{R(R-1)/2}\frac{M+1}{i}\right).$$

On the other hand, in the minimum k-cut algorithm, the approximation rate is $H(R-1)$. The minimum traffic quantity in the network should be at least $\frac{f}{H(R-1)}$, and it is also the lower bound of network cost (i.e. all the traffic is routed on a path with only 1 hop). Accordingly, the upper bound of approximation ratio of our algorithm is

$$\frac{f\left(\sum_{i=1}^{M}\sum_{j=1}^{d}\frac{i}{(M-1)d+j}+\sum_{i=Md+1}^{R(R-1)/2}\frac{M+1}{i}\right)}{\frac{f}{H(R-1)}}$$

$$= H(R-1)\left[\sum_{i=1}^{M}\sum_{j=1}^{d}\frac{i}{(M-1)d+j}+\sum_{i=Md+1}^{R(R-1)/2}\frac{M+1}{i}\right] \quad \square$$

From Lemma 4, we can get following theorem:

Theorem 1. *If the wavelengths in the network are enough and nodal degree of each rack is large than 2, i.e. $d > 2$, the upper bound of approximation ratio of our algorithm is*

$$\frac{HR(R-1)}{d^2}\left[\sum_{i=1}^{M}\sum_{j=1}^{d}\frac{i}{(M-1)d+j}+\sum_{i=Md+1}^{R(R-1)/2}\frac{M+1}{i}\right] \tag{23}$$

Proof. For rack i,the minimum traffic rate on its adjacent links (say the link carried least demand is link (i,j)) should be less than $\frac{f}{d}$, where f is the traffic rate carried by the block that contains rack i. When link (i,j) is deleted and all the traffic on link (i,j) is rerouted, the maximum networks cost incurred by such traffic should be less than $\frac{R}{d}\frac{f}{d}$ and induce at most a factor $\frac{R}{d^2}$ to the approximation ratio in Lemma 4. □

It is worth noting that the factor incurred by edge adjusting approaches the bound when there are only two blocks merging into one. In addition, one of these two blocks is large enough and carrying almost all the traffic in the network while the other one is very small and carrying merely traffic. This condition is almost impossible since we route all the demands in a non-increasing order in terms of their traffic rate. The demands with larger traffic rate will get relatively shorter route and hence have large probability to be routed in the smaller size block. Furthermore, our algorithm pursues that all the demands are distributed in the network evenly. Accordingly, there is little probability to form such a unbalanced traffic distribution that the bound of approximation ratio is achieved.

4.5. Discussion

In previous subsections, we propose an algorithm to increase the network scalability by jointly optimizing VM placement and DDN topology based on concrete analysis. However, there remains one more issue in this algorithm. We do not consider the VM migration and topology adjusting cost (such as more energy consumption, more traffic in the network, and loss of delay sensitive flows) when we design the algorithm. Since we focus on how to improve the network traffic scalability in this paper, i.e. to adapt to more traffic in the network, which should be dealt in a long term perspective, while the VM migration and topology adjusting cost can only bring a short term cost to the network, we ignore such cost. Even if this cost cannot be ignored when it is amortized to the duration of adjacent network adjustings (either VM migration or topology adjusting), we can check whether the benefit brought by the network adjusting is cost efficient. If not, the adjusting will not be executed.

On the other hand, we can adopt some engineering methods to mitigate the cost brought by VM migration and topology adjusting. For example, we can use the second highest priority to delivery the flows for VM migration and reduce the migration duration (The highest priority is left for the delay sensitive flows.). We can also leverage the intermediate topology [20] to guarantee the transmission of delay sensitive flows.

5. Online algorithm

In previous sections, we formulated an MILP model for jointly optimizing VM placement and network topology. Due to the complexity of the MILP model, a heuristic is proposed for this problem based on Lagrange's relaxation decomposition. However, in realistic system, the traffic matrix is time-varying and we cannot execute Algorithm 1 and 3 and adjust the topology too frequently. One reason is that the complexity of minimum k cut algorithm is too high to be executed frequently (several minutes are required in a network with hundreds of hosts), while the other reason is that adjusting the topology too frequently may incur route oscillation problem. In this case, the correlation of sequential traffic matrices can be used to avoid route oscillation and reduce computation complexity.

5.1. Analysis for online algorithm design

To design an efficient online algorithm, we first briefly analyze the traffic characteristic in DCNs. Fig. 4 shows the incoming traffic rate and outgoing traffic rate of two typical VMs collected from a data warehouse hosted by IBM Global Services. It is shown that the traffic at each VM is relatively stable and occasionally a burst occurs.

Considering that an insignificant traffic change is to slightly modify the feasible range of **JOVT**, new optimal solution may be close to current solution. Accordingly, we can just do a local search to improve the network cost. On the other hand, it is not reasonable to adjust topology frequently since it may suffer from route oscillation problem. In our work, we constantly try to swap VMs (see definition in Section 5.2) to improve network cost. When there is a traffic burst in the network, some links may be congested. In this case, we trigger static joint optimization process.

5.2. Online optimization for VM placement

To clearly present our online optimization algorithm, we first briefly introduce some terms used in our algorithm.

Home rack: If VM k is directly connecting to rack i (i.e. VM k is under rack i), we say rack i is VM k's home rack, which is denoted by $i = R(k)$ and $k \in V(i)$.

Expected traffic to rack i: For VM k, its expected traffic to rack i is defined as

$$T(k,i) = \sum_{l \in V(i)} v_{kl}$$

Served traffic to rack i: For VM k, its served traffic to rack i is

$$T^{(s)}(k,i) = \sum_{l \in V(i)} v_{kl}^{(s)}$$

where $v_{kl}^{(s)}$ is the real traffic rate from VM k to VM l.

VM k's traffic cost: It is the network cost caused by VM k's traffic. When the topology and demand route are fixed, VM k's traffic cost can be calculated by

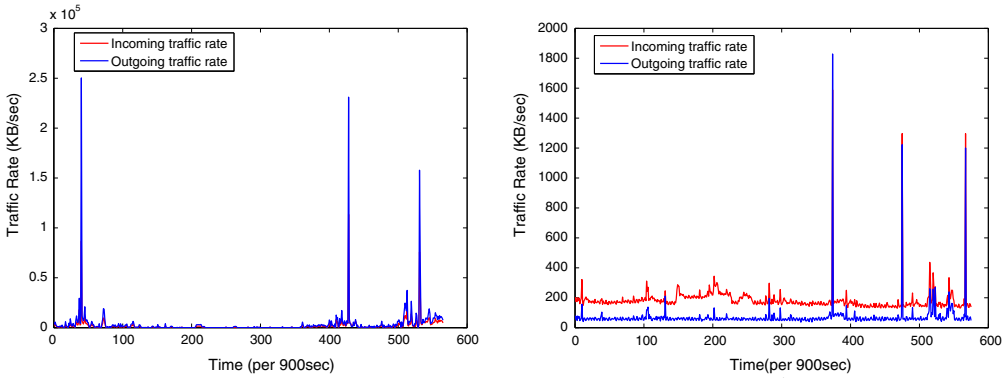


Fig. 4. Typical traffic rate for VMs in DCNs.

$$CT(k,i) = \sum_{j \neg = R(k)} h_{ij} T^{(s)}(k,j) + \gamma[T(k,j) - T^{(s)}(k,j)]$$

where γ is the large number in (8) and (12), h_{ij} is the hop number from rack i to rack j.

Swap VM k ***and VM*** l: It means swapping the location of these two VMs, i.e. placing VM k at VM l's host and vice versa.

Obviously, if $CT(k,i) + CT(l,j) > CT(k,j) + CT(l,i)$, placing VM k under rack j and VM l under rack i is better than placing VM k under rack i and VM l under rack j in terms of reducing the network cost. Based on this fact, we design a simple random algorithm to online optimize VM placement, which is shown in Algorithm 4. In this algorithm, solver only needs to continuously check whether there are VM pairs that can be swapped to reduce the network cost. If so, swap them, otherwise continue searching. Accordingly, this algorithm can be always executed in the backstage to keep optimizing the network traffic scalability. As in the static joint optimization, if the VM swapping cost is not negligible, we can take this cost into account at Line 3 in Algorithm 4.

Algorithm 4. Online optimization for VM placement

Require: Hop number between each rack $H = \{h_{ij}\}$. Current VM placement solution $R(k)$ for all VM k and $V(i)$ for all rack i.
1: **while** (1) **do**
2: Randomly find a pair of VMs, say VM k and VM l, such that $i = R(k) \neq R(l) = j$
3: **if** $CT(k,i) + CT(l,j) > CT(k,j) + CT(l,i)$ **then**
4: Swap VM k and VM l
5: **end if**
6: **end while**

It is clear that there will be no oscillation (if the traffic matrix is fixed) in this algorithm as every swap will reduce the network cost. Another advantage of this algorithm is the low computation complexity. Only four VMs' traffic costs need to be calculated in each iteration.

5.3. Trigger static joint optimization process

In realistic system, topology reconfiguration may incur route oscillation and a great amount of data loss. Therefore, the routes should be kept as stable as possible. Accordingly, there are only two cases that will trigger static joint optimization process and lead to topology reconfiguration in our algorithm.

The first case is that an overflow occurs in the networks. In this case, it is difficult to say which demands result in the overflow and find out how to relieve the overflow by swapping VMs. Therefore, static joint optimization should be triggered to deal with the overflow.

The other case is that a bad local optimal state being derived by online optimization method. In an online system, static algorithm proposed in Section 4 is also executed at the backstage of the system. When the difference between the realistic cost and the solution of static optimization algorithm excesses a predefined threshold, DDNs should be reconfigured according to the solution of static joint optimization algorithm.

5.4. Tenant enters and exits

In a realistic DCN, there may be new tenants entering into the network and some tenants may exit from the network. When a new tenant enters into a DCN, the operator can check whether there are empty hosts. If so, the new tenant can be assigned to an empty host which bring least network cost increment. Otherwise, the new tenant should be prohibited from entering due to the network capacity limitation.

When a tenant exits, we can easily set the corresponding host as an empty host and enable it to be occupied by other/new tenant. It should be noted that the local optimization iteration procedure discussed in previous subsections should be maintained when the tenants enter or exit the networks.

6. Evaluation

In this section, we first evaluate the performance of joint optimization, and compare it with the scheme that

only optimizes VM placement or topology. As the benchmark, we also show the performance in a random regular graph with random VM placement.

We do this evaluation from two perspectives. The first one is the static algorithm performance in the networks with different maximum nodal degree, and the other one is the performance in networks with different size. The reason is that nodal degree and network size are the two most important parameters for the algorithm. When we study the algorithm performance in the network with different maximum nodal degree, the network scale is fixed and follows OSA [3] which has 80 racks and 2560 hosts. Under each rack, there are 32 hosts. We study the cases that the maximum nodal degree of each rack is 3, 4 and 8 respectively. When the algorithm performance in different network scale is investigated, we fix the maximum nodal degree of each rack to be 4 (the same as OSA prototype in [3]) and the total hosts number to be 2560, while the network size changes as 40, 80, 160. All the hosts are evenly distributed under each rack.

In addition, we also evaluate the online algorithm performance. In Algorithm 4, the iteration should be executed all the time, which is difficult to realize in the simulator. Therefore, we first study the trade-off between the interaction times and the algorithm performance. After that, the online algorithm performance is evaluated by optimizing a serial of traffic matrices.

Our traffic was collected from a productive datacenter hosted by IBM Global Services. However, this dataset only provides the traffic incoming rate and outgoing rate of each VM, whereas we require traffic matrix for our algorithm. To this end, we applied the Gravity model [21] to estimate the traffic matrix. In the Gravity model, the traffic rate from VM i to VM j is determined by $D_{ij} = \frac{D_i^{out} D_j^{in}}{\sum_k D_k^{in}}$, where D_i^{out} is the total outgoing rate at VM i, and D_j^{in} is the total incoming rate at VM j.

6.1. Static algorithm in networks with different maximum nodal degree

6.1.1. Performance on network cost

We first study the performance of our joint optimization in terms of network cost. For this experiment, we assume there is enough wavelength at each rack and no traffic overflow could happen. Fig. 5 shows the results for joint optimization, single VM placement or topology optimization, as well as no optimization. From this figure, we make the following observations.

First, in all the three cases, the joint optimization has remarkable performance improvement compared with the single optimization on either VM placement or topology. For example, while VM placement optimization decreases the network cost by 5.69–39.48% and topology optimization reduces the network cost by 1.95–48.72%, the joint optimization leads to 16.59–58.78% network cost reduction.

Second, when the nodal degree of each rack is relatively small (e.g., 3 and 4), topology optimization will outperform VM placement optimization. On the other hand, optimizing VM placement will achieve better performance than optimizing topology when the nodal degree of each rack is relatively large (e.g., 8). This is because the smaller nodal degree would lead to larger optimization space for changing the topology. When the nodal degree becomes large, this optimization space is decreased and thus the benefit becomes smaller.

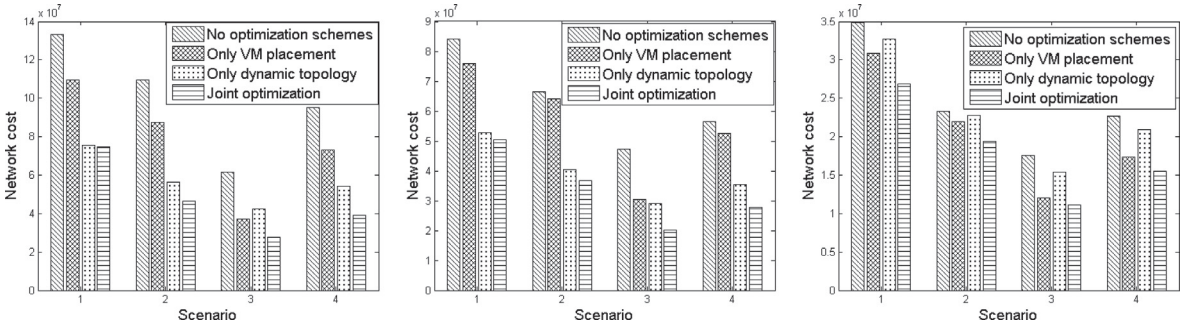
Third, the performance improvement brought by both VM placement and topology optimization will decrease when the nodal degree increases. This is again due to the reason that the optimization space is reduced when there are more links in the network. For example, in case the nodal degree of each rack is $R-1$, there would be no performance improvement with respect to topology optimization.

6.1.2. Performance on avoiding traffic overflow

To evaluate the performance of joint optimization on relieving traffic overflow, we repeat the above experiments but assume that there are only 80 wavelengths can be used by each rack and the capacity of each wavelength is 10G. We show the simulation results in Fig. 6.

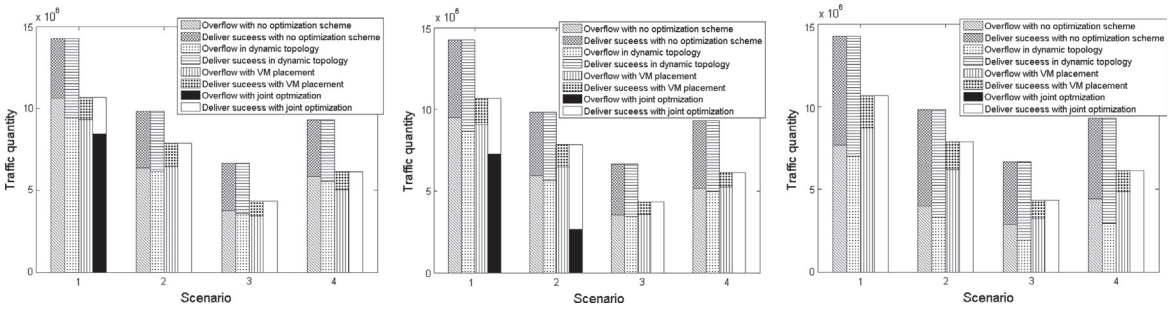
From this figure, we find that while both VM placement and topology optimization can relieve overflow in the network, joint optimization can reduce overflow more effectively, and it can completely avoid overflow in most of the cases.

We further find that in certain cases VM placement optimization may even result in a worse performance than random scheme (e.g. Scenario 2 in Fig. 6(b)). The reason is that it leverages the traffic aggregation to reduce the traffic



(a) Each rack is of Nodal degree 3 (b) Each rack is of Nodal degree 4 (c) Each rack is of Nodal degree 8

Fig. 5. Performance of reducing network cost vs. maximal nodal degree.



(a) Each rack is of no dal degree 3 (b) Each rack is of no dal degree 4 (c) Each rack is of no dal degree 8

Fig. 6. Performance of avoiding overflow vs. maximal nodal degree.

quantity delivered into the networks, thus making some links overused. Fortunately, the flexible link capacity introduced by topology optimization can balance the traffic aggregation problem incurred by VM placement. Therefore, joint optimization can yield a much better performance than only optimizing either VM placement or topology.

6.2. Static algorithm in networks with different size

To study how the algorithm performance is in networks with different size, we repeat the simulation in Section 6.1, but configure the networks with different size.

6.2.1. Performance on network cost

The algorithm performance of optimizing network cost with different network size is shown in Fig. 7. From this figure, we can see that joint optimization always obtains the minimum network cost regardless of the network size.

However, some observations different from those in Section 6.1.1 can be made. One observation is that with the same traffic matrix among different VMs, the network cost is increasing with the network size. This is because that with the same maximum nodal degree, the average hop number between each rack pair is increasing with the network size. Accordingly, it results in larger network cost.

Another observation is that with the increasing of the network size, the topology optimization has better performance than it in smaller networks, while the VM placement has worse performance. The reason is that there are less VMs under each rack in larger networks due to the fixed rack number. Hereby, there is less optimization space for the VM placement. On the other hand, more racks in the network provide more connection options to the topology optimization. Therefore, topology optimization performs better in the larger networks. In the extreme case that there is only one host under each rack, there is no optimization space for the VM placement.

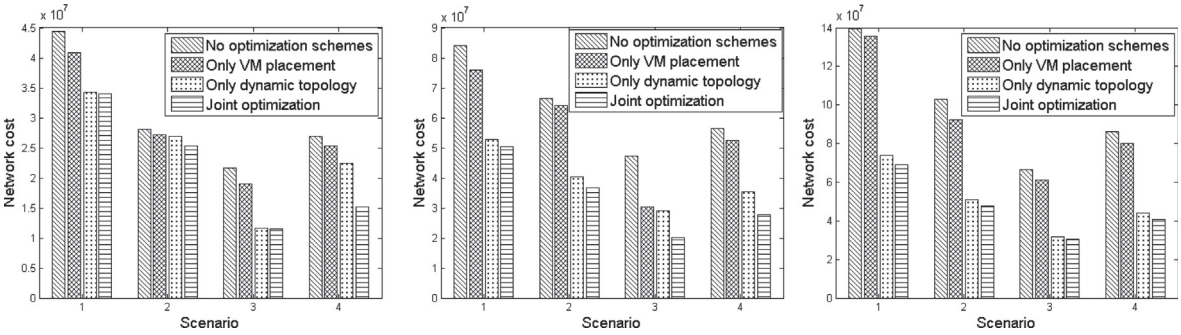
6.2.2. Performance on avoiding traffic overflow

Fig. 8 shows the algorithm performance to avoid traffic overflow in different scale of networks. When the network size is relatively small, less network resources are required to serve the traffic between VMs (which is discussed in last subsection), joint optimization can avoid the traffic overflow in all the scenarios. With the increasing of network size, more network resources are required to serve all the traffic. When there 80 racks in our experiment, overflow may occur in some scenarios. Furthermore, when the rack number increases to be 160, overflow occurs in all the scenarios. However, joint optimization can avoid overflow or minimize the overflow regardless of the experiment settings.

6.3. Online algorithm evaluation

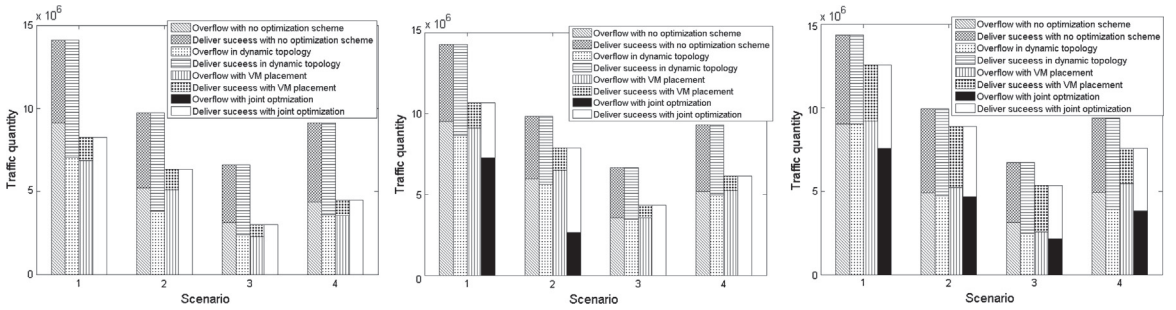
6.3.1. Iteration times/performance trade-off

Fig. 9 shows how the network cost changes with the interaction times in Algorithm 4. In this figure, the initial network configuration (includes network topology, VM placement and traffic routing) is obtained by the static



(a) 40 rack sin the network (b) 80 racks in the network (c) 160 racks in the network

Fig. 7. Performance of reducing network cost vs. network size.



(a) 40 racks in the network (b) 80 racks in the network (c) 160 racks in the network

Fig. 8. Performance of avoiding overflow vs. network size.

algorithm with traffic matrix in previous time slot as input. After that, we route all the traffic in current time slot with the initial network configuration and trigger the Algorithm 4. It can be observed that online iteration can greatly reduce the network cost even though we do not trigger the static algorithm. On the other hand, we can also observe that it is hard to greatly reduce the network cost further after more than 900 iterations are executed.

6.3.2. Performance of online algorithm

To evaluate the performance of our online algorithm, we run it on 20 traffic snapshots in consecutive time slots (each slot lasts 900 s) extracted from our data source. The algorithm begins with a static joint optimization process to work out an initial topology and VM placement solution. After that we trigger the Algorithm 4. In each slot, we repeat the iteration process 1000 times if no traffic overflow happens on the topology used in the previous slot. Otherwise, we trigger the static joint optimization algorithm discussed in Section 4 once overflow occurs. As a benchmark, we not only test the network cost without VM swapping iteration and we only trigger joint optimization when overflow occurs, but also run the static joint optimization algorithm.

The simulation result is shown in Fig. 10. From the figure, we can see that the online iteration process can effectively reduce the network cost compared with we

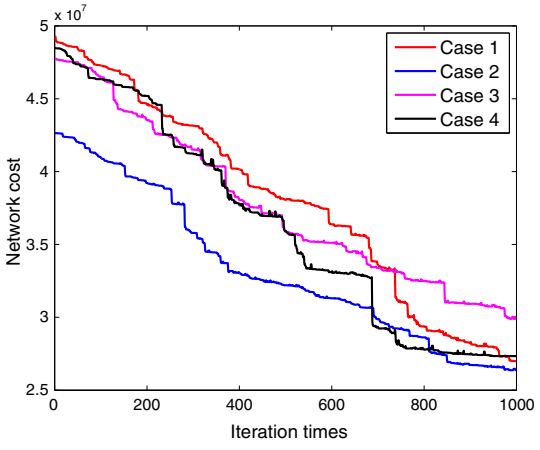


Fig. 9. Iteration times vs. algorithm performance.

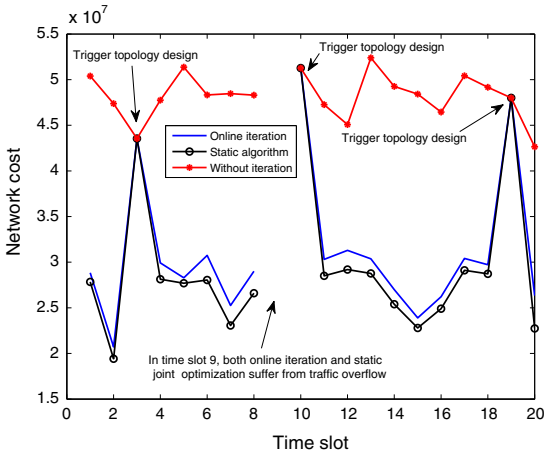


Fig. 10. Performance for online iteration algorithm.

only trigger joint optimization when overflow occurs. Also, our online algorithm only triggers topology reconfiguration a few times, so that it greatly reduces the overhead compared with running static joint optimization in every time slot. Furthermore, online algorithm can get a solution very close to the static joint optimization algorithm. This is because that the input/output traffic is relatively stable for a rack, and burst occurs in very small fraction of the racks. Therefore, migrating these VMs together can greatly reduce the network cost without changing the network topology.

7. Related Work

We briefly review the related works of our study in three aspects: dynamic topology, VM placement problem and joint optimization.

To deal with the unbalanced traffic in datacenter networks, many works leveraged the dynamic topology, though OSA [3] provides the most flexible topology. c-Through [14] proposes a hybrid packet and circuit switch architecture which uses optical circuit switching to fix the high bandwidth requirement in part of the network if it is required. Helios [4] also designed a hybrid packet and circuit switch architecture in the DCNs to manage the unbalanced traffic, but its objective is to find the opti-

mal trade-off between the number of switching elements, cabling, cost, and power consumption.

The main objective of VM placement is to save resource in DCNs, such as CPU, RAM and network. In addition to the foundation of our work [10,13] is also working on enhancing the network traffic scalability. Different from [10,13] is a network-aware algorithm which takes the local physical resources, such as CPU and memory into account. On the other hand, [22] focused on improving the datacenter efficiency by sharing the resources of physical servers. Since VM placement problem is usually a NP-hard problem, all these works solved their problems by designing an efficient heuristic.

On the joint optimization to enhance the DCN traffic scalability, [9] is a representative. Different from our work, it jointly optimizes the VM placement and traffic routing in the DCNs under the assumption that the demands can be served through multipath routing. In addition, Ref. [9] focuses on the case that the network topology is fixed. Usually, multipath routing is not allowed in DCNs since it may result in TCP disordering problem and degrade the traffic throughput. On the other hand, when the topology is dynamic, the Markov approximation method is difficult to deploy since the solution space is much larger than that when the topology is fixed. Even when there are only 20 racks in the network, the solution space should be over 12 billion times larger than that with fixed topology.

8. Conclusion

In this paper, we have jointly optimized VM placement and topology to improve traffic scalability in dynamic datacenter networks. We formulated this problem as an MILP and leveraged Lagrange's relaxation to analyze it. Based on the analysis, we proposed an efficient heuristic algorithm and derived a theoretical bound of the proposed solution. Furthermore, we designed an iteration-based online algorithm for time-vary traffic scenario. Simulation results show that our algorithm yields a much lower network cost (i.e., higher traffic scalability), and our online algorithm greatly improves traffic scalability without frequently reconfiguring topology and traffic route.

Acknowledgements

This work was supported in part by National Basic Research Program of China (973) under Grants 2013CB329103, 2011CB302601, 2014CB340303, HKRGC-ECS 26200014, Huawei Noah's Ark Lab, NSFC Fund (61271165, 61271171, 61201129, 61301153 and 91438117), PCSIRT Fund, 111 Project B14039, and the Fundamental Research Funds for the Central Universities.

References

[1] Q. Duan, Y. Yan, A. Vasilakos, A survey on service-oriented network virtualization toward convergence of networking and cloud computing, IEEE Trans. Network Service Manage. 9 (4) (2012) 373–392, http://dx.doi.org/10.1109/TNSM.2012.113012.120310.

[2] Cisco Cloud Computing – Data Center Strategy, Architecture, and Solutions. <http://www.cisco.com/web/strategy/docs/gov/CiscoCloudComputing_WP.pdf>.

[3] K. Chen, A. Singla, A. Singh, K. Ramachandran, L. Xu, Y. Zhang, X. Wen, Y. Chen, OSA: an optical switching architecture for data center networks with unprecedented flexibility, in: NSDI 2012, 2012.

[4] N. Farrington, G. Porter, S. Radhakrishnan, H.H. Bazzaz, V. Subramanya, Y. Fainman, G. Papen, A. Vahdat, Helios: a hybrid electrical/optical switch architecture for modular data centers, in: SIGCOMM 2010, 2010.

[5] C. Leiserson, Fat-trees: universal networks for hardware-efficient supercomputing, IEEE Trans. Comput. C-34 (10) (1985) 892–901, http://dx.doi.org/10.1109/TC.1985.6312192.

[6] A. Greenberg, J.R. Hamilton, S.K.N. Jain, P.L.C. Kim, D.A. Maltz, P. Patel, S. Sengupta, Vl2: a scalable and flexible data center network, in: SIGCOMM 09, 2009.

[7] C. Guo, G. Lu, D. Li, X.Z.H. Wu, Y. Shi, C. Tian, Y. Zhang, S. Lu, Bcube: a high performance, server-centric network architecture for modular data centers, in: SIGCOMM 09, 2009.

[8] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, S. Lu, Dcell: a scalable and fault-tolerant network structure for data centers, in: SIGCOMM 08, 2008.

[9] J.W. Jiang, T. Lan, S. Ha, M. Chen, M. Chiang, Joint VM placement and routing for data center traffic engineering, in: Proc. 31th IEEE Conf. on Computer Communication (INFOCOM), 2012.

[10] X. Meng, V. Pappas, L. Zhang, Improving the scalability of data center networks with traffic-aware virtual machine placement, in: INFOCOM 2010, 2010.

[11] N. Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, A. Vahdat, PortLand: a scalable fault-tolerant layer 2 data center network fabric, in: SIGCOMM 09, 2009.

[12] J.P. Srikanth Kandula, P. Bahl, Flyways to de-congest data center networks, in: 8th ACM Workshop on Hot Topics in Networks, 2009.

[13] O. Biran, A. Corradi, M. Fanelli, L. Foschini, A. Nus, D. Raz, E. Silvera, A stable network-aware VM placement for cloud systems, in: 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), 2012, 2012, pp. 498–506 (http://dx.doi.org/10.1109/CCGrid.2012.119).

[14] G. Wang, D.G. Andersen, M. Kaminsky, K. Papagiannaki, T.S.E. Ng, M. Kozuch, M. Ryan, c-Through: part-time optics in data centers, in: SIGCOMM 2010, 2010.

[15] M.G. Rabbani, R.P. Esteves, M. Podlesny, G. Simon, L.Z. Granville, R. Boutaba, On tackling virtual data center embedding problem, in: IFIP/IEEE IM 2013, 2013.

[16] K. Aardal, Lattice Basis Reduction and Integer Programming, Tech. rep., UU-CS-1999-37, Universiteit Utrecht, 1999. <http://www.cs.uu.nl/research/techreps/repo/CS-1999/1999-37.pdf>.

[17] K. Obraczka, P. Danzig, Finding Low-Diameter, Low Edge-Cost, Networks.

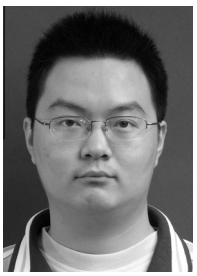
[18] Knapsack Problem. <http://en.wikipedia.org/wiki/Knapsack_problem>.

[19] G.H. Hardy, J. Littlewood, G. Plya, Inequalities, Cambridge University Press, 1952.

[20] Y. Zhao, S. Wang, S. Luo, H. Yu, S. Xu, X. Zhang, Dynamic topology management in optical datacenter networks, in: IEEE Globecom 2014, 2014.

[21] Y. Zhang, M. Roughan, N. Duffield, A. Greeberg, Fast accurate computation of large-scale ip traffic matrices from link loads.

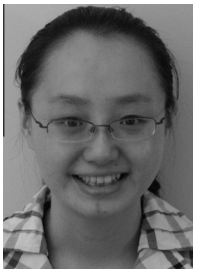
[22] H. Jin, D. Pan, J. Xu, N. Pissinou, Efficient vm placement with multiple deterministic and stochastic resources in data centers, in: 2012 IEEE Global Communications Conference (GLOBECOM), 2012, pp. 2505–2510 (http://dx.doi.org/10.1109/GLOCOM.2012.6503493).



Yangming Zhao is a Ph.D. candidate in University of Electronic Science and Technology of China (UESTC). He received his B.S. degree in Communication Engineering from UESTC in July 2008. His research interests include network optimization and data center networks.

Yifan Huang is a Ph.D. candidate in Department of Electronic and Computer Engineering at the Hong Kong University of Science and Technology. She received her B.S. degree in Communication Engineering from University of Electronic Science and Technology of China (UESTC) in July 2013. Her research interests include oblivious routing and network on chip.

Kai Chen is an Assistant Professor with Department of Computer Science and Engineering at the Hong Kong University of Science and Technology. He received a Ph.D. degree in Computer Science from the Northwestern University in June 2012. He received B.S. and M.S. degrees in Computer Science both from the University of Science and Technology of China in 2004 and 2007 respectively.



Minlan Yu is an assistant professor in the Computer Science Department at University of Southern California. She co-lead the Networked Systems Lab with Ramesh Govindan and Ethan Katz-Bassett. She is interested in data networking, distributed systems, enterprise and data center networks, network virtualization, and software-defined networking. She received Ph.D. from Princeton University in August 2011, advised by Jennifer Rexford. After that, she was a postdoctoral scholar working with Ion Stoica at UC Berkeley for one year.

Sheng Wang is served as a professor in University of Electronic Science and Technology of China (UESTC). He is a Senior Member of Communication Society of China, a member of IEEE, a member of ACM and a member of CCF (China Computer Federation). He received his B.S. degree in Electronic Engineering from UESTC in July 1992. He received his M.S. degree and Ph.D. degree in Communication Engineering from UESTC in 1995 and 2000, respectively. His research interests include planning and optimization of wire and wireless networks, next generation of Internet and next generation optical networks.

Dongsheng Li is an associate professor in School of Computer, National University of Defense Technology (NUDT), China. He received his B.S. degree in Computer Science from NUDT in 1999, followed by his Ph.D. in 2005. His graduate work was supervised by Prof. Xicheng Lu, a famous computer scientist in China and an academician of Chinese Academy of Engineering. He was a visiting student at Department of Computing, HongKong Polytechnique University from March 2005 to March 2006, under supervision of Dr. Jiannong Cao. After his Ph.D. work, he joined National Laboratory for Parallel and Distributed Processing (PDL) in NUDT as an assistant professor, and was promoted as associate professor in December 2007.