Towards Robust Green Virtual Cloud Data Center Provisioning

Yang Yang, Xiaolin Chang, *Member, IEEE*, Jiqiang Liu, *Member, IEEE*, and Lin Li

Abstract—Cloud data center (CDC) network virtualization is being regarded as a promising technology to provide performance guarantee for cloud computing applications. One critical issue in CDC network virtualization technology is virtual data center (VDC) embedding, which deals with the CDC physical resource allocation to virtual nodes (virtual switches and virtual servers) and virtual links of a VDC. When node and link constraints (including CPU, memory, network bandwidth, and network delay) are both taken into account, the VDC embedding problem is NP-hard, even in the offline case. Node heterogeneity and CDC network scale bring challenges to the VDC embedding. This paper aims to embed a VDC in a robust and green way. We propose two effective, computation-efficient and energy-efficient embedding algorithms. Extensive simulations under various network scales and topologies are carried out to compare the proposed algorithms with the existing VDC embedding algorithms in terms of the VDC acceptance ratio, the long-term revenue of the cloud service provider (CSP), the CDC's long-term energy consumption in light-load CDCs, and in terms of CSP's long-term revenue in heavy-load CDCs.

Index Terms—Cloud data center, virtual data center embedding, energy consumption, mixed integer programming

1 Introduction

Cloud data centers are being a promising platform for running network-intensive applications. For example, cloud-based Hadoop [1] is proposed to achieve a better cost-performance ratio in parallel processing of huge datasets. Experiment results demonstrated that cloud-based Hadoop could effectively process the large distributed data required for big data analysis [2], [3]. Several commercial infrastructure-as-a-service (IaaS) cloud providers are now offering manageable Hadoop services [4], e.g., Amazon Elastic Compute Cloud (EC2) system [5], Google Compute Engine [6], and Microsoft Windows Azure [7].

The performance of network-intensive applications depends not only on computing and storage resources of physical servers, but also on the available network resources on which they are deployed [8], [10]. Researchers explored the improvement of these applications' performance through designing datacenter network architectures [11] and designing network-aware virtual machine placement mechanisms [30]. Network virtualization technology was proposed to address the performance problems existing in Internet Service Provider (ISP) networks. Recently, this technology is extended to cloud data center (CDC) networks in order to offer assured performance to cloud applications [12] and to alleviate network attacks from malicious tenants' VMs [13]. CDC network virtualization allows multiple virtual CDC networks to cohabit a shared CDC physical network without interfering with each other. A tenant can use "virtual data center network" (VDC) as a means of specifying their resource requirements to cloud service providers (CSP).

There are several definitions of VDC [9]. As in [12], this paper defines VDC to represent a virtual network (including virtual servers, virtual switches/routers, and virtual links) allocated to a tenant. Each component of the VDC has its own physical resource demands. One of the key issues to be tackled in CDC network virtualization is the VDC embedding (VDCE) problem. Many virtual network embedding (VNE) algorithms, seeing [12] and [21] and references therein, have been proposed for ISP network virtualization. However, the following two unique features for VDC embedding problems challenge the existing VNE algorithms' ability in embedding VDCs.

1) *Node types*. ISP networks/VNs mostly consider homogeneous physical/virtual nodes, namely packet forwarding elements (e.g., routers/switches). But there exist different types of physical/virtual nodes in CDC/VDC networks, including servers, routers/switches, storage nodes and so on. Thus, different from VNE problem which consists of virtual node mapping and virtual link mapping, VDCE problem usually consists of at least three kinds of mappings: virtual server mapping, virtual switch mapping, and virtual link mapping.
2) *Physical network scale*. There are usually several hundreds of nodes in ISP backbones, e.g., 487, 624, and 862 nodes in AT&T, Level3, and Verio ISPs, respectively [14]. However, current CDCs may consist of thousands of nodes, e.g., around 12,000 servers in one Google Compute cluster [15] and over 320,000 servers in Amazon's US East data center [16]. The scalability issue cannot be ignored in a VDCE algorithm. Our experimental results (see Appendix A, which can be found on the Computer Society

• *The authors are with Beijing Jiaotong University, P.R. China. E-mail: {13120443, xlchang, jqliu, lilin}@bjtu.edu.cn.*

Manuscript received 4 Dec. 2014; revised 11 Apr. 2015; accepted 8 July 2015. Date of publication 22 July 2015; date of current version 7 June 2017. Recommended for acceptance by C. Mastroianni, S. U. Khan, and R. Bianchini. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below. Digital Object Identifier no. 10.1109/TCC.2015.2459704

Digital Library at http://doi.ieeecomputersociety.org/10.1109//TCC.2015.2459704) of some existing VNE algorithms, including R-ViNE, Artificial-intelligence-based VNE algorithms, suffer from the scalability problem.

Electrical energy cost is one of the major operation costs of a CDC [17]. With the unprecedented growth in CDCs, reducing energy consumption is being a primary challenge for CSPs. VDC embedding is a complex task and it may result in high energy consumption if it is not addressed effectively.

This paper aims to explore effective and efficient VDCE approaches with the objective to produce CSP's high long-term revenue (defined in Equation (2)), while minimizing energy consumption in light-workload CDCs and minimizing embedding cost in heavy-workload CDCs. We propose two VDCE approaches. The first approach includes two phases: 1) mapping virtual servers by using NSS approach described later, and 2) mapping virtual switches and virtual links coordinately. The second approach includes three phases: 1) mapping virtual servers as in the first approach, 2) mapping virtual switches in an energy-aware and random way, and 3) mapping virtual links by using a green Breadth First Search algorithm (BFS). Unless otherwise specified, embedding cost of a VDC represents the amount of physical resources allocated to this VDC.

We summarize the main contributions as follows:

1) Propose a Nearest-edge-Switch approach to map virtual Servers (NSS). NSS reduces energy consumption and embedding cost by using those physical servers, which are as close as possible, to host virtual servers. NSS can not only reduce the complexity caused by the CDC's large scale and reduce the number of active servers but also implicitly facilitate the embedding cost reduction in virtual switches and virtual links mapping. Some authors have mentioned the importance of locality in designing mapping VDCE algorithms, such as in [8]. But no detail about how to apply locality information to allocate resources was given.
2) Propose a *Joint* mapping approach to map virtual Switches and *Links* (denoted as JointSL), used in the first VDC embedding approach (denoted as NSS-JointSL). We formulate the problem of mapping virtual switches and links as a form of Mixed Integer Programming Multi-Commodity Flow (MIP-JointSL) problem. JointSL applies the CPLEX solver [36] to solve each MIP-JointSL model. Although the abilities of CPLEX solver-based exact algorithms for ultra-large-scale VNE problem were challenged such as in [30], they performed well in scenarios where both CDC and VDC are not too large. In the CDCs of 16,000 servers, there are only about 400 physical switches and 400 physical links. Our experiment results indicate that JointSL algorithm can produce perfect results in a reasonable time. To the best of our knowledge, there is no published literature which explored mapping virtual switches and virtual links in an explicitly joint way in order to minimize the embedding energy consumption while trying to minimize embedding cost.
3) Investigate the effectiveness and efficiency of Breadth-First-Search (BFS) algorithm in virtual link mapping. A Green Breadth First Search algorithm is proposed. We also investigate two kinds of GBFS algorithms, namely Switch-NSS-GBFS and Server-NSS-GBFS. The only difference between them is that Switch-NSS-GBFS starts from the physical switch when mapping a virtual link, but Server-NSS-GBFS starts from the physical server when mapping a virtual link.
4) We perform a comprehensive comparative study of NSS-JointSL, NSS-GBFS (Switch-NSS-GBFS and Server-NSS-GBFS) with the existing VDCE algorithms in various scenarios, including different CDC network topologies (single-root [18] in Fig. 2, fat-tree [19] in Fig. 3, and clos [20] in Fig. 4), different sized substrate networks (small-sized, medium-sized and large-sized networks), different sized VDCs, and different VDC topologies (see Fig. 5). To the best of our knowledge, no performance evaluation has been carried out in various scales of networks. The existing VDCE algorithms' evaluations were carried out only in the small-sized or medium-sized scenarios. It is possible that one VDCE algorithm can perform better in a particular scenario, but may not work well when there is a change in the environment settings.

It is noted that this paper focuses on how to allocate virtualized resources to tenant requests at the request admission control time. However, software and hardware failure are a fact of life in CDCs [42], which may affect tenants' Quality of Service (QoS). Furthermore, elasticity is a key feature of cloud services, allowing resource requirements to grow and shrink dynamically. Migration based mechanisms [32], [41] have been proposed to deal with these situations. Our algorithms can work seamless with these migration-based mechanisms to provide flexible and fault-tolerant services to tenants.

The rest of this paper is organized as follows. Section 2 presents the background and related work. In Section 3, we present NSS-JointSL and NSS-GBFS. Section 4 presents the performance evaluation of the proposed algorithms. Finally, we conclude the paper and discuss directions for future work in Section 5.

2 Background and Related Work

This section first presents the VDCE problem and typical CDC topologies. Then we discuss existing researches related to VDC embedding algorithms.

2.1 Network Model and Problem Description

Both the substrate/physical data center network and the VDC network are modeled as weighted graphs denoted by $G_P = (N^{PS}, N^{PH}, E^P)$ and $G_V = (N^{VS}, N^{VH}, E^V)$, respectively. Here N^{PH}/N^{VH} is the set of physical/virtual servers, N^{PS}/N^{VS} is the set of physical/virtual routers/switches, and E^P/E^V is the set of physical/virtual links.

Each physical node $n^P \in N^{PS} \cup N^{PH}$ may be associated with memory, processing power, storage resources and so on. This paper only considers the processing power of servers and the memory of switches in order to simplify the

description and then make it easy to understand the proposed algorithms. Note that the VDCE algorithms proposed in this paper can be directly applied to the situation where a VDC has demands for other kinds of physical resources. For example, for physical server, it is done by simply replacing CPU constraint in our proposed algorithms with all server-related resource constraints.

We define $N^P = N^{PS} \cup N^{PH}$ and $N^V = N^{VS} \cup N^{VH}$. Each physical link $e^P(v^P, w^P) \in E^P$ between physical nodes (v^P, w^P) is associated with bandwidth capacity. All the physical resources (e.g. bandwidth and CPU) in G_P are limited. Usually, the virtual node's QoS requirements include the CPU demand and a preferred value d^{n^V} expressing how far a virtual node $n^V \in N^V$ can be placed from the specified location $l(n^V)$. The QoS requirements of a virtual link $e^V(v^V, w^V) \in E^V$ between virtual nodes (v^V, w^V) include the bandwidth requirement and a delay demand. Embedding a VDC request is defined as a mapping from G_V to G_P with the constraints:

1) Each virtual server is mapped to a physical server in a one-to-one manner, and the virtual server QoS requirements are satisfied.
2) Each virtual switch is mapped to a physical switch in a one-to-one manner, and the virtual switch QoS requirements are satisfied.
3) Each virtual link $e^V(v^V, w^V)$ is mapped to a physical path (an unsplittable model) or a flow (a splittable model) in G_P between physical nodes v^V and w^V, respectively, with at least two requirements. One is that the $e^V(v^V, w^V)$ bandwidth requirement is below the total available bandwidth of the physical path or the flow. The second is that the delay constraint of the virtual link is met.

The available resource amount $A_N(n^P)$ of a physical node n^P is defined as $A_N(n^P) = c(n^P) - \sum_{\forall n^V \uparrow n^P} c(n^P)$. Here $n^V \uparrow n^P$ denotes that virtual node n^V is hosted on the physical node n^P. For a switch μ, $c(u)$ denotes a node's memory resource. For a server μ, $c(u)$ denotes a node's CPU resource. The available bandwidth $A_E(e^P(v^P, w^P))$ of a physical link $e^P(v^P, w^P)$ is defined as $b(e^P(v^P, w^P))$ minus the total bandwidth used by virtual links that pass through $e^P(v^P, w^P)$.

Objective. As in [22], [23], [24], [25], we assume that:

1) Whenever a physical link is turned off, energy is saved in the pair of interfaces/ports on its endpoints (also turned off).
2) A physical node (switch/server) is turned off only when all the network ports are turned off, namely all the physical links connected to this node are turned off.
3) If a physical node is unused, then it is simply turned off and has zero power consumption.

Table 1 gives the definition of the variables, used in the following.

Assume that there are K active VDCs in the interval T = t2-t1, the kth VDC (denoted as G_{Vk}) is served by CSP from time $t^{start}_{G_{Vk}}$ and leaves at time $t^{start}_{G_{Vk}} + T^d_{G_{Vk}}$. Equation (1) defines the revenue $R(G_{Vk})$ of serving G_{Vk} at unit time.

TABLE 1
Variable Definition

Term	Definition
λ_{u^P}	A binary variable.$u^P \in N^P$. Denote whether a physical node is active or not before embedding G_V: 1 means active; otherwise 0.
$\eta_{u^P v^P}$	A binary variable. Denote whether a physical link is active or not before embedding G_V: 1 represents active; otherwise 0. $e^P(u^P, v^P) \in E^P$.
$g^{e^P}_{e^V}$	A binary variable. Its value is 1 if $e^P(u^P, v^P)$ is on the path hosting the virtual link of e^V. Otherwise, it is set to 0. $e^P(u^P, v^P) \in E^P$.
$z^{u^P}_{u^V}$	An integer decision variable. $u^P \in N^P$ and $u^V \in N^V$. Its value is 1 if virtual node u^V is mapped to physical node u^P; otherwise, set to 0.
$x_{u^P v^P}$	A binary variable.$u^P, v^P \in N^P$. After embedding G_V, its value is 1 if $e^P(u^P, v^P)$ is active; otherwise, it is set to 0.
y_{u^P}	A binary variable, $u^P \in N^P$. After embedding G_V, its value is 1 if the physical node u is active; otherwise it is set to 0.
$NumAS(s)$	The number of active physical servers directly connected to edge/ToR nodes.
$var(NumAS)$	The variance of $NumAS$.

$$R(G_{Vk}) = \sum_{e^V \in E^{Vk}} b(e^V) + \alpha_{switch} \cdot \left(\sum_{n^V \in N^{VSk}} c(n^V) \right) + \alpha_{server} \cdot \left(\sum_{n^V \in N^{VHk}} c(n^V) \right). \tag{1}$$

Here, α_{server} and α_{switch} are the weights to determine the relative importance of physical server resources, the physical switch resources and bandwidth resources. Equation (2) gives the CSP's average revenue over T.

$$R^{Inp}_{avg} = \frac{\sum_{k \in K} \sum_{t=0}^{T} \left(R(G_{Vk}) \cdot L_{G_{Vk}} \right)}{T}, \tag{2}$$

where

$$L_{G_{Vk}} = \begin{cases} T, & t^{start}_{G_{Vk}} < t_1 \quad \text{and} \quad T < T^d_{G_{Vk}} \\ T - (t^{start}_{G_{Vk}} - t_1), & t^{start}_{G_{Vk}} \geq t_1 \quad \text{and} \quad T < T^d_{G_{Vk}} \\ T^d_{G_{Vk}} - (t^{start}_{G_{Vk}} - t_1), & t^{start}_{G_{Vk}} \geq t_1 \quad \text{and} \quad T \geq T^d_{G_{Vk}} \\ T^d_{G_{Vk}} + (t^{start}_{G_{Vk}} - t_1), & t^{start}_{G_{Vk}} < t_1 \quad \text{and} \quad T \geq T^d_{G_{Vk}}. \end{cases} \tag{3}$$

Each virtual link $e^{Vk}_i(s^{Vk}_i, t^{Vk}_i)$ is considered as a commodity from source physical node to destination physical node, $\left(\forall i \in \left\{1 \ldots \left|E^{Vk}\right|\right\}, s^{Vk}_i, t^{Vk}_i \in N^{Vk}\right)$. The energy consumption $E_u(G_{Vk})$ for serving the G_{Vk} in unit time is defined as

$$E_u(G_{Vk}) = \sum_{v^P \in N^P} P^{v^P}_1 \cdot (y_{v^P} - \lambda_{v^P}) + \sum_{e^P(v^P, w^P) \in E^P} P^{e^P}_2 \cdot (x_{v^P w^P} - \eta_{v^P w^P}) + \sum_{i \in \left\{1 \ldots \left|E^{Vk}\right|\right\}} \left(P^{P_{ki}}_3 \cdot c\left(s^{Vk}_i\right) + P^{t_{ki}}_3 \cdot c\left(t^{Vk}_i\right) \right). \tag{4}$$

Here, P^v_1, $P^{e^S}_2$ and P^u_3 are weights. P^u_3 represents the power consumption of unit physical resource utilization of physical node u in unit time. Equation (5) defines the CDC's

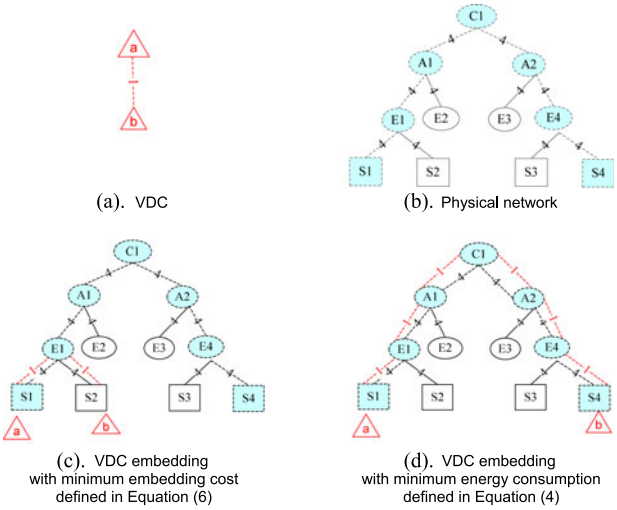


Fig. 1. An example of two VDC embeddings.

average energy consumption over T:

$$E_{avg}^{InP} = \frac{\sum_{k\in K}\sum_{t=0}^{T}\left(E_u(G_{Vk})\cdot L_{G_{Vk}}\right)}{T}. \tag{5}$$

Equation (6) defines the cost of serving G_{Vk} in terms of the consumed physical CPU, memory and link bandwidth resources:

$$RC(G_{Vk}) = \sum_{e^S\in E^S}\sum_{i}\left(\eta_{link}(e^S,i)\cdot f_{e^S}^{i}\right) + \eta_{switch}\cdot\sum_{n^V\in N^{VSk}} c(n^V) + \sum_{n^V\in N^{Vk}} c(n^V). \tag{6}$$

Here $\eta_{link}(e^P,i)$ and η_{switch} are weight, $f_{e^S}^{i}$ is defined to denote the total amount of flows from physical node v to w on the physical link $e^S(v,w)$ for the VDC's ith virtual link, $i\in\{1\ldots|E^{Vk}|\}$. The average cost over T is defined in Equation (7):

$$RC_{avg}^{Inp} = \frac{\sum_{k\in K}\sum_{t=0}^{T}\left(Cost(G_{Vk})\cdot L_{G_{Vk}}\right)}{T} \tag{7}$$

Figs. 1c and 1d, respectively, illustrates two ways of embedding a VDC request (in Fig. 1a) on the CDC (in Fig. 1b). A shaded node is active node and the unshaded node means that this node is inactive.

2.2 Data Center Network Topology

Various network architectures for data centers have been proposed and implemented in both academia and industry. This section focuses on single-root Tree topology, Clos topology and fat-tree topology, which are often used to evaluate cloud resource allocation mechanisms.

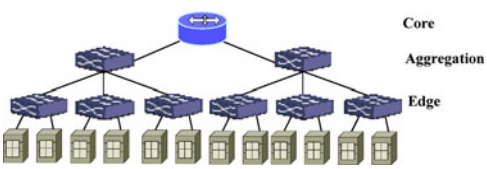


Fig. 2. Single-root data center topology.

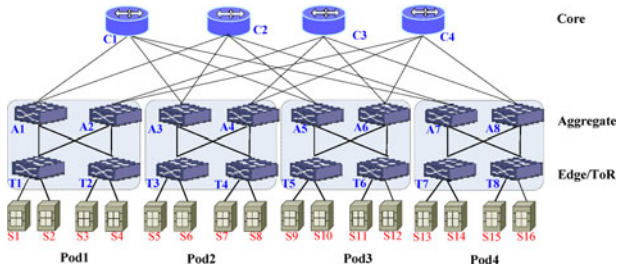


Fig. 3. Fat-Tree data center topology.

Fig. 2 shows a three-level tree topology. The tree root is core router, connected to aggregation switches at the second level. The tree leaves are physical servers, connected to top-of-rack (ToR) switches. In a fat-tree network with n-port switch, there are n pods. Each pod includes $\frac{n}{2}$ aggregation switches and $\frac{n}{2}$ edge/ToR switches. In each pod, each n-port edge switch is connected to $\frac{n}{2}$ servers and $\frac{n}{2}$ aggregation switches. Each pod consists of a ToR/edge switch and some hosts. Fig. 3 illustrates a four-port fat-tree topology. A Clos topology is a multi-rooted tree and usually consists of three levels of switches: ToR/edge switches directly connected to servers, aggregation switches connected to ToR switches, and intermediate/core switches, each of which is connected to each aggregation switches. Fig. 4 shows an example of three level Clos topology. Evaluating their effectiveness is beyond the scope of this paper.

2.3 Related Work

This section only discusses the existing algorithms which considered mapping not only virtual nodes but also virtual links.

VNE algorithms. Previous work indicated that VNE problem is NP-hard. Various cost-aware and/or energy-aware VNE approaches have been proposed. See [21], [22], [23] and references therein. All the existing VNE algorithms assumed that nodes were homogeneous. Moreover, the abilities of these algorithms in the large-scale CDCs are challenged.

Cloud service placement algorithms. Significant efforts have been devoted to mitigate the network effects on the QoS guarantee for cloud applications in CDCs. Various rate-limiter based approaches were also proposed to achieve throughput and/or latency guarantees, see [9] and references therein. Recently, data center network virtualization is proposed for benefiting those network-intensive cloud applications. Feamster et al. [29] addressed VDCE problem in distributed infrastructures. They proposed a VDC partitioning algorithm with the objective of minimizing the bandwidth cost and proposed a heuristic sub-VDC embedding algorithm with the objective to reduce power and carbon

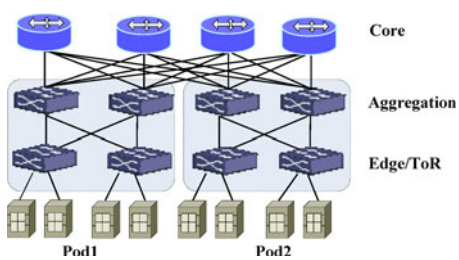


Fig. 4. Clos topology data center topology.

footprint costs. However, no detail on the relationship between energy consumption and the sub-VDC mapping was given in [29]. Note that our algorithms can be directly applied to distributed infrastructures.

Larumbe and Sansò [30] considered the scenario where there was one tenant request to be embedded. They first proposed a CPLEX-based embedding algorithm by formulating the problem as a MIP model and then proposed a heuristic algorithm so as to reduce the execution time. Note that this heuristic embedding algorithm was not energy-aware. But our heuristic algorithms are energy-aware. Konstanteli et al. [31] came up with a probabilistic model for optimizing allocation of resources, including hosts, their subnets, storage and computational capacity. Both affinity and anti affinity rules were used in [32]. Fuerst et al. [28] also discussed the benefit of collocation (affinity) and proposed a preclustering mechanism. NSS-JointSL without Equation (15) can be applied to the collocation situation. NSS-GBFS can be applied directly to the collocation situation.

Both Larumbe [30] and Konstanteli [31] aimed to minimize various costs, such as physical resource consumption and energy consumption. But they assumed that the relative significance of optimization objectives was regulated by the CSPs. Our JointSL algorithm can achieve an automatic regulation of optimization objectives according to CDC resource state. Such automatic regulation is more preferable for dynamic CDCs.

Zhani et al. [32] proposed a migration-assisting VDC embedding algorithm. Zhang et al. [33] considered the existence of various failures and proposed an availability-aware VDC embedding algorithm. Gu et al. [34] considered regional multifailure events and proposed failure-aware VNE algorithms.

All the algorithms in [29], [30], [31], [32], [33], [34] did not consider node heterogeneity, which exists in cloud data centers. Some algorithms have been proposed to map virtual VMs and virtual switches, respectively. Rabbani et al. [26] proposed a VDC embedding algorithm (denoted Rab in the rest of the paper) in order to minimize the embedding cost by using an exhaustive searching way. Different from Rab, our algorithms can minimize the energy consumption in the light-load scenarios while minimizing the embedding cost in the heavy-load scenarios. How to establish an environmental friendly cloud data center is being a hot topic. Various techniques have been proposed, like using the green energy, exploiting cheap energy, reducing energy consumption, and designing energy-aware virtual machine placement mechanisms. Rabbani also considered energy in the VDC embedding problem in [27]. But the proposed algorithm [27], denoted as Hi-Vi in the following, assumed that there is only one virtual switch in each VDC request. Our algorithms can deal with a VDC request with any number of switches.

3 The Proposed VDC Embedding Approaches

This section first presents NSS-JointSL and NSS-GBFS in Section 3.1 and Section 3.2, respectively. Then we discuss the algorithm complexity in Section 3.3.

3.1 The NSS-JointSL

Algorithm 1 describes the NSS-JointSL algorithm, including two steps: NSS algorithm (Step 1–Step 20 of Algorithm 1) responsible for mapping virtual servers and JointSL algorithm (described in Algorithm 3) responsible for mapping virtual switches and links.

3.1.1 NSS Algorithm for Mapping Virtual Servers

NSS first maps the virtual server v^{VH} with largest rank, denoted by $NR^{virtual}_{server}$. In the rack with the largest NR_{ToR}, the physical server with the largest NR_{server} is chosen to host v^{VH}. This paper uses the resource amount requested by a virtual server as its $NR^{virtual}_{server}$ and uses the amount of available resources of a physical server as its NR_{server}. Discussing the methods of computing $NR^{virtual}_{server}$ and NR_{server} is beyond the scope of this paper. Rank NR_{ToR} of each ToR switch, defined in Equation (8), is used to determine the priority of a ToR switch to be selected and physical servers connected to it will be put into the candidate list

$$NR_{ToR}(n^{PS}) = \frac{MA_N(n^{PS})}{maxMA_N} \frac{1}{1+\mathrm{var}(NumAS)} + NumAS(n^{PS}) \quad (8)$$

Here, $MA_N(n^{PS})$ denotes the maximum $A_N(u^{PH})$ among all the active physical servers in the rack of n^{PS}. $maxMA_N$ denotes the maximum $MA_N(n^{PS})$ among the ToR switches in *ToRQ*. $NumAS(n^{PS})$ and $\mathrm{var}(NumAS)$ are defined in Table 1.

Algorithm 1. The **NSS-JointSL** Algorithm

Input: (G_V, G_P)
Output: The embedding solution.

1: *VnodeQ*←virtual servers in non-increasing order of their $NR^{virtual}_{server}$;
2: $v^{VH} = VnodeQ$.pop();
3: *fToRQ* ←ToR switches in non-increasing order of NR_{ToR};
4: **if**($i = 0; i < fToRQ.size; i^{++}$)
5: $s^{PS} = fToRQ$.pop(); *fCanPool*: = NULL;
6: *fCanPool* ← active physical servers connected to s^{PS} and satisfies the requirement of v^{VH} in non-increasing order of NR_{server}; repeat these for all inactive physical servers and put them to the tail of *fCanPool*;
7: while (*fCanPool*! = NULL) && (*MaxT*! = 0)
8: $w^{PH} = fCanPool$.pop(); $w^{PH}.touched =$ **TRUE**; *MaxT* −;
9: Apply NSS-Left algorithm with input $(VnodeQ, G_P, w^{PH})$;
10: **if** (NSS-Left returns **FALSE**) **then**
11: $w^{PH}.touched =$ **FALSE**;
12: **endif**
13: **endwhile**
14: **if** *MaxT* = = 0 **then** return **FALSE**;
15: **output** "there is no embedding solution";
16: **endif**;
17: **endif**;
18: **if** *fCanPool* is empty **then** return **FALSE**;
19: **output** "there is no embedding solution";
20: **endif**;
21: Apply JointSL algorithm with input (G_V, G_P);
22: if (JointSL returns **TRUE**) then
23: **output** the embedding solution; return;
24: Endif

Some explanations of Equation (8) are given as follows. The value of $(\frac{MA_N(n^{PS})}{maxMA_N} \cdot \frac{1}{1+\mathrm{var}(NumAS)})$ is not larger than 1. Thus, $NumAS(n^{PS})$ determines the value of NR_{ToR} if there

exists difference among$NumAS$. That is, a ToR switch candidate with the maximum $NumAS$ is chosen first, facilitating the reduction in both the number of physical links to be used and energy consumption.

Algorithm 2. NSS-Left Algorithm

Input: $(VnodeQ, G_P, w^{PH})$
Output: Mapping results of left virtual nodes in *VnodeQ*
1: **Construct *ToRQ and UnusedToRQ***. Put the ToR switch connected to w^{PH} into *ToRQ*; put in *UnusedToRQ* the left ToR switches in non-increasing order of NR_{ToR};
2: **Construct *CanPool***. Put in *CanPool* all physical servers u^{PH} which is connected to one of the ToR switches in *ToRQ*;
3: **for** $(j = 1; j < VnodeQ.size; j^{++})$
4: $u^{VH} = VnodeQ$.pop();
5: *flag*: = FALSE;
6: **Construct *LocalCanPool***. Put in *LocalCanPool* all active physical servers in *CanPool* in non-increasing order of NR_{ToR}; Put all inactive physical servers in *CanPool* to the tail of *LocalCanPool* in non-increasing order of NR_{ToR};
7: **while (TRUE)**
8: ***Filter LocalCanPool***. Remove all elements in *LocalCanPool* which do not satisfies the requirement of u^{VH};
9: **if** (*LocalCanPool* is not empty) **then**
10: $u^{PH} = LocalCanPool$.pop();$u^{PH}.touched$: = **TURE**;
11: remove u^{PH} from *CanPool*; *flag*: = TRUE; **break**;
12: **endif**
13: **if** *UnusedToRQ* = = NULL **then** return FALSE; **endif**;
14: **if** there exists a ToR switch s^{PS} in *UnusedToRQ*, which is a sibling to one of ToR switches in *ToRQ* **then**
15: put all such ToR switches into *tempToRQ* and set *ToRQ* = *ToRQ*∪ *tempToRQ*;
16: **else**
17: pick up a ToR switch s^{PS} with the maximum available physical resources and move from *UnusedToRQ* into *ToRQ*;
18: **endif**
19: **Construct *LocalCanPool***. Put in *LocalCanPool* all active physical servers connected to one of the ToR switches in *tempToRQ* in non-increasing order of NR_{ToR};
20: Repeat step 21 for all inactive physical servers connected to one of the ToR switches in *tempToRQ* and add them to the tail of *LocalCanPool*;
21: *CanPool:* = *CanPool* ∪ *LocalCanPool*;
22: **Endwhile**
23: **Endfor**
24: return **TRUE**

Among the ToR switches with the same $NumAS$, their NR_{ToR} are distinguished by $\frac{MA_N(n^{PS})}{maxMA_N}$. That is, the ToR switch connected to the physical server with largest $MA_N(n^{PS})$ is to be selected first. Now explain the function of $\frac{1}{1+\text{var}(NumAS)}$. If all ToR switches have same $NumAS$, $\frac{1}{1+\text{var}(NumAS)}$ has no effect on NR_{ToR}. When there exists difference among $NumAS$ of some ToR switches, it is possible that a switch with small $NumAS$ has large $MA_N(n^{PS})$ but a switch with large $NumAS$ has small $MA_N(n^{PS})$. In such situation, we use $\frac{1}{1+\text{var}(NumAS)}$ to reduce the effect of $(\frac{MA_N(n^{PS})}{maxMA_N} \cdot \frac{1}{1+\text{var}(NumAS)})$ on the value of NR_{ToR}. That is, a switch with large $NumAS$ but with small $MA_N(n^{PS})$ is chosen.

After mapping v^{VH}, NSS invokes NSS-Left algorithm (described in Algorithm 2) to map the left virtual servers. If NSS-Left cannot map the left virtual servers successfully, NSS uses Step 7-13 of Algorithm 1 to find another available physical server to host v^{VH} and NSS-Left is invoked again. This process is repeated until VDC is embedded successfully or the maximum time (*MaxT*) is reached. Note that if physical server u^P is chosen to host u^V in NSS and NSS-Left, u^P must meet the following two conditions:

i) $A_N(u^{VH}) \leq A_N(u^{PH})$,
ii) $A_E(e(u^{VH}, v^{VS})) \leq A_E(e(u^{PH}, w^{PS}))$.

Now we explain NSS-Left algorithm. This paper calls as sibling two ToR/edge switches with the same switch parent. NSS-Left algorithm reduces the mapping time in large-scale physical scenarios by trying to map all virtual servers to the physical servers, which are close to each other as possible. Such mapping strategy can also help reduce the link mapping cost and reduce energy consumption. NSS-Left first tries to map the left virtual nodes to the physical servers which are in the same rack with w^{PH}. If there are no available physical servers, the sibling ToR switches are added to *ToRQ* and the physical servers connected to these newly-added ToR switch candidates are added to *CanPool*. The next virtual server will find a physical server in *CanPool* to host it. If there is no available physical server found in CanPool, more ToR switches will be moved from *UnusedToRQ* to *ToRQ* and NSS-Left tries to find available physical servers from the racks of these newly selected ToR switches.

3.1.2 JointSL Algorithm for Mapping Virtual Switches and Links

Algorithm 3 describes JointSL algorithm. The rationality behind Step 1 is that when mapping virtual switches and virtual links, it is not necessary to consider all the physical servers which do not host a virtual server and the physical links which are adjacent to one of these physical servers. We illustrate $G_{\overline{P}}$ in Step 1 by using Figs. 3 and 5a. We assume that the virtual servers of VDC1 in Fig. 5a are mapped to S1 and S12 of the physical network in Fig. 3, respectively. Then we get $G_{\overline{P}}$ described in Fig. 6, in which the bold links and the nodes connected by them make up $G_{\overline{P}}$. We set $G_{\bar{\bar{P}}} = G_P - G_{\overline{P}}$. Thus $N^{\bar{\bar{P}}H} = N^{PH} - N^{\bar{P}H}$, and $N^{\bar{\bar{P}}S} = N^{PS} - N^{\bar{P}S}$. **Proposition 1** in **Appendix B, available in the online supplemental material,** guarantees that mapping each virtual link $e^V \in G_V$ in $G_{\overline{P}}$ is same as mapping in G_P.

Algorithm 3. The **JointSL** algorithm

Input: (G_V, G_P)
Output: The mapping results of virtual switches and edges
1: Remove from G_S all the physical servers which do not host a virtual server and the physical links which is adjacent to one of these physical servers. The left topology is denoted as $G_{\overline{P}}$;
2: Use CPLEX solver to solve MIP-JointSL with input $(G_V, G_{\overline{P}})$;
3: if **CPLEX solver** return TRUE then return **TRUE**;
4: else return **FALSE**

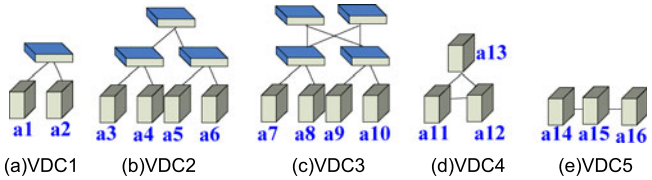


Fig. 5. Examples of VDC network topologies.

With the above definitions, we get the MIP formulation, MIP-JointSL as follows. Equation (9) defines the objective function, where

$$f(G_V)=\omega\cdot\left(\sum_{v\in N^P}P_1^v y_v+\sum_{e^{\bar{P}}\in E^{\bar{P}}}P_2^{e^{\bar{P}}}x_{vw}\right)+RC(G_V). \quad (9)$$

Objective:

$$\text{Minimize } f(G_V). \quad (10)$$

Subject to:
Capacity constraints:

$$\sum_{\forall u^V\in N^V} z_{u^V}^{u^P} c(u^V)\le A_N(u^P)y_{u^P},\forall u^P\in N^{\bar{P}} \quad (11)$$

$$\sum_{\forall e^{\overline{P}}(u^P,v^P)\in E^{\overline{P}}} g_{e^V(u^V,v^V)}^{e^{\bar{P}}(u^P,v^P)} b(e^V)\le A_E(e^{\bar{P}})x_{u^Pv^P},\forall e^V(u^V,v^V)\in E^V. \quad (12)$$

Flow related constraints:

$$\sum_{\forall e^{\overline{P}}(w^P,v^P)\in E^{\overline{P}}} g_{e^V(u^V,v^V)}^{e^{\overline{P}}(w^P,v^P)}-\sum_{\forall e^{\overline{P}}(u^P,w^P)\in E^{\overline{P}}} g_{e^V(u^V,v^V)}^{e^{\overline{P}}(u^P,w^P)}=z_{u^V}^{w^P}-z_{v^V}^{w^P},$$
$$\forall w^P\in N^{\overline{P}},\forall e^V(u^V,v^V)\in E^V. \quad (13)$$

Binary constraints:

$$\sum_{u^{PS}\in\Omega(u^{VS})} z_{u^{VS}}^{u^{PS}}=1,\forall u^{VS}\in N^{VS}, \quad (14)$$

$$\sum_{\forall u^{VS}\in N^{VS}} z_{u^{VS}}^{u^{PS}}\le 1,\forall u^{PS}\in N^{PS}, \quad (15)$$

$$z_{u^V}^{u^P}=0,\forall u^P\in N^{\overline{P}H},\forall u^V\in N^{VS}, \quad (16)$$

$$x_{u^Pv^P}=x_{v^Pu^P},\forall u^P,v^P\in N^{\bar{P}}, \quad (17)$$

$$x_{u^Pv^P}\le y_{u^P},\forall u^P,v^P\in N^{\bar{P}}, \quad (18)$$

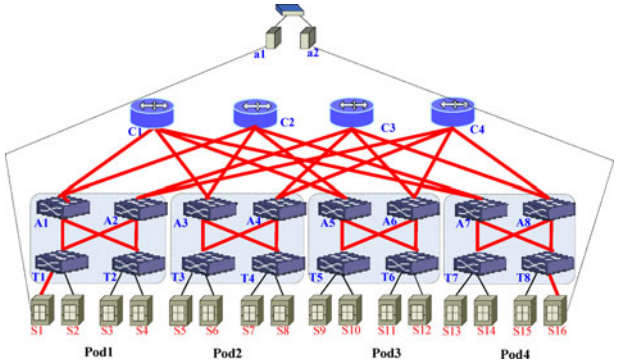


Fig. 6 Example of $G_{\overline{P}}$.

$$y_{v^P}\le\sum_{u^P\in N^{\bar{P}}} x_{u^Pv^P},\forall u^P,v^P\in N^{\bar{P}}, \quad (19)$$

$$\lambda_u\le y_u,\forall u\in N^{\bar{S}}, \quad (20)$$

$$\eta_{uv}\le x_{uv},\forall u,v\in N^{\bar{S}}, \quad (21)$$

$$y_u\ge z_{u^V}^u,\forall u\in N^{\bar{S}},\forall u^V\in N^V. \quad (22)$$

Domain constraints:

$$x_{u^Pv^P}\in\{0,1\},\forall u^P,v^P\in N^{\overline{P}}, \quad (23)$$

$$y_{u^P}\in\{0,1\},\forall u\in N^{\overline{P}}, \quad (24)$$

$$\eta_{u^Pv^P}\in\{0,1\},\forall u^P,v^P\in N^{\overline{P}}, \quad (25)$$

$$\lambda_{u^P}\in\{0,1\},\forall u^P\in N^{\overline{P}}, \quad (26)$$

$$g_{e^V(u^V,v^V)}^{e^{\overline{P}}(u^P,v^P)}\in\{0,1\},\forall e^{\overline{P}}\in E^{\overline{P}},\forall e^V\in E^V, \quad (27)$$

$$z_{u^V}^{u^P}\in\{0,1\},\forall u^P\in N^{\overline{P}},\forall u^V\in N^V. \quad (28)$$

Remarks:

1) Equation (10) is the objective function, which aims to minimize the energy consumption (defined in Equation (4)) first and then tries to minimize the embedding cost (defined in Equation (6)). We

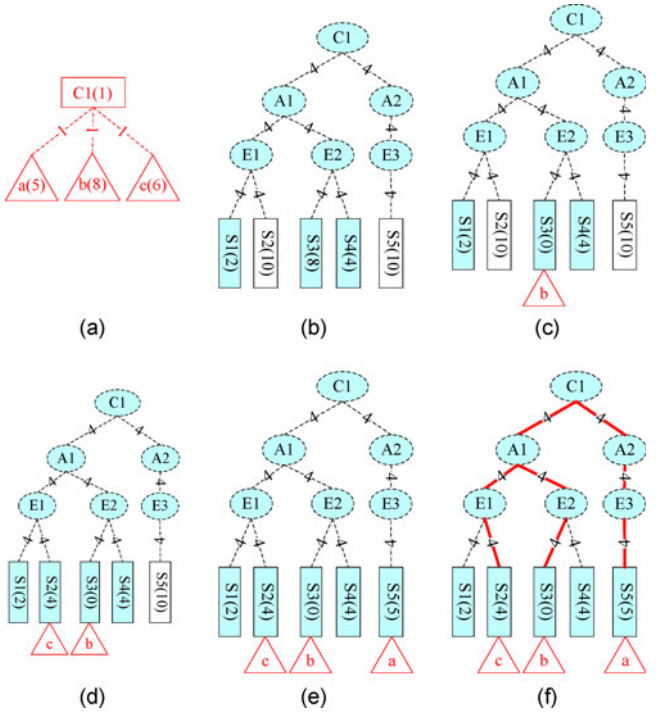


Fig. 7. An example of a VDC request's embedding using NSS-JointSL.

prove this property in **Proposition 3** of **Appendix B, available in the online supplemental material.**

2) Constraint sets (11-12) enforce the capacity bounds of physical switches.
3) Constraint set (13) refers to the flow conservation conditions and makes sure that the two virtual nodes connected by virtual link $e^V(u^V, v^V)$ are mapped to physical node u and v, respectively.
4) Constraint set (14) makes sure that only one physical switch is selected for each virtual switch, whereas constraint set (15) ensures that no more than one virtual switch for a VDC request is placed on a physical node.
5) Constraint set (17) states the features of an undirected graph.
6) Constraint sets (18-19) state that a physical switch is active if and only if there exists at least a physical link connecting this physical switch is active.
7) Constraint sets (20-21) state that if a physical switch/link is active before embedding G_V, it must be active after embedding G_V.
8) Constraint set (22) states that if a physical switch is hosting a virtual switch, it must be active.

3.1.3 An Example to Illustrate NSS-JointSL

Fig. 7 is used as an example to illustrate the process of embedding a VDC request (in Fig. 7a) on the CDC physical network (in Fig. 7b) under NSS-JointSL. A shaded node is an active node and the unshaded means inactive. The values in parentheses denote the CPU amount requested by the VDC or the available CPU of a physical server.

Firstly, sort all virtual servers and get: b, c, a. Then, sort ToR switches and put into *fToRQ*. We assume that the example is working in the light-load condition and then $NumAS(n^{PS})$ determines the value of NR_{ToR}. Thus, *fToRQ* includes E2, E1, E3. Since Server S3 is connected to E2 and has maximum available CPU, it is chosen to host virtual node b, shown in Fig. 7c. Since physical server S1 has no enough available resources for hosting virtual server c, E2's sibling ToR switch, E1, is added to *ToRQ*. Then physical server S2 is selected for hosting virtual node c, shown in Fig. 7d. Since all physical servers below A1 cannot host virtual server a, all ToR switches below C1 are added and sorted according to their NR_{ToR}. Thus, S5 is chosen to host c.

After all virtual servers are mapped successfully by using NSS, JointSL is applied to map virtual switches and links. In the example, $G_{\overline{P}}$ consists of all the bold links in Fig. 7f and the physical nodes connected by these links。

3.2 The NSS-GBFS Algorithm

NSS-GBFS algorithm maps virtual servers using NSS algorithm also, but applies a simple energy-aware randomly-mapping algorithm (randomly pick up one first among active physical switches and then among inactive physical switches) to map each virtual switch and uses GBFS to map each virtual link. GBFS is a variant of BFS algorithm. The only difference is that in each step of GBFS, the inactive physical switch is considered only after there is no available active physical switch.

3.3 Time Complexity Analysis

Runtime complexity is an important concern when designing algorithms for real-time applications. Both the methods of calculating ranks of servers and switches can be solved in polynomial time [21]. Thus, NSS is a polynomial-time algorithm. Since JointSL and BFS are both polynomial-time, the algorithms we proposed can be solved in polynomial-time.

4 Performance Evaluation

It is known that CDC network virtualization is not widely deployed. The actual characteristics of both substrate networks and VDC requests are still not completely understood. Thus, we use simulations to study the performance of our embedding algorithms as in [26] and [27]. We use the network topologies and configurations, which have been used to study other cloud computing problems, to do simulations. In the following, we first present the VDC embedding algorithms to be compared. Then we present the topologies for simulations. At last we present the simulation results and discussions.

4.1 Details of the Algorithms Compared and Performance Metrics

To the best of our investigated literature, only Rab [26] and Hi-Vi [27] considered node heterogeneity. Thus, we only compare the performance of NSS-JointSL, NSS-GBFS, Rab and Hi-Vi. Hi-Vi aims for the survivable VDC provision. In our simulations, we only consider its ability in VDC embedding. Since it can only handle VDC requests with one switch, its performance is only evaluated under star-type VDC requests. Rab algorithm does not describe how to construct the candidate physical server pool. We use randomly selecting policy. All these algorithms are implemented in our simulator. Without loss of generality, only CPU

constraints are considered for servers and only memory constraints are considered for physical/virtual switches as in [35]. *MaxT* in our algorithm is set to 1 in all simulations.

The metrics considered in heavy-load scenarios include

i) *Acceptance Ratio*, which measures the percentage of total VDC requests accepted by an algorithm over a given period.
ii) *Normalized Average Revenue*. *Average Revenue* measures the generated revenue (defined in Equation (1)) of an embedding algorithm over time (50 time units). We use 10,000 to normalize it in order to describe the result conveniently.
iii) *CPU Utilization and Network Utilization*, which measures the utilization of physical recourses.

The metrics considered in light-load scenarios include

i) Average Active Node Number and Average Active Link Number. They measure the number of active physical nodes (including servers and switches) and the number of active physical links of an interval T (set to 1,000 time units), respectively.

4.2 Simulation Environment Configurations

Four physical network scales are considered: small-scale, medium-scale, large-scale and ultra-large-scale. The substrate network topologies and virtual network topologies are generated by using the GT-ITM tool [39].

4.2.1 Small-Scale Scenarios

Physical network topology. VL2 topology (Clos Topology with no oversubscription) is used for small-scale physical topology. The physical topology is a three-level VL2-tree topology, containing 160 physical servers in four racks connected by 12 physical switches. Thus, there are 192 physical links. Each physical server is set to have 64 cores [40] and each switch has 64 G memory. Bandwidth between a ToR switch and an aggregation switch is set to 10 Gbps. Bandwidth between aggregation switch and core switch is set to 10 Gbps. Bandwidth between ToR switch and physical server is set to 1 Gbps.

Virtual network topology. Two kinds of VDC topologies are considered: star topology and VL2 topology. The star topology is suitable for hosting several types of applications like web applications and MapReduce [8].

In each topology, the numbers of virtual servers in the VDC requests is uniformly distributed from 2 to 10. The bandwidth of the link between a virtual switch and a virtual server is uniformly distributed from 1 to 100 Mbps. The CPU requirement of each virtual server is uniformly distributed from one to eight cores, and memory requirement of each virtual switch is from 1 to 2 G.

4.2.2 Medium-Scale Scenarios

Physical network topology. We used the VL2 topology described in [26] for medium-scale simulations, which provides full bisection bandwidth in the data center network. The physical topology is a three-level VL2 topology, containing 400 servers in 4 racks connected by 12 switches. Thus, there are 432 physical links. The configurations of physical servers, physical switches and link bandwidth are set as in small-scale simulations.

Virtual network topology. The configuration of each VDC request is also same as that in small-scale scenarios.

4.2.3 Large-Scale Scenarios

Physical network topology. We use the large fat-tree topology [35] for large-scale simulations. The physical topology is a three-level fat-tree topology with degree 16, containing 1,024 servers in 128 racks connected by 320 switches. Thus, there are 3,072 physical links. The configurations of physical servers, physical switches and link bandwidth are set as in small-scale simulations.

Virtual network topology. We also consider star and VL2 topologies. In each topology, the numbers of VMs in a VDC request is uniformly distributed from 2 to 20. The bandwidth of the link between the switch and the server is uniformly distributed from 1 to 100 Mbps. The CPU requirement of each virtual server is uniformly distributed from one to four cores, and memory requirement of each virtual switch is from 1 to 2 G.

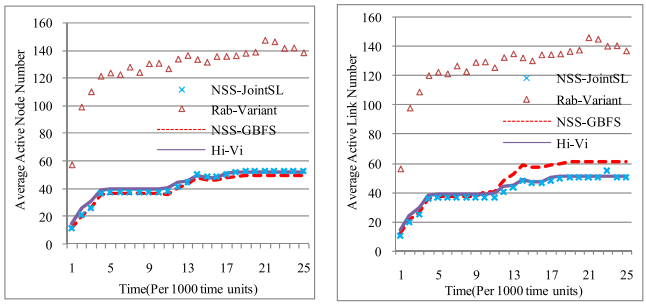
4.2.4 Ultra-Large-Scale Scenarios

Physical network topology. The settings of the ultra-large-scale datacenter are as follows. There are 16,000 physical servers, each with four VDC slots. 40 physical servers constitute a rack and are linked with a ToR switch with 1 Gbps links. Every 20 ToR switches are connected to an aggregation switch, and 20 aggregation switches are connected to the core switch of the datacenter. The default oversubscription of the physical network is 4, i.e., ToR switches are connected to aggregation switches with 10 Gbps links, and aggregation switches to the core switch with 50 Gbps links. Such settings have been used for performance evaluation in [8]. Processing resources of each physical server and RAM of each physical switch are set as in the small-scale scenarios.

Virtual network topology. Two lists of VDC requests are generated. The VDC network topology in the first list is virtual cluster. The second is virtual oversubscribed cluster requests. The number of VMs in a VDC request in each list is exponentially distributed around a mean of 49, as in [8]. As in [8], the tenant VMs are arranged in $\sqrt{|N^V|}$ groups, each containing $\sqrt{|N^V|}$ VMs. The default oversubscription of the VDC network is 4. We begin with a physical network with 10:1. The bandwidth of the link between the switch and the server is uniformly distributed from 1 to 100 Mbps. The CPU requirement of each virtual server is uniformly distributed from one to eight cores, and memory requirement of each virtual switch is from 1 to 2 G.

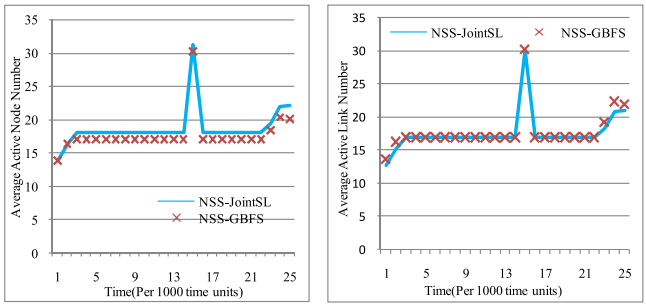
4.3 Simulation Result Analysis

In all simulations, VDC requests arrive in a Poisson process with an average rate of 0.01 requests per second. The lifetimes of the VDC requests follow an exponential distribution with an average of 3 hours. Each simulation is run for 1,200 VDC requests. All the weights defined in Section 2 are set to 1 in all simulations. The simulations are done in CentOS 5.4 running in VMware workstation virtual machine. The PC configuration is Intel Core i7-3770 (quad-core 3.40 GHz) CPU and 8 GB memory.



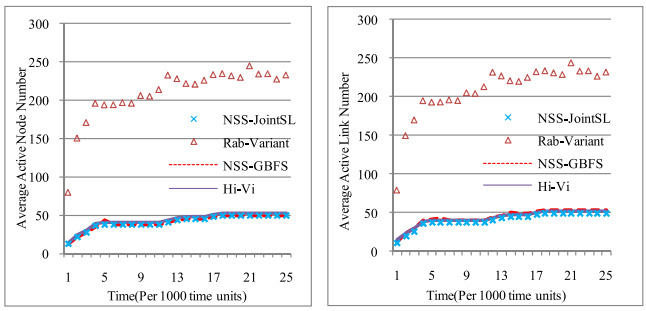
(a) Active node number (b) Active link number

Fig. 8. Star-type VDC request results in small-scale scenarios.



(a) Active node number (b) Active link number

Fig. 9. VL2-type VDC request results in small-scale scenarios.

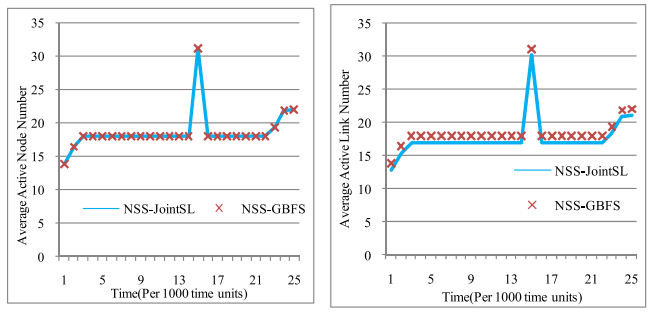


(a) Active node number (b) Active link number

Fig. 10. Star-type VDC request results in medium-scale scenarios.

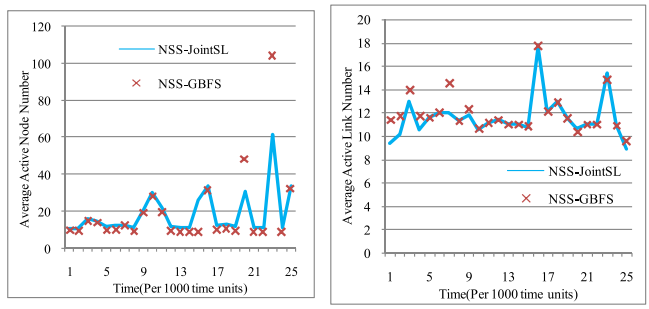
The light-load simulations aim to compare the energy consumption under each algorithm. Thus, the VDC *Acceptance Ratio* of each algorithm is all 100 percent. However, it is hard for Rab-Variant to achieve 100 percent *Acceptance Ratio*. In order to evaluate the capability of its virtual server mapping algorithm, we define Rab-Variant algorithm. It applies the virtual server mapping algorithm of Rab-Variant but applies the virtual switch and link mapping method of Hi-Vi. Figs. 8, 9, 10, 11, 12 illustrate the results. Rab-Variant can only handle a VDC request with one switch also. Our experiments indicate that when the scale of the physical infrastructure is increasing, the computing times of both Rab-Variant and Hi-Vi are increased significantly and then both of them cannot complete an experiment in a reasonable time in large/ultra-large scale scenarios. Thus, there are no results of their experiments in Figs. 9, 11 and 12.

The heavy-load simulations aim to compare the VDC acceptance ratio, the CSP long-term revenue, and the embedding cost under each algorithm. Since the performances of each evaluated algorithm under VL2-type VDC requests and under star-type VDC requests are similar, we only give the results of star-type VDC requests in small and



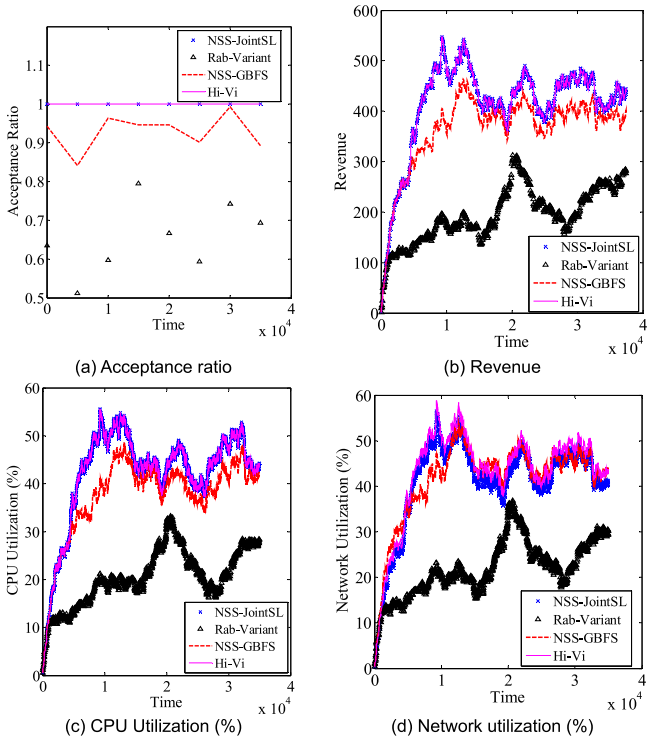
(a) Active node number (b) Active link number

Fig. 11. VL2-type VDC request results in medium-scale scenarios.



(a) Active node number (b) Active link number

Fig. 12. Star-type VDC request results in large-scale scenarios.



(a) Acceptance ratio (b) Revenue

(c) CPU Utilization (%) (d) Network utilization (%)

Fig. 13. Star-type VDC request results in small-scale scenarios.

medium-scale simulations. Figs. 13, 14, 15, 16, 17 describe the results. Rab-Variant and Hi-Vi are not evaluated in the large-scale and ultra-large-scale simulations due to computation complexity.

Note that in all simulations, both Switch-NSS-GBFS and Server-NSS-GBFS perform same in terms of the metrics, mentioned in Section 4.1. Thus, Figs. 8, 9, 10, 11, 12, 13, 14, 15, 16, 17 give the results of only one algorithm of them, denoted

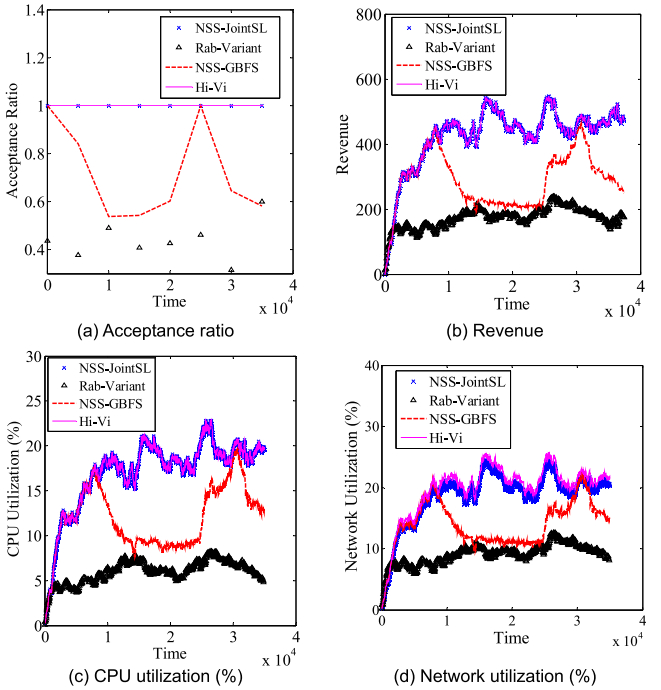


Fig. 14. Star-type VDC request results in medium-scale scenarios.

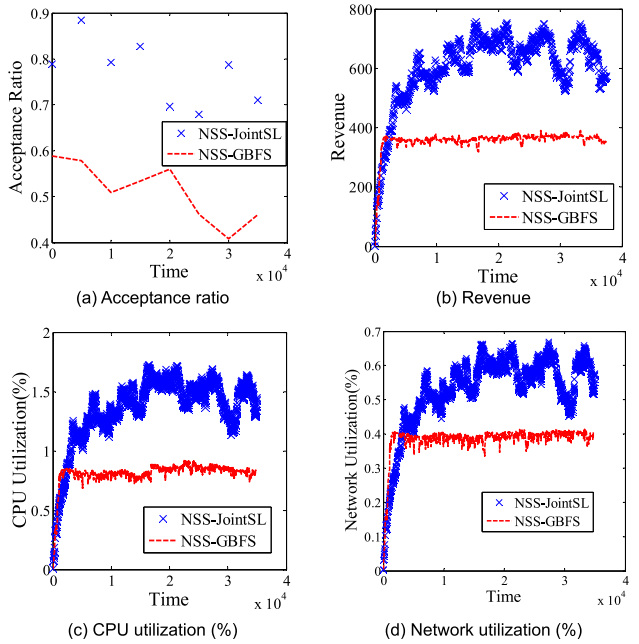


Fig. 16. Voc-type VDC request results in ultra-large-scale scenarios.

as NSS-GBFS. But their computation time of embedding a VDC is different. See Fig. 18, in which kVDC represents that the VDC has k virtual servers. The average time for embedding a VDC successfully is used as the metric.

4.3.1 Result Analysis in Light-Load Scenarios

1) *Rab-Variant performs worst in all scenarios in terms of both active node number and active link number.*

The results in Figs. 8 and 10 indicate that both active node number and active link number under Rab-Variant are larger than those under the other three algorithms. Especially after the 5,000 th time units, more than half of the total nodes/links are active. But less than one third of the total nodes/links are active under the other three algorithms.

It is because that Rab-Variant uses a simple random way to map virtual servers. The random selection results in more active nodes and then more active links. What's more, Rab-Variant is inefficient because it uses an exhaustive way for finding out feasible mapping solutions. Therefore, it is hard to be applied in large-scale and ultra-large-scale scenarios.

2) *The average active node number and average active link number of Hi-Vi are close to those of NSS-JointSL and*

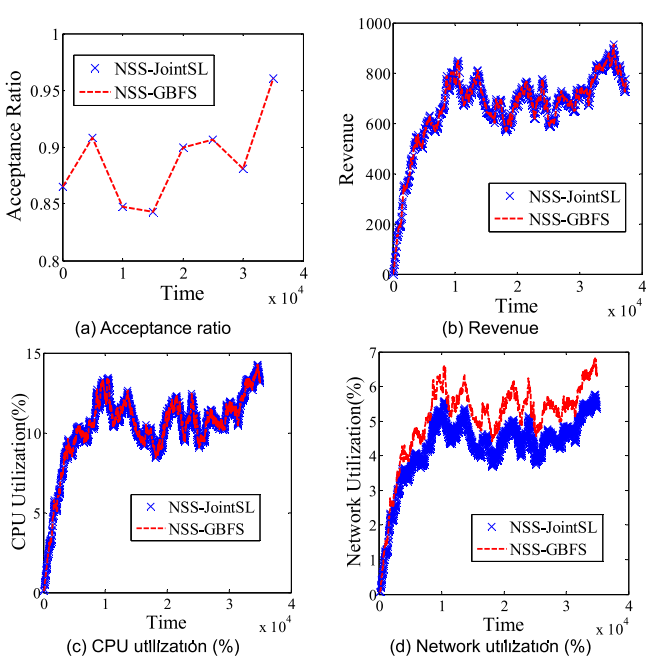


Fig. 15. Star-type VDC request results in large-scale scenarios.

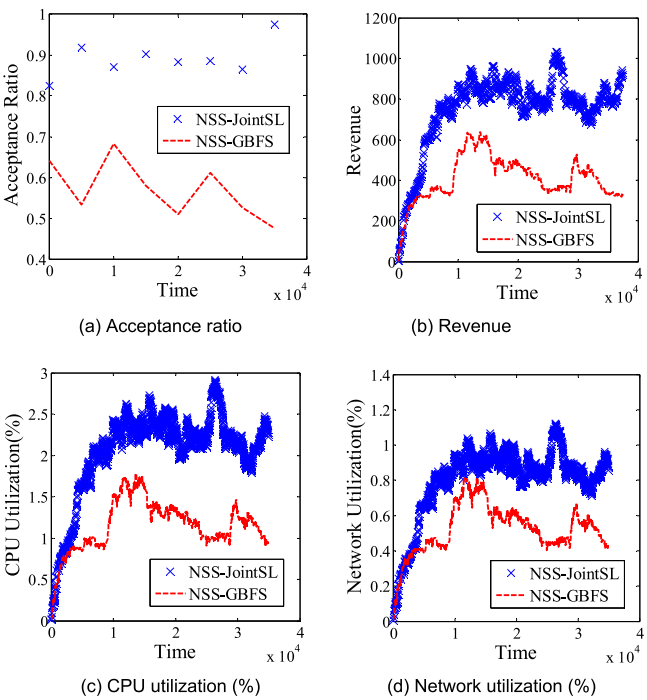


Fig. 17. Vc-type VDC request results in ultra-large-scale scenarios.

NSS-GBFS, but lower than that of Rab-Variant.

The results in Figs. 8 and 10 indicate that Hi-Vi performs well. The reason is that Hi-Vi maps virtual servers by considering active physical nodes first and then inactive physical nodes. In addition, it maps virtual switches and links by traversing all the alternative solutions in order to find out the best solution. Such exhaustive way can help it obtain the best embedding solution in terms of less energy consumption in light-load scenarios. However, it is hard for Hi-Vi to work in large-scale scenarios.

3) *NSS-JointSL performs best in terms of energy consumption in most light-load simulations.*

Figs. 8, 9, 10, 11, 12 indicate that NSS-JointSL performs best in all light-load simulations except in small-scale scenarios. The reason is that NSS-JointSL considering active physical nodes (links) first and then inactive physical nodes (links). In addition, NSS-JointSL tries to map virtual servers on the physical servers, which are as close as possible. These strategies help the reductions in the number of active physical nodes and physical links.

Note that in Fig. 8, NSS-GBFS is best in terms of average active node number. But the average active link number under NSS-GBFS is larger than that under NSS-JointSL in the last 15,000 time units, shown in Fig. 8b. It is because that NSS-JointSL maps virtual switches and links in an optimized way. This may cause some physical links to be bottlenecks and then more physical servers are turned on.

4.3.2 Result Analysis in Heavy-Load Scenarios

1) *Rab-Variant performs worst in all simulations in terms of both revenue and efficiency.*

The results in Figs. 13b and 14b indicate that *Acceptance Ratio* and *Average Revenue* of Rab-Variant are significantly lower than that of the other algorithms.

The reason is that during mapping virtual servers, physical nodes are added to the candidate list one by one randomly, without considering the factors which may affect acceptance ratio. This may cause more physical links to be used to host the virtual links and then less available resources are left for future VDC requests. Thus, Rab-Variant produces lower acceptance ratio and then lower long-term revenue.

2) *Hi-Vi performs well in small- and medium-scale scenarios in terms of both revenue and resource utilization.*

The results in Figs. 13b and 14b indicate that *Acceptance Ratio* and *Average Revenue* of Hi-Vi are similar to that of NSS-JointSL and are better than that of NSS-GBFS. It is due to its exhaustive searching way. But it does not work in large-scale scenarios.

3) *NSS-JointSL performs best in all scenarios in terms of long-term revenue.*

Figs. 13, 14 show that *Acceptance Ratio* and *Average Revenue* of NSS-JointSL and Hi-Vi are almost similar, better than that of NSS-GBFS and Rab-Variant. But NSS-JointSL's *Network Utilization* is better than Hi-Vi, shown in Figs. 13d and 14d. This is because NSS-JointSL maps virtual switches and virtual links in a joint way, leading to its perfect performance. The lower *Acceptance Ratio* and *Average Revenue* of NSS-GBFS are due to the method of selecting physical switches randomly. Then bandwidth under NSS-GBFS becomes the bottleneck, resulting in the high rejection of VDC requests.

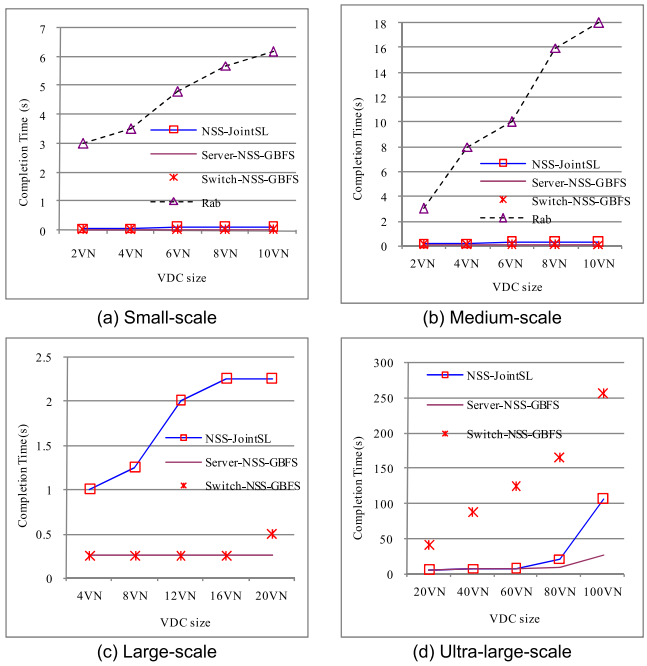


Fig. 18. Solving time of embedding a VDC.

4.3.3 Completion Time of Embedding a VDC Request

The results in Fig. 18 demonstrate the time efficiency of NSS-GBFS and NSS-JointSL in all scales, compared to Rab-Variant. As is shown in Figs. 18a, 18b, in small-scale and medium-scale topology, when embedding a VDC request, solving time under Rab-Variant is longer than under the other algorithms dramatically. It is because it uses an exhaustive way for finding out feasible mapping solutions, wasting lots of time. But the candidate pool under NSS algorithm is increasing step by step. The solving time under NSS-JointSL is larger than under NSS-GBFS. It is due to JoinSL's time complexity.

We also observe that Server-NSS-GBFS is fastest because of the special CDC topology. In CDCs, there is only one switch connecting with each server but each switch may connect many servers. Therefore, when mapping a virtual link, if path searching starts from a physical server, the searching space is much smaller than starting from a physical switch.

5 Conclusions and Future Work

This paper proposes two effective and efficient VDCE approaches, NSS-JointSL and NSS-GBFS, to address VDCE problem with the goal of minimizing energy consumption in light-workload CDCs while minimizing embedding cost in heavy-workload CDCs. The proposed virtual server mapping algorithm reduces energy consumption and

embedding cost by selecting physical servers as close as possible. The proposed joint mapping approach achieves the embedding goals by mapping virtual switches and links simultaneously. Simulations results validate NSS-JointSL's ability in achieve the objectives. Simulations results also demonstrate that Server-NSS-GBFS is an effective and efficient VDCE algorithm.

Although the proposed VDCE approaches outperformed previous works, the computation time of embedding a VDC request is still a challenge, especially when more kinds of physical resources are considered. One future work is to design parallel algorithms for the proposed approaches in order to reduce the computation time. Another future work direction is to explore the integration of Software-defined Networking (SDN) and network virtualization techniques to control the large-scale cloud data centers. In addition, physical nodes and links are prone to failures. We are thinking about the extension of the proposed algorithms to allow mapping backup virtual nodes, switches and links at VDC admission time so as to provide tenants with survivable VDC.

Acknowledgments

The work described in this paper has been supported by National Natural Science Foundation of China (No. 61572066) and NCET-11-0565. Xiaolin Chang is a corresponding author of this paper.

References

[1] Hadoop [Online]. Available: http://lucene.apache.org/hadoop, 2012.
[2] A. M. Matsunaga, M. O. Tsugawa, and J. A. B. Fortes, "CloudBLAST: Combining mapreduce and virtualization on distributed resources for bioinformatics applications," in *Proc. 4th Int. Conf. e-Sci., e-Sci.*, 2008, pp. 222–229.
[3] B. Sharma, T. Wood, and C. R. Das, "HybridMR: A hierarchical mapreduce scheduler for hybrid data centers," in *Proc. 33rd IEEE Int. Conf. Distrib. Comput. Syst.*, 2013, pp. 102–111.
[4] [Online]. Available: http://www.zdnet.com/hadoop-in-the-cloud-with-amazon-google-and-mapr-7000000069/, 2012.
[5] [Online]. Available: Amazon EC2 http://aws.amazon.com/ec2/, 2013.
[6] GCE [Online]. Available: https://cloud.google.com/compute/, 2013.
[7] [Online]. Available: http://www.microsoft.com/windowsazure/, 2013.
[8] H. Ballani, P. Costa, T. Karagiannis, and A. I. T. Rowstron, "Towards predictable datacenter networks," in *Proc. ACM SIGCOMM Conf.*, 2011, pp. 242–253.
[9] S. Angel, H. Ballani, T. Karagiannis, G. O'Shea, and E. Thereska, "End-to-end performance isolation through virtual datacenters," in *Proc. 11th USENIX Conf. Operating Syst. Des. Implementation*, 2014, pp. 233–248.
[10] [Online]. Available: https://www.altiscale.com/hadoop-blog/gigaom-hadoop-as-a-service/, 2014.
[11] K. Bilal, M. Manzano, S. U. Khan, E. Calle, K. Q. Li, and A. Y. Zomaya, "On the characterization of the structural robustness of data center networks," *IEEE T. Cloud Comput.*, vol. 1, no. 1, p. 1, Jan.–Jun. 2013.
[12] M. Bari, R. Boutaba, R. Esteves, L. Granvilley, M. Podlesny, M. Rabbani, Q. Zhang, and F. Zhani, "Data center network virtualization: A survey," *IEEE Commun. Surveys Tuts.*, vol. 15, no. 2, pp. 909–928, 2nd Quarter 2013.
[13] P. Rygielski and S. Kounev, "Network virtualization for QoS-aware resource management in cloud data centers," *A Survey. Praxis der Informationsverarbeitung und Kommunikation*, vol. 36, no. 1, pp. 55–64, 2013.
[14] N. Spring, R. Mahajan, D. Wetherall, and T. Anderson, "Measuring ISP topologies with rocketfuel," *IEEE/ACM Trans. Netw.*, vol. 12, no. 1, pp. 2–16, Feb. 2004.
[15] Q. Zhang, M. F. Zhani, Q. Zhu, S. Zhang, R. Boutaba, and J. Hellerstein, "Dynamic energy-aware capacity provisioning for cloud computing environments," in *Proc. IEEE/ACM 9th Int. Conf. Autonomic Comput.*, 2012, pp. 145–154.
[16] Amazon data center size [Online]. Available: http://goo.gl/ZRH2X, 2012.
[17] J. Komey, C. Belady, M. Patterson, A. Santos, and K. Lange, "Assessing trends over time in performance, costs and energy use for servers," *LLNL*, Intel Corporation, Microsft Corporation and Hewlett-Packard Corporation released on the web on August 17, 2009.
[18] Cisco Data Center Infrastructure 2.5 Design Guide [Online]. Available: https://www.cisco.com/application/pdf/en/us/guest/netsol/ns107/c649/ccmigration_09186a008073377d.pdf, 2007.
[19] A. Greenberg, J. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. Maltz, P. Patel, and S. Sengupta, "VL2: A scalable and flexible data center network," in *Proc. ACM SIGCOMM Conf. Data Commun.*, Aug. 2009, pp. 51–62.
[20] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *Proc. ACM SIGCOMM*, 2008, pp. 63–74.
[21] X.L. Chang, X.M. Mi, and J.K. Muppala, "Performance evaluation of artificial intelligence algorithms for virtual network embedding," *Int. J. Eng. Appl. Artif. Intell.*, vol. 26, pp. 2540–2550, 2013.
[22] E. Rodriguez, G. Alkmim, D. M. Batista, and N. L. Fonseca, "Green virtualized networks," in *Proc. IEEE Int. Conf. Commun.*, 2012, pp. 1970–1975.
[23] X. L. Chang, B. Wang, J. Q. Liu, W. B. Wang, and J. K. Muppala, "Green cloud virtual network provisioning based ant colony optimization," in *Proc. 15th Annu. Conf. Companion Genetic Evol. Comput.*, 2013, pp. 1553–1560.
[24] J. Chabarek, J. Sommers, P. Barford, C. Estan, D. Tsiang, and S. Wright, "Power awareness in network design and routing," in *Proc. IEEE INFOCOM*, 2008, pp. 457–465.
[25] V. Sivaraman, A. Vishwanath, Z. Zhao, and C. Russell, "Profiling per-packet and per-byte energy consumption in the NetFPGA gigabit router," in *Proc. IEEE Conf. Comput. Commun. Workshops*, 2011, pp. 331–336.
[26] M. G. Rabbani, R. P. Esteves, M. Podlesny, G. Simon, L. Z. Granville, and R. Boutaba, "On tackling virtual data center embedding problem," in *Proc. IEEE/IFIP Int. Symp. Integr. Netw. Manag.*, 2013, pp. 177–184.
[27] M. G. Rabbani, M. F. Zhani, and R. Boutaba, "On achieving high survivability in virtualized data centers," *IEICE Trans. Commun.*, vol. E97-B, no. 1, Jan. 2014.
[28] C. Fuerst, S. Schmid, and A. Feldmann, "Virtual network embedding with collocation: Benefits and limitations of pre-clustering," in *Proc. CLOUDNET*, 2013, pp. 91–98
[29] A. Amokrane, M. F. Zhani, R. Langar, R. Boutaba, and G. Pujolle, "Greenhead: Virtual data center embedding across distributed infrastructures," *IEEE Trans. Cloud Comput.*, vol. 1, no. 1, pp. 36–49, Jan.–Jun. 2013.
[30] F. Larumbe and B. Sansò, "A tabu search algorithm for the location of data centers and software components in green cloud computing networks," *IEEE Trans. Cloud Comput.*, vol. 1, no. 1, pp. 22–35, Jan.–Jun. 2013.
[31] K. Konstanteli, T. Cucinotta, K. Psychas, and T. A. Varvarigou, "Elastic admission control for federated cloud services," *IEEE Trans. Cloud Comput.*, vol. 2, no. 3, pp. 348–361, Jul. 2014.
[32] M. F. Zhani, Q. Zhang, G. Simon, and R. Boutaba, "VDC Planner: Dynamic migration-aware Virtual Data Center embedding for clouds," in *Proc. IEEE/IFIP Int. Symp. Integr. Netw. Manag.*, 2013, pp. 18–25
[33] Q. Zhang, M. F. Zhani, M. Jabri, and R. Boutaba, "Venice: Reliable virtual data center embedding in clouds," in *Proc. IEEE INFOCOM*, 2014, pp. 289–297.
[34] F. Gu, M. Rahnamay-Naeini, K. Shaban, S. U. Khan, N. Ghani, M. Hayat, and C. Assi, "Survivable cloud network mapping for disaster recovery support," *IEEE Trans. Comput.*, vol. 64, no. 8, pp. 2353–2366, Aug. 1, 2015.
[35] M. Moshref, M. L. Yu, A. Sharma, and R. Govindan, "Scalable rule management for data centers," in *Proc. 10th USENIX Symp. Netw. Syst. Des. Implementation*, 2013, pp. 157–170.

[36] ILOG CPLEX [Online]. Available: http://www.ilog.com/products/cplex/, 2011.
[37] N. Karmarkar, "A new polynomial-time algorithm for linear programming,"in *Proc. ACM 16th Annu. ACM Symp. Theory Comput.*, 1984, pp. 302–311.
[38] L. A. Wolsey and D. L. Nemhauser, *Nemhauser Integer and Combinatorial Optimization*. New York, NY, USA: Wiley, Jul. 1999.
[39] E. Zegura, K. Calvert, and S. Bhattacharjee, "How to model an Internetwork," in *Proc. IEEE INFOCOM*, 1996, pp. 594–602.
[40] S. Bell, B. Edwards, J. Amann, et al. "TILE64-processor: A 64-core SoC with mesh interconnect," in *Proc. IEEE Solid-State Circuits Conf. Dig. Techn. Papers*, 2008, pp. 88–598.
[41] S. Ghorbani, C. Schlesinger, M. Monaco, E. Keller, M. Caesar, J. Rexford, and D. Walker, "Transparent, live migration of a software-defined network," in *Proc. ACM Symp. Cloud Comput.*, 2014, pp. 1–14.
[42] K. V. Vishwanath, and N. Nagappan, "Characterizing cloud computing hardware reliability," in *Proc. 1st ACM Symp. Cloud Comput.*, Indianapolis, IN, USA, 2010, pp. 193–204.

Yang Yang received the BS degree from East China JiaoTong University, Jiangxi, China, in 2012. She is currently working toward the MS degree at Beijing Jiaotong University. Her research interests include cloud data center and machine learning.

Xiaolin Chang received the PhD degree in computer science from Hong Kong University of Science and Technology, in 2005. She is currently an associate professor at the School of Computer and Information Technology, Beijing Jiaotong University. Her research interests include cloud data center and network security. She is a member of the IEEE.

Jiqiang Liu received the BS and PhD degrees in 1994 and 1999, respectively, from Beijing Normal University. He is currently a professor at the School of Computer and Information Technology, Beijing Jiaotong University. He has published more than 60 scientific papers in various journals and international conferences. His research interests include trusted computing, cryptographic protocols, privacy preserving, and network security. He is a member of the IEEE.

Lin Li received the PhD degree in computer science from Shandong University, in 2007. She is currently an assistant professor at the School of Computer and Information Technology, Beijing Jiaotong University. Her current research interests include cryptography, cloud computing, and network security.