Zhang S, Qian ZZ, Wu J *et al.* Service-oriented resource allocation in clouds: pursuing flexibility and efficiency. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY Vol.x No.x: 1–x Month 2014

Service-Oriented Resource Allocation in Clouds: Pursuing Flexibility and Efficiency

Sheng Zhang[1,2] (张 胜), *Student Member, CCF, IEEE*, Zhu-Zhong Qian[1,2,*] (钱柱中), *Member, CCF, IEEE*, Jie Wu[3] (吴 杰), *Fellow, IEEE*, and Sang-Lu Lu[1,2] (陆桑璐), *Member, CCF, IEEE*

[1]*State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, 210023, China*
[2]*Department of Computer Science and Technology, Nanjing University, Nanjing, 210023, China*
[3]*Department of Computer and Information Sciences, Temple University, Philadelphia, PA 19122, U.S.A.*

E-mail: zhangsheng@dislab.nju.edu.cn; {qzz,sanglu}@nju.edu.cn; jiewu@temple.edu

Received December 24, 2013

Abstract The networking-oblivious resource reservation model in today's public clouds cannot guarantee the performance of tenants' applications. Virtual networks that capture both computing and networking resource requirements of tenants have been proposed as better interfaces between cloud providers and tenants. In this paper, we propose a novel virtual network model that could specify not only absolute and relative location requirements but also time-varying resource demands. Building on top of our model, we study how to efficiently and flexibly place multiple virtual networks in a cloud, and we propose two algorithms, *MIPA* and *SAPA*, which focus on optimizing resource utilization and providing flexible placement, respectively. The mixed integer programming-based MIPA transforms the placement problem into the multi-commodity flow problem through augmenting the physical network with shadow nodes and links. The simulated annealing-based SAPA achieves resource utilization efficiency through opportunistically sharing physical resources among multiple resource demands. Besides, SAPA allows cloud providers to control the trade-offs between performance guarantee and resource utilization, and between allocation optimality and running time, and allows tenants to control the trade-off between application performance and placement cost. Extensive simulation results confirm the efficiency of MIPA in resource utilization and the flexibility of SAPA in controlling trade-offs.

Keywords Resource allocation, virtual network embedding, opportunistic resource sharing

1 Introduction

The paradigm of cloud computing has experienced serious growth in recent years, attracting increasing attention from academic and industrial communities [1]. Today's public clouds (*e.g.*, Amazon EC2 and Microsoft Azure) adopt

Regular Paper
This work was supported in part by the National Natural Science Foundation of China under Grant Nos. 61202113 and 61321491, the Key Project of Jiangsu Research Program under Grant No. BE2013116, the Natural Science Foundation of Jiangsu under Grant No. BK2011510, EU FP7 IRSES MobileCloud Project under Grant No. 612212, and HUAWEI under Project No. YBIN2011056.
*Corresponding Author

a networking-oblivious resource reservation model, which only allows tenants to specify computing resource demands, and ignores networking completely. That is, almost all current clouds offer just best-effort networking service. The performance of tenants' applications are influenced by various factors, such as virtual machine (VM) placement and clouds workload. Considering the scarcity and the common oversubscription of cloud bandwidth resources, this reservation model makes a tenant's application performance very unpredictable [2]. The unpredictability further leads to two major consequences. First, tenants' expenses are increased, since cloud providers charge tenants based on the duration of an application, which depends on both computing and networking resources; second, cloud providers' revenues are decreased, since unpredictability impacts cloud applicability, and further limits the adoption of clouds [3, 4].

Recent works [3, 5, 6, 7, 8] noticed this lack of bandwidth guarantee in clouds, and proposed a *service-oriented* approach [9] that allows a tenant to specify his/her demand in a *virtual network* (VN), where nodes represent VMs with CPU demands, and edges represent bandwidth demands between VMs. Such resource demands are then enforced in clouds through source routing [5] or rate limiting [3, 6]. Albeit these works make a good start, there are still some limitations. For example, virtual networks only have star or tree topologies in [3, 6]; fixed resources are reserved throughout a virtual network's lifetime in [5, 7, 8].

In this paper, we introduce *RLVN*, a novel virtual network model that could specify not only absolute and relative location requirements, but also time-varying resource demands. Many virtual networks have time-varying resource demands, as evidenced in existing profiling experiments [6]. Thus, provisioning fixed resources for virtual networks throughout their lifetimes is clearly wasteful. Besides, many applications want to have location constraints on VMs. For example, VMs for content distribution services should be deployed in clouds as widely as possible to obtain a broad geographical footprint; VMs for parallel computing should be near each other to mitigate the impact of networking latency on task makespan; and the backup VMs should be placed in different failure regions to avoid geographically correlated region failures.

Building on top of our model, we study how to efficiently and flexibly place multiple virtual networks in a cloud, and we propose two algorithms, MIPA and SAPA, which focus on optimizing resource utilization and providing flexible placement, respectively. To maximize physical resource utilization, we design the mixed integer programming-based MIPA. The main idea of MIPA is transforming the placement problem into the multi-commodity flow problem [10], and constructing a mixed integer programming-based formulation through augmenting a physical network with shadow nodes and links.

To provide flexible and efficient virtual network placement, we design the simulated annealing-based SAPA. To efficiently utilize physical resources while retaining a performance guarantee, SAPA opportunistically shares physical resources between multiple virtual network demands. To flexibly control the trade-off between placement optimality and running time of the placement algorithm, we adopt simulated annealing [11] as our optimization framework. The motivation of providing such a trade-off is that, different virtual networks have different requirements on response time. For example, a virtual network request for supporting the real-time VoIP service should be deployed as quickly as possible, irrespective of whether the placement is optimal; on the contrary, for a virtual network request without requirements on response time, the cloud provider should focus on placement optimality. We show through extensive simulations that, MIPA and S-APA accepts up to 8.18% and 6.05% more virtual networks than two state-of-the-art allocation algorithms [7, 12], and the flexibility of SAPA in controlling several trade-offs is well confirmed.

The contributions of this paper are threefold. First, to the best of our knowledge, we are the first to study placing virtual networks with both location constraints and time-varying resource demands. We provide a generic VN model for tenants to flexibly trade off between application performance and placement cost. Second, through augmenting the physical network with shadow-nodes, we provide a Mixed Integer Programming-based algorithm, which gives us an upper bound on resource utilization. Third, we design a simulated annealing-based practical algorithm, which aims at providing flexible placements.

The remainder of this paper is organized as follows. The RLVN model is introduced in Section 2. The problem definition is described in Section 3. Local resource sharing in a single physical machine is given in Section 4. We present MIPA and SAPA in Sections 5 and 6, respectively. We conduct performance evaluations in Section 7. Before concluding the paper in Section 9, we go over related work in Section 8.

2 Virtual Network Model with Time-Varying Resource and Location Requirement (RLVN)

In this section, we first present the traditional virtual network and the RLVN models, then we present a simple model generation strategy for R-LVN, finally we show the advantages of the proposed model.

2.1 Traditional Virtual Network Model

We focus mainly on CPU and bandwidth resources in this paper, which has typically been the case in most of the prior studies [7,8,12,13]. Without loss of generality, the physical network is assumed to be based on time division multiplexing, where time is partitioned into multiple frames of equal length, and each frame is further divided into equal time slots. Both CPU and bandwidth resources are measured in time slots.

A traditional virtual network request is denoted by a weighted undirected graph $G^v = (V^v, E^v)$, where V^v is the set of virtual machines (VMs), and E^v is the set of virtual links (VLs). Each VM $n^v \in V^v$ is associated with a CPU demand $R_{\text{cpu}}(n^v)$ in time slots, and each VL $e^v_{uv} = (n^v_u, n^v_v) \in E^v$ is associated with a bandwidth demand $R_{\text{bw}}(e^v_{uv})$ in time slots. Each VN has a lifetime L, indicating how long the requested resources should be reserved in a cloud. Fig. 1(a) shows an example, where the corresponding resource demand of each VM or VL is written next to the respective node or link that represents it.

2.2 The RLVN Model

We find that most of the prior studies do not take dynamic resource demands of virtual networks or physical location constraints of virtual machines into account. On one hand, cloud tenants usually lease physical resources from cloud providers for installing their applications and services, which are accessed by end users. The randomness of end users and the dynamics of applications make the amount of physical resources actually utilized by virtual networks fluctuate over time, as shown in prior measurements [6, 14]. On the other hand, cloud tenants usually would like to restrict the physical locations of virtual machines for security, backup, coverage, or some other purposes. Combining them together motivates us to design RLVN that captures both time-varying resource and physical location requirements.

One can potentially derive some complicated functions, e.g., high-order polynomials, to capture the actual resource demands in a very precise way [6]. However, such smooth functions complicate the representation and provisioning of physical resources in clouds. To strike a balance between modeling precision and implementation difficulties, as well as to initiate a tractable study, we resort to a simple probabilistic model.

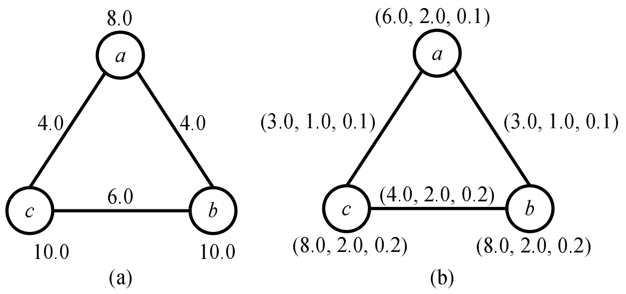

Fig. 1: In our RLVN model, the resource demand of a node n^v (resp. link e^v) is denoted by a tuple $(R^1_{\text{cpu}}(n^v), R^2_{\text{cpu}}(n^v), p(n^v))$ (resp. $(R^1_{\text{bw}}(e^v), R^2_{\text{bw}}(e^v), p(e^v)))$, and the preferable physical locations of a node n^v is denoted by a set $R_{\text{loc}}(n^v)$. (a) Traditional VN model. (b) The RLVN model.

We assume that the computing resource demand of a VM n^v is the probabilistic combination of two parts: a basic part $R^1_{\text{cpu}}(n^v)$, which exists throughout the lifetime of the virtual network, and a variable part $R^2_{\text{cpu}}(n^v)$, which exists with a probability of $p(n^v)$. That is, the variable part $R^2_{\text{cpu}}(n)$ follows a Bernoulli distribution. Here, $R_{\text{cpu}}(n^v) = R^1_{\text{cpu}}(n^v) + R^2_{\text{cpu}}(n^v)$. We denote the resource demands of a VM n^v and a VL e^v by tuples $(R^1_{\text{cpu}}(n^v), R^2_{\text{cpu}}(n^v), p(n^v))$ and $(R^1_{\text{bw}}(e^v), R^2_{\text{bw}}(e^v), p(e^v))$, respectively. Taking Fig. 1(b) for example, the demand of VM a is $(6.0, 2.0, 0.1)$, which means that, in a time slot, this VM requires 6 and 8 units of computing resources with probabilities of 0.9 and 0.1, respectively.

Since a cloud tenant may have requirements on the physical locations where his/her VMs are deployed, the RLVN model allows cloud tenants to specify two types of location constraints, *i.e.*, absolute and relative constraints. The absolute constraint of a VM n^v is denoted by a set $R_{\text{loc}}(n^v)$, which contains the physical machines that n^v should be placed on. For the relative location constraint, we assume a cloud physical network consists of multiple disjoint failure regions; that is, the failure regions form a partition of the physical network. For example, the physical network in Fig. 3 has three failure regions, as indicated by gray areas. If $R_{\text{loc}}(a) = \{A, B, C\}$, then VM a should be placed on one of the three physical machines; if $R_{\text{loc}}(b) \cap R_{\text{loc}}(c) = \emptyset$, then these two VMs should be placed in different failure regions.

2.3 Model Generation Strategy

This subsection presents a simple strategy which can be used by a cloud tenant to generate model parameters for each VM and VL in his/her virtual network request. It contains two steps.

First, a cloud tenant must get the computing and networking usage traces and guarantee them to be consistent with the realistic deployment in clouds. We envision that cloud providers offer profiling runs for tenants to obtain their resource usage traces. That is, a cloud tenant can tentatively deploy its VN request in a cloud for a relatively short time period; the cloud system collects the computing and networking usages over time, and feeds them back to the tenant.

Second, given the usage traces, a tenant needs to generate an appropriate tuple for each VM and VL. Here we provide here a strategy for a tenant to flexibly control the trade-off between application performance and cost through tuning R^1_{cpu}. It is better to illustrate the strategy using the example in Fig. 2, where the solid black curve shows the computing resource demand of a VM over time, while the dashed red curve represents our model. As mentioned in previous studies [6,15], the actual resource demands of virtual networks often exhibit cyclic patterns. In this example, we assume the cycle is T. Given R^1_{cpu} and the peak demand, *i.e.*, $(R^1_{\text{cpu}} + R^2_{\text{cpu}})$, we have $p = \frac{(t_4-t_3)+(t_2-t_1)}{T}$.

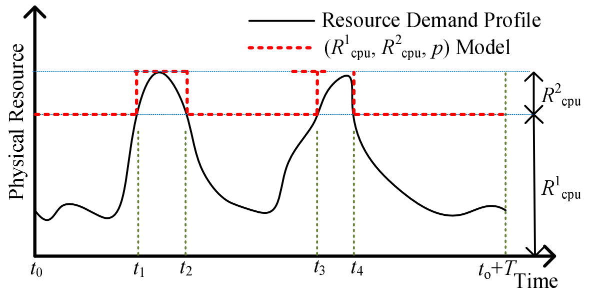

Fig. 2: An example of the generation strategy.

2.4 The Advantages of RLVN

RLVN has several desirable properties. First and foremost, previous VN models can be seen as special cases of our model. It ensures that our system is compatible with existing VN models, *e.g.*, virtual cluster [3], and virtual data center [5].

Second, we provide great flexibility for tenants to trade off between application performance and rent cost. At one extreme, if a tenant has a large amount of funds, and only cares about application performance, the tenant can request physical resources that are equal to the peak resource demand (*i.e.*, let the variable part be zero in RLVN), though some of the requested physical resources are not utilized most of the time. At the other extreme, if a tenant only cares about placement costs, the tenant can let the basic part be zero in RLVN. In general, a tenant can adjust model parameters to best suit his/her objective.

Finally, our model is also a trade-off between modeling complexity and precision. When the number of parts in our model (currently 2) increases, the model precision increases, and hence, can represent realistic resource demands more accurately. However, the complexity in generating parameters for RLVN, not surprisingly, increases as well, which may complicate the interactions between cloud providers and tenants.

One limitation of RLVN is that, modeling generation incurs some profiling overheads. However, this overhead can be drastically reduced if tenants have to reserve resources for the same type of VNs repeatedly and lastingly. For example, about 40% of applications are recurring in Bing's production data center [16]. For the same type of VNs, the cloud provider only needs to offer one profiling run, and the same results could be fed back to tenants who want to deploy that type of VN. Thus, the profiling overhead for cloud providers would be greatly reduced.

3 Problem Statement

The cloud physical network is modeled as a weighted undirected graph, $G = (V, E)$, where V denotes the set of physical machines (PMs), and E denotes the set of physical links (PLs). The amount of available CPU resources in a PM $n \in V$ is denoted by $A_{\text{cpu}}(n)$. A PL e_{ij} connects two PMs n_i and n_j, *i.e.*, $e_{ij} = (n_i, n_j)$. The amount of available bandwidth resources in a PL $e_{ij} \in E$ is denoted by $A_{\text{bw}}(e_{ij})$. PMs on the same rack are considered to be in the same failure region. In Fig. 3, the physical network consists of three disjoint failure regions. We use $P(n_i, n_j)$ to represent the set of loop-free physical paths between n_i and n_j. We also denote by P the set of all loop-free paths in the physical network. In Fig. 3, there are four loop-free paths between PMs B and J. The physical network is assumed to be fixed; for fault-tolerant resource allocation in clouds, please refer to [17,18,19].

Placing a RLVN request can be decomposed

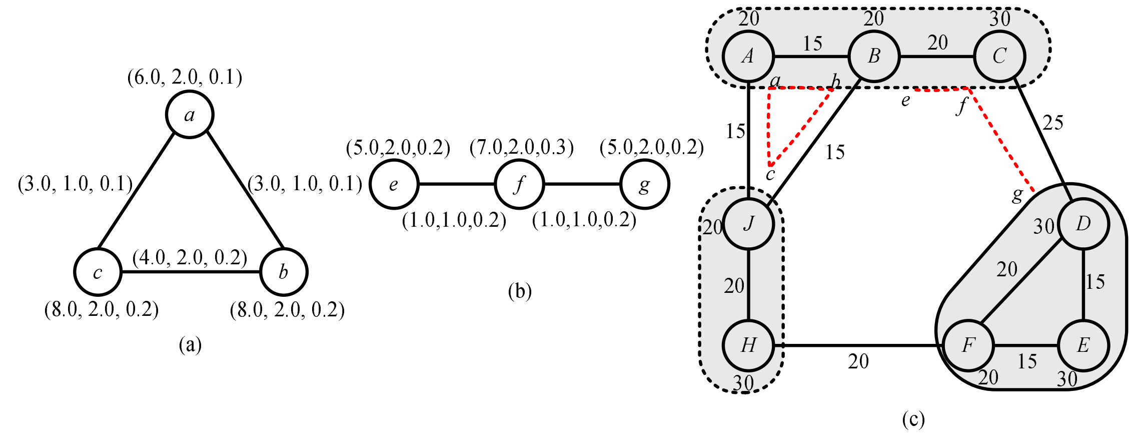

Fig. 3: An example of RLVN placement. The physical network consists of three failure regions (indicated by gray areas). The resource demand/capacity of each node or link is written next to the respective node or link that represents it. The dashed red lines give a possible placement of the two RLVNs. (a) RLVN G_1^v. (b) RLVN G_2^v. (c) Physical network G.

into two phases, namely, *VM mapping* and *VL mapping*. The VM mapping phase provides the placement results of VMs, and it can be seen as an mapping $\mathcal{M}_{\mathrm{V}}$ from V^v to V. We have, $\forall n_i^v, n_j^v \in V^v$:

$$\begin{cases} \mathcal{M}_{\mathrm{V}}(n_i^v) \in V, \\ A_{\mathrm{cpu}}(\mathcal{M}_{\mathrm{V}}(n_i^v)) \geq R_{\mathrm{cpu}}(n_i^v), \\ \mathcal{M}_{\mathrm{V}}(n_i^v) = \mathcal{M}_{\mathrm{V}}(n_j^v) \Leftrightarrow n_i^v = n_j^v. \end{cases}$$

The second condition ensures that a VM must be placed on a PM that has enough physical resources. The third condition guarantees that different VMs must be placed on different PMs, as in previous studies [7,8,12].

The VL mapping phase provides the placement results of VLs, and it can be seen as an mapping $\mathcal{M}_{\mathrm{E}}$ from E^v to P, where $\forall e_{uv}^v = (n_u^v, n_v^v) \in E^v$:

$$\begin{cases} \mathcal{M}_{\mathrm{E}}(e_{uv}^v) \subset P(\mathcal{M}_{\mathrm{V}}(n_u^v), \mathcal{M}_{\mathrm{V}}(n_v^v)), \\ \sum_{p \in \mathcal{M}_{\mathrm{E}}(e_{uv}^v)} A_{\mathrm{bw}}(p) \geq R_{\mathrm{bw}}(e_{uv}^v). \end{cases}$$

The available bandwidth of a path p is defined as the minimum of the bandwidths of all physical links along the path. That is, $A_{\mathrm{bw}}(p) = \min_{e \in p} A_{\mathrm{bw}}(e)$.

Taking Fig. 3 for example, the VM mapping for RLVN G_1^v is $\{a \rightarrow A, b \rightarrow B, c \rightarrow J\}$, and the VL mapping is $\{(a,b) \rightarrow \{(A,B)\}, (b,c) \rightarrow \{(B,J)\}, (c,a) \rightarrow \{(J,A)\}\}$. The VM mapping for RLVN G_2^v is $\{e \rightarrow B, f \rightarrow C, g \rightarrow D\}$, and the VL mapping is $\{(e,f) \rightarrow \{(B,C)\}, (f,g) \rightarrow \{(C,D)\}\}$.

Different VNs from different tenants usually offer different services, so it is reasonable to assume that the resource demands from different VNs are mutually independent. To provide efficient physical resource utilization, we propose the opportunistic sharing of physical resources among multiple variable resource parts from different VNs. However, when more than one variable part of resource demand occurs simultaneously, a capacity violation happens. To provide *probabilistic performance guarantee*, a cloud provider must *provide an upper bound on the collision probability*. We denote the upper bound by p_{th}. For example, in Fig. 3, VM b from G_1^v and VM e from G_2^v are placed on the same PM. If resource sharing is not exploited as in prior studies, these two VMs would occupy a total of 17 units of computing resources on PM B. However, when opportunistic resource sharing is allowed, these two VMs would occupy a total of only 15 units of resources on PM B. For example, let $p_{\rm th}$ be 0.1, since $p(b) \times p(e) = 0.2 \times 0.1 = 0.01 < p_{\rm th}$, we only have to allocate 2 units of resources to the variable resource parts of VMs b and e.

This paper focuses on placing multiple RLVNs that arrive and depart over time in a given physical network. Upon the arrival of a RLVN request, a decision must be made to determine whether or not to accept the request. Here, we assume that RLVN requests arrive one by one, and batch processing is not the focus of this paper. The goal is to maximize a cloud provider's revenue through efficiently utilizing physical resources, and in the meanwhile, to provide flexibility for both tenants and providers. Following prior studies [7, 8, 12], a cloud provider's revenue $\mathbb{R}(G_i^v)$ from embedding G_i^v is proportional to the amount of allocated resources and its lifetime L_i. That is,

$$\mathbb{R}(G_i^v) = [\alpha \sum_{n^v \in V^v} R_{\rm cpu}(n^v) + \beta \sum_{e^v \in E^v} R_{\rm bw}(e^v)] \times L_i,$$

where a cloud provider can adjust α and β to ensure that, neither CPU nor bandwidth becomes a bottleneck. In our simulation, both of α and β are set to 1. The total revenue of a cloud provider can be denoted by $\sum_{G_i^v} \mathbb{R}(G_i^v)$, where G_i^v is an accepted RLVN request.

We denote by *acceptance ratio* the ratio of the number of accepted RLVN requests to the number of all requests; denote by *node utilization ratio* the ratio of the amount of allocated computing resources to that of overall computing resources in a physical network; denote by *link utilization ratio* the ratio of the amount of allocated networking resources to that of overall networking resources in a physical network. For example, in Fig. 3, after successfully placing RLVNs G_1^v and G_2^v, the acceptance ratio is $2/2 = 1$, the node utilization ratio is $(8+10+10+7+9+7)/200 = 0.255$, and the link utilization ratio is $(4+4+6+2+2)/180 = 0.1$. It is easy to see that, maximizing a cloud provider's revenue is equal to maximizing the acceptance ratio, node utilization ratio, or link utilization ratio. We formally define the problem below.

Problem 1. *(RLVN Placement Problem) Given a physical network $G = (V, E)$ and a series of RLVN requests $G_i^v = (V_i^v, E_i^v)$, find a placement for these requests to maximize the acceptance ratio while providing flexibility for both tenants and providers.*

Determining whether a virtual network request could be placed in a given physical network is proven to be NP-hard [20], and thus, placing multiple RLVNs in a cloud network to maximize the acceptance ratio is also NP-hard. In this paper, we design two heuristic yet efficient algorithms, MIPA and SAPA, which focus on optimizing physical resource utilization and providing flexible allocation, respectively.

4 Local Resource Sharing

Before we present MIPA and SAPA in the next two sections, in this section, we first introduce how to share physical resources between multiple variable parts of resource demands from different VNs in *a single PM*. The technique developed in this section serves as a basic component for MIPA and SAPA.

Recall that both computing and networking resources are measured in time slots. In this section, we only present the technique for local resource sharing in a PM, and the result can be used in a PL without any major changes.

4.1 The Time Slot Assignment Problem

The number of time slots in a frame of each PM is proportional to the physical capacity of the respective PM. Consider the following time slot assignment problem: based on the placement generated by MIPA or SAPA, a set of m VMs are placed on a PM. The resource demand of the i-th VM n_i^v is $< R^1_{\text{cpu}}(n_i^v), R^2_{\text{cpu}}(n_i^v), p(n_i^v) >$. For the basic parts of resource demands, we have no choice but to allocate the respective amount of physical resources to them; for the variable parts of resource demands, we propose the opportunistic sharing of physical resources among them. Given an upper bound on the collision threshold p_{th}, the objective is to find a sharing solution that minimizes the amount of time slots used.

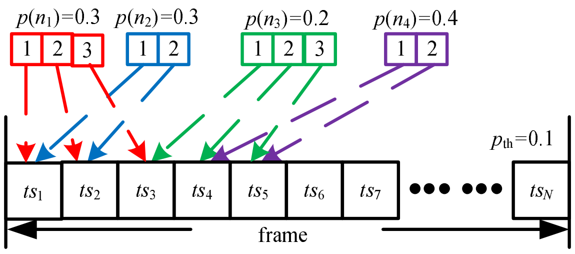

Fig. 4: An example of resource sharing among multiple variable parts of resource demands. The collision threshold serves as the "capacity" of a time slot.

We illustrate the idea of opportunistic resource sharing with the example in Fig. 4. Four VMs are placed on the same PM. VMs n_1, n_2, n_3 and n_4 require 3, 2, 3 and 2 time slots with probabilities of 0.3, 0.3, 0.2 and 0.4, respectively. The objective is to minimize the time slots occupied by the four VMs. Fig. 4 also shows a possible sharing solution, which occupies a total of 5 time slots. In this solution, ts_1 is shared between n_1 and n_2, because they collide with a probability of $0.3 \times 0.3 = 0.09$, which is less than p_{th}; ts_1 cannot be shared among n_1, n_2, and n_3, because their collision probability is 0.174, which is larger than p_{th}. More formally, the collision probability of a set S of variable parts of resource demands is defined as

$$P(S) = 1 - \prod_{n_i \in S}(1 - p(n_i)) - \sum_{n_i \in S}(p(n_i) \prod_{n_k \in S, k \neq i}(1 - p(n_k))).$$

4.2 The Benefit of Resource Sharing

This subsection provides some insights into the benefit of local resource sharing. In general, local resource sharing produces a win-win situation—cloud tenants' costs are lowered, and cloud providers' revenues are increased.

Consider a cloud provider CP_1 that has a physical link with a bandwidth capacity equal to 20 time slots, and there are three cloud tenants, CT_1, CT_2 and CT_3. Each of them wants to lease 8 time slots in the substrate link. Without resource sharing, it is clear that CP_1 can only accept two requests $(8 \times 3 = 24 > 20)$. If CP_1 charges one dollar for one slot per hour, then CP_1 can get 16 dollars per hour, and each tenant pays 8 dollars per hour to CP_1.

However, each tenant may find that his/her resource demand is composed of a basic part of 6 time slots and a variable part of 2 time slots, which occurs with a probability of 0.3. With resource sharing, CP_1 could accept all of the three requests in the following way. CP_1 assigns 18 dedicated slots to the basic parts, and lets the variable parts share the remaining 2 slots. The collision probability in each of the two sharing slots is

$$1 - 0.8 \times 0.8 \times 0.8 - 3 \times 0.8 \times 0.8 \times 0.2 = 0.104.$$

Since there are collisions for the variable parts of resource demands, $CP1$ may charge 0.1 dollar for one sharing slot per hour. Thus, the cost of an accepted request is $(6+0.1+0.1) = 6.2$ dollars per hour, which is smaller than the previous amount, and CP_1 gets $(6+0.1+0.1) \times 3 = 18.6$ dollars per hour, which is larger than the previous one.

We see that local resource sharing enables better utilization of physical resources, and hence, increases the revenues of cloud providers, and decreases the costs of cloud tenants. We believe that local resource sharing can benefit all parties through reasonable pricing. We will not discuss how to set prices in this paper, as it is out of this paper's scope and deserves separate study.

4.3 First-fit-based Sharing Algorithm

The time slot assignment problem has been proven to be NP-hard in the strong sense [12]. We observed that it is similar to the classic bin packing problem [21], which is to find a packing in unit-sized bins for a set of items with sizes that are less than one, so that the number of bins used is minimized. First-fit is an approximation algorithm with a factor of two for the bin packing problem.

```
First-Fit-Based Sharing (FFS)
 1: input: $R^2_{\text{cpu}}(n_i)$ and $p(n_i)$ for $i = 1$ to $m$
 2: for $i = 1$ to $m$
 3:     $count \leftarrow 0$, $j \leftarrow 1$
 4:     while $count < R^2_{\text{cpu}}(n_i)$ do
 5:         if $ts_j$ can accommodate $n_i$ then
 6:             place $n_i$ in $ts_j$
 7:             $count \leftarrow count + 1$
 8:         $j \leftarrow j + 1$
 9:     end while
10: end for
```

Fig. 5: Pseudocode for the first-fit-based sharing algorithm. It handles the micro-level time slot assignment, serving as a basic component for MIPA and SAPA.

We design *FFS*, a first-fit-based sharing algorithm, as shown in Fig. 5. For the i-th variable part of resource demand characterized by $R^2_{\text{cpu}}(n_i)$ and $p(n_i)$, FFS attempts to place it in the first time slot that can accommodate it; if this is not possible, FFS moves to the next time slot. FFS keeps on finding another time slot in which the i-th variable part of resource demand can be placed, until the number of these slots is equal to $R^2_{\text{cpu}}(n_i)$. By "accommodate" we mean that the collision probability is not larger than the threshold p_{th}. FFS can be executed in an on-line fashion, and has a low time complexity.

The arrows in Fig. 4 show the results after applying FFS to the example. FFS firstly tries to place the variable parts of resource demands from n_1, and places them in the first three time slots; when FFS checks whether ts_1 could accommodate n_2, since $p(n_1) \times p(n_2) < p_{\text{th}}$, FFS places the variable parts of resource demands of n_2 in the first two time slots, and so on.

We summarize this section by providing two final notes. First, in Section 3, we assume that different VMs from the same VN should be placed on different PMs. This assumption is made for brevity. The proposed algorithms can naturally adapt to the scenario when a tenant wants to deploy multiple VMs (*e.g.*, n_1, n_2, ..., n_k) on one PM. In this case, we can treat these k VMs as one large VM n_1^k. Since resource demands from VMs in the same VN are usually correlated, we cannot simply sum up the variable parts of resource demands. The resource demand of n_1^k consists of $(k + 1)$ parts: one basic part, *i.e.*, $\sum_{1 \le i \le k} R^1_{\text{cpu}}(n_i)$, and k variable parts, where the i-th variable part $R^2_{\text{cpu}}(n_i)$ occurs with a probability of $p(n_i)$. In FFS, we just have to ensure that the variable parts of resource demands from the same VM cannot share physical resources.

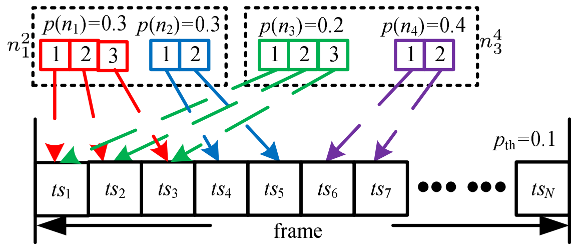

Fig. 6: Extending FFS to the scenario when a tenant wants to deploy multiple VMs on one PM.

Fig. 6 shows an example. Suppose that a tenant wants to treat n_1 and n_2 as one VM n_1^2, and another tenant wants to treat n_3 and n_4 as one VM n_3^4. When we use FFS to deal with time slot assignment, the variable parts of resource demands from n_1^2 cannot share physical resources. The arrows show the final assignment.

Second, while MIPA and SAPA generate the mapping from VMs (resp. VLs) to PMs (resp. loop-free physical paths), FFS determines the micro-level time slot assignment in a single PM or PL, and is a basic procedure in both MIPA and SAPA.

5 Mixed Integer Programming-based Algorithm (MIPA)

In this section, we first present the MIP-based algorithm for the case where all physical location requirements of VMs are in the form of a set of preferable PMs, then we show how to deal with the other cases.

5.1 Augmented Physical Network

The main idea of MIPA is to transform the placement problem into the multi-commodity flow problem. Given a RLVN request G^v, we can extend the physical network G to construct an augmented physical network G' by exploiting the absolute location requirements of G^v.

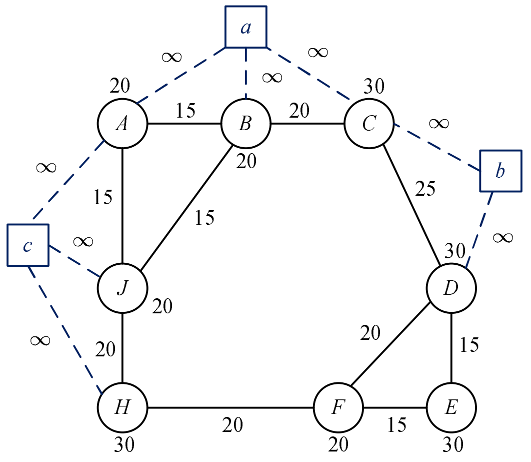

Fig. 7: The augmented physical network for placing RLVN G_1^v in physical network G (see Fig. 3 for details of G_1^v and G). The location requirements of G_1^v are $R_{\text{loc}}(a) = \{A, B, C\}$, $R_{\text{loc}}(b) = \{C, D\}$, and $R_{\text{loc}}(c) = \{A, J, H\}$.

The augmented physical network can be constructed as follows. For each $n^v \in V^v$, we create a corresponding *shadow-node* $\vartheta(n^v)$ (or n^v without causing confusions), and we connect n^v with all the PMs belonging to $R_{\text{loc}}(n^v)$ using *shadow-links* with infinite bandwidth capacities. Denote the augmented network of $G = (V, E)$ by $G' = (V', E')$, which is the combination of G, shadow-nodes, and shadow-links. We have

$$V' = V \cup \{n^v | n^v \in V^v\},$$
$$E' = E \cup \{(n^v, n) | n^v \in V^v, n \in R_{\text{loc}}(n^v)\}.$$

Fig. 7 shows an example, where the location requirements of G_1^v are $R_{\text{loc}}(a) = \{A, B, C\}$, $R_{\text{loc}}(b) = \{C, D\}$, and $R_{\text{loc}}(c) = \{A, J, H\}$. The shadow-links with infinite bandwidths connect a shadow-node to the physical nodes that belong to the respective set of preferable locations.

5.2 MIP-based Formulation

Based on the augmented network, we can take a virtual link $e_{ij}^v = (n_i^v, n_j^v) \in E^v$ as a commodity flow starting from n_i^v and ending at n_j^v. By forcing all flows starting from and ending at each shadow-node n^v to go through the same neighbor of n^v in G', we effectively oblige n^v to connect to only one active physical machine in $R_{\text{loc}}(n^v)$. In doing so, we have the result of VM mapping, that is, we can place n^v on that active PM. More formally, we present the MIP-based formulation below.

MIP-RLVN-Placement:

Unknown decision variables: $h(n^v, n)$, the indicative binary variable, which is 1 if n^v is mapped to n; otherwise, it is 0; $f(e_{ij}^v, e_{uv})$: the amount of flow from virtual link e_{ij}^v in the $u \to v$ direction of physical link e_{uv}.

Objective:

$$\min \sum_{n^v \in V^v} R_{\text{cpu}}(n^v) + \sum_{e_{ij}^v \in E^v} \sum_{e_{uv} \in V} f(e_{ij}^v, e_{uv}) \quad (1)$$

Capacity constraints:

$$h(n^v, n) \times R_{\rm cpu}(n^v) \le A_{\rm cpu}(n), \forall n^v \in V^v, \forall n \in V' \quad (2)$$

$$f(e^v_{ij}, e_{uv}) \ge 0, \forall e_{uv} \in E', \forall e^v_{ij} \in E^v \quad (3)$$

$$\sum_{e^v_{ij} \in E^v} (f(e^v_{xy}, e_{uv}) + f(e^v_{ij}, e_{vu})) \le A_{\rm bw}(e_{uv}), \forall e_{uv} \in E \quad (4)$$

Shadow-node constraints:

$$h(n^v, n) \in \{0, 1\}, \forall n^v \in V^v, n \in V' \quad (5)$$

$$\sum_{n^v \in V^v} h(n^v, n) \le 1, \forall n \in V' \quad (6)$$

$$\sum_{n \in R_{\rm loc}(n^v)} h(n^v, n) = 1, \forall n^v \in V^v \quad (7)$$

$$\sum_{n \notin R_{\rm loc}(n^v)} h(n^v, n) = 0, \forall n^v \in V^v \quad (8)$$

$$\sum_{e^v_{ij} \in E^v} (f(e^v_{ij}, e_{n^v w}) + f(e^v_{ij}, e_{w n^v})) \le h(n^v, n_w) \times A_{\rm bw}(e_{n^v w}), \forall n^v \in V^v, \forall n_w \in R_{\rm loc}(n^v) \quad (9)$$

Flow constraints:

$$\sum_{n_u \in V'} f(e^v_{ij}, e_{uv}) = \sum_{n_w \in V'} f(e^v_{ij}, e_{vw}), \forall n_v \in V' \setminus \{n^v_i, n^v_j\} \quad (10)$$

$$\sum_{n_w \in V'} f(e^v_{ij}, e_{n^v_i w}) = \sum_{n_u \in V'} f(e^v_{ij}, e_{u n^v_i}) + R_{\rm bw}(e^v_{ij}), \forall e^v_{ij} \in E^v \quad (11)$$

$$\sum_{n_w \in V'} f(e^v_{ij}, e^s_{n^v_j w}) = \sum_{n_u \in V'} f(e^v_{ij}, e_{u n^v_j}) - R_{\rm bw}(e^v_{ij}), \forall e^v_{ij} \in E^v \quad (12)$$

There are two kinds of unknown decision variables that correspond to VM and VL mappings, respectively. Recall that our goal is to maximize the cloud provider's revenue by utilizing physical resources efficiently; therefore, we use the objective function (Eq. (1)) to minimize the total physical resources that are allocated to the RLVN request.

Eqs. (2)-(4) provide the capacity constraints. If n^v is placed on n, then the available CPU resource of n must be no less than the resource demand of n^v (Eq. (2)). All flows must be positive (Eq. (3)), and the flows going through a physical link must not exceed the bandwidth capacity of that link (Eq. (4)).

Eq. (6) ensures that no more than one VM is placed on a PM. Eqs. (7) and (8) makes sure that the location requirements are respected. Remember that, for a shadow-node n^v, there is only one PM n_w in $R_{\rm loc}(n^v)$ that satisfies $h(n^v, n_w) = 1$, therefore, Eq. (9) forces all of the flows starting from n^v and ending at n^v to pass the same neighbor n_w. Eqs. (10)-(12) are flow-related constraints.

Solving an MIP is NP-hard; in our algorithm, we adopt LP relaxation and randomized rounding [22]. After we get the macro-level mapping solution, for each PM and PL, we invoke FFS to handle the micro-level time slot assignment.

5.3 How to Deal With the Other Cases

We have shown how to construct an augmented physical network for embedding a RLVN request that only has absolute location requirements. We now provide remarks on how to deal with the other cases, *i.e.*, relative and no location requirements.

If a VM n^v does not have any location requirements, we just assume $R_{\rm loc}(n^v)$ contains all PMs in the physical network, *i.e.*, $R_{\rm loc}(n^v) = \{n | n \in V\}$. If there are relative location requirements, the augmented network is constructed as follows. For each physical failure region, we create a corresponding *region-head-node*, and we connect a region-head-node with all the PMs belonging to the respective region using *region-links* with infinite bandwidths. We further connect a shadow-node with these region-head-nodes using shadow-links with infinite bandwidths.

Fig. 8 shows an example, where the location requirements of G_1^v are $R_{\text{loc}}(a) = \{A, B, C\}$, and $R_{\text{loc}}(b) \cap R_{\text{loc}}(c) = \emptyset$. By forcing all flows starting from and ending at b to pass the same region-head-node (*e.g.,* f_1) and the same PM (*e.g.,* C), we effectively select a PM (*e.g.,* C) for b. By forcing a region-head-node to be an active neighbor of only one shadow-node, we ensure that b and c are not placed in the same region.

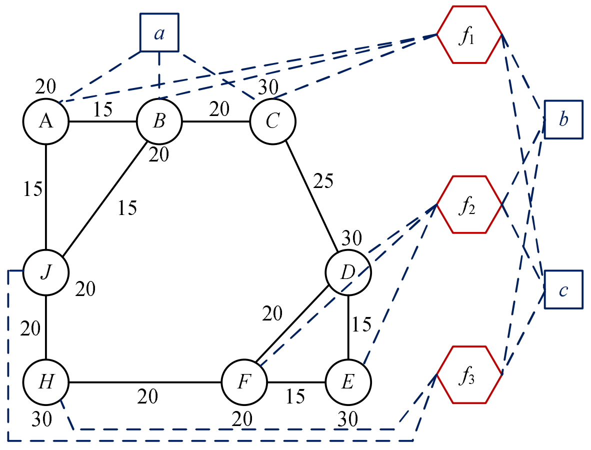

Fig. 8: The augmented physical network for placing G_1^v, where the location requirements are $R_{\text{loc}}(a) = \{A, B, C\}$, and $R_{\text{loc}}(b) \cap R_{\text{loc}}(c) = \emptyset$. The region-links with infinite bandwidths connect a region-head-node to the physical nodes belonging to the respective region.

Based on this augmented network, we can define a similar MIP-based formulation as before, and solve it by LP-relaxation and randomized rounding. The details are omitted due to space limitations, and are left to the reader.

6 Simulated Annealing-based Practical Algorithm (SAPA)

Based on previous expositions, this section presents a practical meta-heuristic-based algorithm. We first introduce the main idea of our algorithm, then we explains several key functions in the algorithm.

Determining whether a virtual network request could be placed in a given physical network is proven to be NP-hard [20]. We resort to simulated annealing to cope with its intractability. Simulated annealing was originally developed for very large scale integration design, and is now widely adopted in optimization problems [23, 24]. Simulated annealing is chosen for its simplicity; we believe that more complex methods would hinder the scalability, while gaining only incremental returns. While simulated annealing is a well-known technique, our contribution lies in the choice of appropriate energy and neighboring generation functions to ensure rapid convergence to a near-optimal allocation.

The main idea is to search through a solution state space to find a near-optimal solution by iteratively improving a candidate solution with regard to a given measure of *energy*. Fig. 9 shows the pseudocode of our algorithm. Given a physical network G, and a RLVN request G^v, SAPA returns a near-optimal solution $\mathcal{M}_B$. We first generate the initial solution and temperature; the function *CalculateEnergy* defines the energy of a solution. Variables $\mathcal{M}_B$ and ϵ_B are used to record the best solution so far, and its energy.

```
SAPA
 1: input: $G$, $G^v$, $N$, $\rho$
 2: $\mathcal{M} \leftarrow$ an initial solution for $G^v$
 3: $T \leftarrow$ an initial temperature
 4: $\epsilon \leftarrow CalculateEnergy(\mathcal{M})$
 5: $\mathcal{M}_B \leftarrow \mathcal{M}$, $\epsilon_B \leftarrow \epsilon$
 6: for $n = 1$ to $N$ do
 7:     $\mathcal{M}_\# \leftarrow GenerateNeighbor(\mathcal{M})$
 8:     $\epsilon_\# \leftarrow CalculateEnergy(\mathcal{M}_\#)$
 9:     if $\epsilon_\# < \epsilon_B$ then
10:         $\mathcal{M}_B \leftarrow \mathcal{M}_\#$, $\epsilon_B \leftarrow \epsilon_\#$
11:     if $Random() < p(\mathcal{M}, \mathcal{M}_\#, T)$ then
12:         $\mathcal{M} \leftarrow \mathcal{M}_\#$, $\epsilon \leftarrow \epsilon_\#$
13:     $T \leftarrow \rho T$
14: end for
15: return $\mathcal{M}_B$
```

Fig. 9: The pseudocode of SAPA.

In each iteration, we move to a slightly-different neighboring solution with a certain probability p, depending on the energies of the current solution, the neighboring solution, and the temperature. The temperature is decreased by a factor of ρ after each iteration. ρ is typically a value between 0.95 and 1 [11,23,24]. Allowing the solution to move to a higher energy solution helps us to avoid local minima.

To better understand the algorithm, the reader could imagine the solution state space as a graph, where the individual solutions are vertices. Two solutions have a edge between them if and only if we can get one of them from the other through *GenerateNeighbor*. Our algorithm is like a walk on this graph. In each iteration, we randomly choose a neighbor solution, and move to it with a probability p, which is determined by the energies of the current solution, the neighbor solution, and the temperature. We may move to a higher energy solution, which helps us avoid local minima. But the probability of moving to a higher energy solution is decreased as the temperature goes down. The larger the iteration count N is, the better the final solution is. Therefore, we can use N to control the trade-off between solution optimality and running time.

Note that, after we get the mapping solution from SAPA, we then invoke FFS to reduce the amount of occupied resources in each PM and PL, and thus improve the physical resource utilization.

6.1 Initial Solution and Temperature

We generate the initial solution using the following greedy approach. We first sort PMs and VMs in the order of decreasing available resources and resource demands, respectively. For each VM in this order, we place it on the first PM that has not been used before. This kind of "maximum-first" mapping fashion is beneficial for future requests that may require some bottleneck resources. We then map each virtual link to the shortest path with sufficient bandwidth between the corresponding PMs.

The moving probability $p(\mathcal{M}, \mathcal{M}_{\#}, T)$ for transition from the current solution $\mathcal{M}$ (with an energy of ϵ) to the neighboring solution $\mathcal{M}_{\#}$ (with an energy of $\epsilon_{\#}$) is defined as

$$p(\mathcal{M}, \mathcal{M}_{\#}, T) = \begin{cases} 1 & \text{if} \quad \epsilon_{\#} < \epsilon, \\ e^{-\frac{\epsilon_{\#}-\epsilon}{T}} & \text{o.w.} \end{cases}$$

The initial temperature T should be set to a value that makes the average moving probability to be 0.8 for "bad" transitions from the initial solution, according to the suggestions in [11]. By "bad" we mean that we generate a higher energy neighbor solution from the initial solution using the function *GenerateNeighbor*. We can get T in the following way: we randomly generate M energy-increasing neighbor solutions from the initial solution and compute the average increase in energy $\bar{\epsilon} = \frac{\sum(\epsilon_{\#}-\epsilon)}{M}$; let $p(\mathcal{M}, \mathcal{M}_{\#}, T) = e^{-\frac{\bar{\epsilon}}{T}} = 0.8$, we have $T = -\frac{\bar{\epsilon}}{\ln(0.8)}$.

6.2 Generating Neighboring Solutions

The function *GenerateNeighbor* finds a slightly-different solution in the neighborhood of the current candidate solution in each iteration. This function should be simple, otherwise it would cost more running time. A well-crafted neighbor generating function intrinsically avoids deep local minima. Fig. 10 shows the pseudocode of *GenerateNeighbor*. We first randomly select a VM $n^v \in V^v$, and denote by n_1 the PM on which n^v is placed in $\mathcal{M}$. We then use a breadth-first search starting from n_1 to find another PM n_2 that satisfies the capacity and location requirements, and place n^v on n_2 in $\mathcal{M}_{\#}$.

Denote by $\mathcal{M}(m^v, n^v)$ the physical path that (m^v, n^v) is placed on in $\mathcal{M}$. For each affected virtual link, *i.e.*, $(m^v, n^v) \in E^v$: if n_2 belongs to the physical path $\mathcal{M}(m^v, n^v)$, we just place (m^v, n^v) on a part of the path in $\mathcal{M}_{\#}$; otherwise, we extend $\mathcal{M}(m^v, n^v)$ in $\mathcal{M}_{\#}$ by adding the shortest path between n_1 and n_2. The randomness we employed in generating neighboring solutions helps us to walk uniformly within the solution state space and avoid local minima.

```
GenerateNeighbor
 1: input: $\mathcal{M}$
 2: $\mathcal{M}_{\#} \leftarrow \mathcal{M}$
 3: randomly select a VM $n^v \in V^v$
 4: $n_1 \leftarrow \mathcal{M}(n^v)$
 5: find another PM $n_2$ using breadth-first
    search starting from $n_1$
 6: $\mathcal{M}_{\#}(n^v) \leftarrow n_2$
 7: for each $(m^v, n^v) \in E^v$
 8:    if $n_2 \in \mathcal{M}(m^v, n^v)$ then
 9:        $\mathcal{M}_{\#}(m^v, n^v) \leftarrow$ the segment from
           $n_2$ to $\mathcal{M}(m^v)$ in $\mathcal{M}(m^v, n^v)$
10:    else
11:        find a path segment $ps$
           connecting $n_1$ and $n_2$
12:        $\mathcal{M}_{\#}(m^v, n^v) \leftarrow \mathcal{M}(m^v, n^v) + ps$
13:    end if
14: end for
15: return $\mathcal{M}_{\#}$
```

Fig. 10: The pseudocode of GenerateNeighbor.

6.3 Calculating Energy

The energy of a solution $\epsilon(\mathcal{M})$ must satisfy the following properties: (i) If a solution $\mathcal{M}$ occupies less physical resources than another solution $\mathcal{M}'$, then $\epsilon(\mathcal{M}) < \epsilon(\mathcal{M}')$; (ii) If a solution does not meet the resource demands of a RLVN request, then it has a higher energy than any other solution that meets the demands; (iii) For two solutions, both of which do not meet the resource demands of a RLVN request, the solution that incurs more unsatisfied resource demands must have a higher energy.

For a solution $\mathcal{M}$, the amount of unsatisfied resource demands can be defined as

$$\Delta(\mathcal{M}) = \sum_{n^v \in V^v} (R_{\text{cpu}}(n^v) - c(n^v, \mathcal{M}(n^v))) + \sum_{e^v_{ij} \in E^v} (R_{\text{bw}}(e^v_{ij}) - \sum_{\mathcal{M}(n^v_i) = n_u} f(e^v_{ij}, e_{uv})),$$

where $c(n^v, \mathcal{M}(n^v))$ and $f(e^v_{ij}, e_{uv})$ denote the amounts of resources that are allocated for n^v in $\mathcal{M}(n^v)$, and allocated for e^v_{ij} in e_{uv}, respectively. The energy of a solution is defined as

$$\epsilon(\mathcal{M}) = \begin{cases} \sum_{n^v \in V^v} R_{\text{cpu}}(n^v) + \sum_{e^v_{ij} \in E^v} \sum_{e_{uv} \in E} f(e^v_{ij}, e_{uv}) \\ \qquad \text{if } \mathcal{M} \text{ meets the demands of } G^v, \\ \epsilon_M + \Delta(\mathcal{M}) \qquad \text{o.w.} \end{cases}$$

Here, ϵ_M is a sufficiently large energy, *e.g.*, ϵ_M could be defined as the sum of all the physical resources in the physical network.

It is straightforward to see that the definition of energy satisfies the above three properties. We note that, the energy can be defined in different ways to achieve different purposes, *e.g.*, a definition that prefers load balancing is provided in [25].

6.4 Summary

Based on the RLVN model, opportunistic resource sharing, and simulated annealing, SAPA achieves high resource utilization through opportunistically sharing physical resources among multiple resource demands. The flexibility of S-APA is reflected in several aspects: allowing tenants to control the trade-off between application performance and placement cost; allowing cloud providers to control the trade-offs between performance guarantee and resource utilization, and between allocation optimality and running time; and so on.

7 Performance Evaluation

This section describes our evaluation results of the proposed algorithms. The simulations settings are similar to those in prior studies [7, 8]. There are two types of settings in our evaluation: *small-setting*, and *large-setting*. The small topology setting is designed for MIPA, due to its high computational complexity; the large topology setting is used in the evaluations without involving MIPA.

In the large-setting, the physical network is configured to have 50 physical machines that are randomly connected with a probability of 0.5. Both of the CPU resource capacity of every physical machine and the bandwidth resource capacity of every physical link are generated uniformly from a range of integers from 50 to 100. The threshold of collision probability is set to 0.1, and the annealing parameter ρ is set to 0.99 according to [11, 23, 24]. The number of failure regions in the physical network is set to 5, and we randomly partition physical machines into five nonempty sets.

For each RLVN request, the number of VMs is uniformly generated from a range of integers from 2 to 10 and each pair is connected with a probability of 0.5. We check whether a virtual network is connected; if not, we regenerate it until we get a connected one. The arrivals of RLVN requests are modeled as a Poisson process with an average rate of two requests per minute. The lifetime of each request is assumed to be exponentially distributed with an average of ten minutes. The CPU resource demand of each VM is uniformly generated from a range of integers from 1 to 20, and the bandwidth resource demand of each VL is uniformly generated from a range of integers from 1 to 50. The ratio of the variable part of resource demand to the overall resource demand of each VM or VL is uniformly generated from a range of real values from 0.1 to 0.2. For the location requirements, we randomly choose two or four VMs from each request and assume them to have relative location requirements; the rest of the VMs are assumed to have absolute location requirements.

In the small-setting, simulation parameters are the same as those in the large-setting, except that the physical network is configured to have ten machines, and the number of VMs in a RLVN request is uniformly generated from a range of integers from 2 to 5.

7.1 Efficiency in Resource Utilization

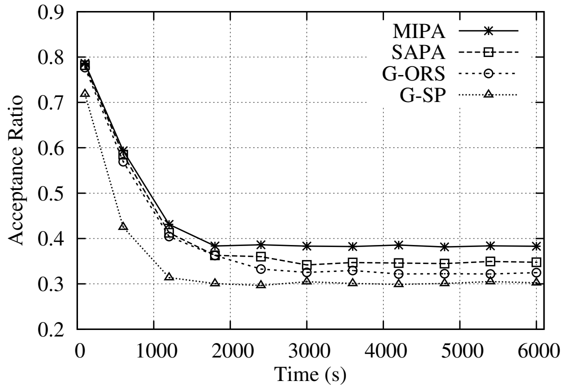

Fig. 11: Comparison results of MIPA, SAPA, G-ORS, and G-SP in the small-setting.

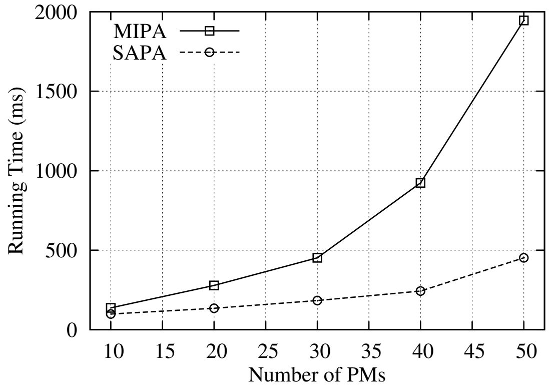

Fig. 12: Running time with varying number of PMs.

The proposed algorithms are compared with *G-ORS* [12] (greedy node mapping with resource sharing) and *G-SP* [13] (greedy node map-

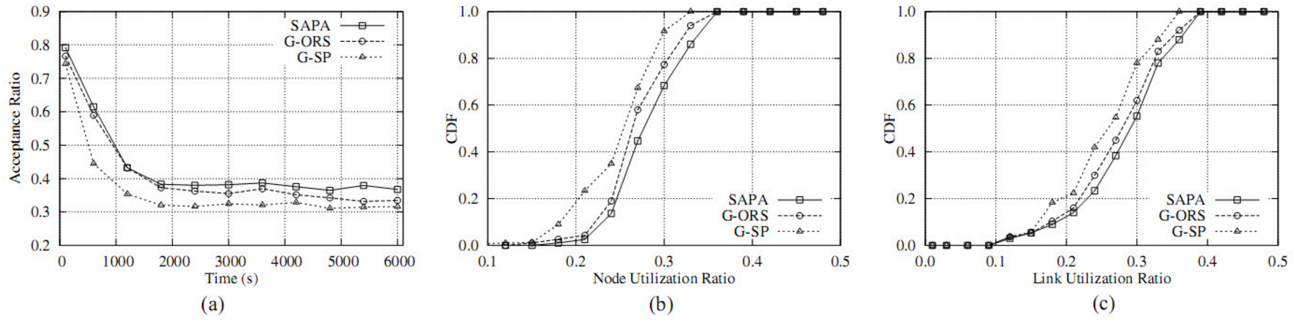

Fig. 13: Comparison results of SAPA, G-ORS, and G-SP in the large-setting. (a) Acceptance ratio over time. (b) CDF of node utilization ratios. (c) CDF of link utilization ratios.

ping with the shortest path-based link mapping). The performance metrics include acceptance ratio, node utilization ratio, and link utilization ratio.

Fig. 11 shows the comparison results on the acceptance ratio of the four algorithms in the small-setting. We note that, MIPA outperforms the other three algorithms, and SAPA achieves the second-highest acceptance ratio. Specifically, the average acceptance ratios of MIPA, SAPA, G-ORS, and G-SP after the first 1,200 seconds are 38.29%, 34.98%, 33.00%, and 30.11%, respectively, which means that MIPA accepts up to 8.18% more requests than G-SP. SAPA performs better than G-ORS and G-SP, as it employs simulated annealing as its optimization framework and takes local resource sharing into account.

Fig. 12 shows the comparison results on the running time of MIPA and SAPA. In this evaluation, we keep the number of VMs fixed at five, and set the iteration count in SAPA to be 1,000. We note that, due to the high time complexity of MIP formulation, the running time of MIPA goes up quickly as the number of physical machines in a cloud increases. For instance, MIPA costs about two seconds when the physical network contains 50 PMs. SAPA has a much smaller time complexity, and has an almost linear running time.

Figs. 13(a), 13(b), and 13(c) show the comparison results on the acceptance ratio, node utilization ratio, and link utilization ratio, respectively, of SAPA, G-ORS, and G-SP in the large-setting. In Fig. 13(a), the average acceptance ratio of these three algorithms after the first 2,000 seconds is around 0.37, which is larger than 0.35 in Fig. 11. The main reason is that, the physical network in the large-setting has a much higher diversity than that in the small-setting, and such a kind of diversity will further enable the physical network in the large-setting to accept more virtual network requests. The average acceptance ratios of SAPA, G-ORS, and G-SP after the first 1,200 seconds are 38.35%, 36.14%, and 32.30%, respectively, which implies that SAPA accepts up to 6.05% more virtual network requests than G-SP. Figs. 13(b) and 13(c) show the cumulative distribution function (CDF) of node and link utilization ratios, respectively. We notice that, the node and link utilization ratios in SAPA are, on average, higher than those in the other two algorithms. The average node utilization ratios of SAPA, G-ORS, and G-SP are 29.51%, 28.31%, and 26.12%, respectively; that is, SAPA utilizes up to 3.39% more physical computing resources than G-SP.

7.2 Flexibility in Providing Tradeoffs

In this subsection, we present simulation results on the ability of SAPA in providing flexible resource allocations. In Fig. 14(a), we show

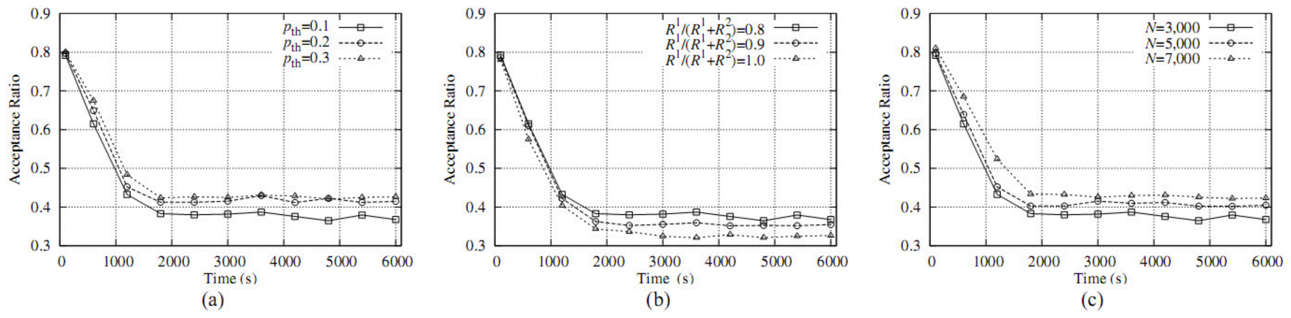

Fig. 14: Evaluations on the flexibilities of SAPA. (a) Tradeoff between performance guarantee and resource utilization. (b) Tradeoff between application performance and placement cost. (c) Tradeoff between placement optimality and running time.

the acceptance ratio of SAPA under three different values of collision threshold, *i.e.*, $p_{\rm th}$. We notice that, SAPA with a larger $p_{\rm th}$ accepts more R-LVN requests than SAPA with a smaller $p_{\rm th}$. The main reason behind this phenomenon is that, when a cloud provider increases the collision threshold, more variable parts of resource demands could coexist in a single physical time slot, which improves physical resource utilization. Therefore, a cloud provider can control the trade-off between performance guarantee and resource utilization through adjusting the collision threshold.

In Fig. 14(b), we present the acceptance ratio of SAPA under three different settings of RLVN requests. We denote by "$R^1/(R^1+R^2)$" the average percentage of the basic part in the total resource demand. When the percentage increases from 0.8 to 1.0, more variable parts of resource demands turn into basic parts, which do not permit local resource sharing. Therefore, the acceptance ratio of SAPA decreases. Since a cloud provider charges a tenant a smaller amount of rent for variable parts of resource demands than that for basic parts, a tenant then can control the trade-off between application performance and placement cost through adjusting the percentage.

Fig. 14(c) shows the acceptance ratio of SAPA with different iteration counts. Generally, more iterations make SAPA perform better. That is, SAPA can generate better allocation results at the expense of time efficiency. It is worth noting that, the performance gain becomes smaller as the number of iterations increases. The cloud providers can control the trade-off between placement optimality and running time through modifying the number of iterations.

In summary, simulation results confirm the respective advantages of both MIPA and SAPA. We wish that the proposed algorithms would provide some potential insights into the future research in this direction.

8 Related Work

This is a rich heritage of studies in cloud resource allocation and virtual network placement that has informed and inspired our work. We describe a subset of these efforts below.

The ability to provide scalable resource on demand is central to cloud computing. Server and network virtulization multiplexes and shares physical resources between cloud tenants, which finally translates into increased provider revenue and decreased tenant cost. A large amount of related solutions have been proposed in the past. Want *et al.* [26] considered bin packing-based virtual machine consolidation with dynamic bandwidth demands. Meng *et al.* [27] focused on network-aware virtual machine placement that minimizes average traffic latency incurred by network infrastructures. Zhang *et al.* [28] proposed a heterogeneity-aware resource management system for dynamic capacity provisioning in cloud environments. Wei *et al.* [29] investigated a QoS (*i.e.*, deadline and budget) constrained resource allocation problem and designed an evolutionary mechanism from the perspective of game theory. Zhang *et al.* [30] proposed a bidding language and an online auction mechanism for cloud resource allocations where users with heterogeneous demands come and leave on the fly. To improve cloud task execution performance, Di and Wang [31] studied the problem of minimizing cloud task makespan under a budget with possible prediction errors and proposed the *ODRA* algorithm. They also investigated the problem of optimizing multi-attribute resource allocation in self-organizing clouds [32]. Some other researchers investigated resource allocation problems in mobile cloud computing [33, 34, 35]. Different from these research efforts, our work aims to provide an efficient and flexible resource allocation algorithm with probabilistic performance guarantee.

Energy efficiency is an important issue in data centers. Gao *et al.* [36] noted the location-specific carbon footprint and electricity price of data centers, and proposed to dynamically control the fraction of user-generated traffic directed to each geographically-distributed data center. Wang *et al.* [37] explored several unique features of data centers, *e.g.*, topology regularity, application characteristics, and so on, to improve the energy efficiency in data center networks. In order to provide balanced and scalable data center architectures, recent studies [2, 38, 39, 40] have proposed several novel architectures, *e.g., VL2, Fat-tree, DCell, and BCube.* Zhou *et al.* [41] proposed augmenting physical cloud networks with 60 GHz wireless links. Ballani *et al.* [42] studies pricing strategies for cloud resources. Ghazar and Samaan [15] designed incentive mechanisms for cloud tenants to proactively regulate their resource demands through exploiting the resource utilization fluctuation in data centers and the delay-tolerant nature of many applications. These studies are complementary to our design, and can be used together with our proposed algorithms to provide better performance.

Virtual network placement is the key challenge in network virtualization environments. To cope with its NP-hardness [20], Ricci *et al.* [43] designed a meta-heuristic-based algorithm. Zhu and Ammar [13] studied how to achieve load balance in placing virtual networks in a physical network with unlimited resources. Yu *et al.* [7] envisioned substrate support for path splitting. Lischka and Karl [44] designed a subgraph isomorphism detection-based virtual network placement algorithm. Chowdhury *et al.* [8] proposed the *ViNE-Yard* framework, where only a special case of location constraint was taken into account. Cheng *et al.* [45] incorporated topology-awareness into embedding virtual networks, and designed a Markov chain-based node ranking algorithm.

To cope with physical node or link failure, Koslovski *et al.* [17] and Yeow *et al.* [18] designed methods for improving virtual network reliability through reserving redundant physical resources. Based on column generation and p-cycle techniques, Jarray and Karmouch [19] proposed augmenting virtual networks with redundant nodes and links to achieve fault-tolerant virtual network embedding. Comparatively, our work takes time-varying resource demands and location constraints into consideration, and focuses on flexible and efficient resource allocation.

9 Conclusions

In this paper, we study how to efficiently and flexibly place virtual networks with dynamic resource demands and physical location requirements in a cloud. We first propose a novel virtual network model that allows cloud tenants to better specify their resource demands; we then propose two algorithms with different designing goals. MIPA focuses on optimizing physical resource utilization, while SAPA concentrates on providing flexible resource allocation. Simulations results demonstrate the advantages of the proposed algorithms. In our future work, we plan to design incentive mechanisms for cloud providers to make tenants proactively regulate their resource demands.

References

[1] Armbrust M, Fox A, Griffith R *et al.* A view of cloud computing. *Communications of the ACM*, 2010, 53(4): 50–58.

[2] Greenberg A, Hamilton J R, Jain N *et al.* VL2: a scalable and flexible data center network. In *Proc. ACM SIGCOMM 2009 Conference*, Aug. 2009, pp.51–62.

[3] Ballani H, Costa P, Karagiannis T *et al.* Towards predictable datacenter networks. In *Proc. ACM SIGCOMM 2011 Conference*, Aug. 2011, pp.242–253.

[4] Xu F, Liu F, Jin H *et al.* Managing performance overhead of virtual machines in cloud computing: a survey, state of the art, and future directions. *Proceedings of the IEEE*, 2014, 102(1): 11–31.

[5] Guo C, Lu G, Wang H *et al.* Secondnet: a data center network virtualization architecture with bandwidth guarantees. In *Proc. the 6th ACM International Conference on emerging Networking EXperiments and Technologies*, Nov. 2010, pp.15–26.

[6] Xie D, Ding N, Hu Y C *et al.* The only constant is change: incorporating time-varying network reservations in data centers. In *Proc. ACM SIGCOMM 2012 Conference*, Aug. 2012, pp.199–210.

[7] Yu M, Yi Y, Rexford J *et al.* Rethinking virtual network embedding: substrate support for path splitting and migration. *ACM SIGCOMM Computer Communication Review*, 2008, 38(2): 17–29.

[8] Chowdhury M, Rahman M, Boutaba R. ViNEYard: Virtual network embedding algorithms with coordinated node and link mapping. *IEEE/ACM Transations on Networking*, 2012, 20(1): 206–219.

[9] Duan Q, Yan Y, Vasilakos A V. A survey on service-oriented network virtualization toward convergence of networking and cloud computing. *IEEE Transactions on Network and Service Management*, 2012, 9(4): 373–392.

[10] Ahuja R K, Magnanti T L, Orlin J B. Network flows: theory, algorithms, and applications. Prentice hall, 1993.

[11] Kirkpatrick S. Optimization by simmulated annealing: quantitative studies. *Journal of Statistical Physics*, 1984, 34(5/6): 975–986.

[12] Zhang S, Qian Z Z, Wu J *et al.* Virtual network embedding with opportunistic resource sharing. *IEEE Transactions on Parallel and Distributed Systems*, 2014, 25(3): 816–827.

[13] Zhu Y, Ammar M. Algorithms for assigning substrate network resources to virtual network components. In *Proc. the 25th Annual*

IEEE International Conference on Computer Communications, Apr. 2006, pp.1–12.

[14] Vogels W. Beyond server consolidation. *ACM Queue*, 2008, 6(1): 20–26.

[15] Ghazar T, Samaan N. Pricing utility-based virtual networks. *IEEE Transactions on Network and Service Management*, 2013, 10(2): 119–132.

[16] Agarwal S, Kandula S, Bruno N *et al.* Re-optimizing data-parallel computing. In *Proc. the 9th USENIX Symposium on Networked Systems Design and Implementation*, Apr. 2012, pp.281–294.

[17] Koslovski G, Yeow W L, Westphal C *et al.* Reliability support in virtual infrastructures. In *Proc. the 2nd IEEE International Conference on Cloud Computing Technology and Science*, Nov. 2010, pp.49–58.

[18] Yeow W L, Westphal C, Kozat U C. Designing and embedding reliable virtual infrastructures. *ACM SIGCOMM Computer Communication Review*, 2011, 41(2): 57–64.

[19] Jarray A, Karmouch A. Cost-efficient mapping for fault-tolerant virtual networks. *IEEE Transactions on Computers*, 2014, PrePrints.

[20] Andersen D G. Theoretical approaches to node assignment. Technical Report of Computer Science Department, Carnegie Mellon University, Dec. 2002.

[21] Vijay V V. Approximation Algorithms. Springer, 2003.

[22] Mitzenmacher M, Upfal E. Probability and computing: Randomized algorithms and probabilistic analysis. Cambridge University Press, 2005.

[23] Anagnostopoulos A, Michel L, Hentenryck P V *et al.* A simulated annealing approach to the traveling tournament problem. *Journal of Scheduling*, 2006, 9(2): 177–193.

[24] Osman I H. Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. *Annals of Operations Research*, 1993, 41(4): 421–451.

[25] Zhang S, Qian Z Z, Guo S *et al.* FELL: a flexible virtual network embedding algorithm with guaranteed load balancing. In *Proc. the 47th IEEE International Conference on Communications*, Jun. 2011, pp.1–5.

[26] Wang M, Meng X, Zhang L. Consolidating virtual machines with dynamic bandwidth demand in data centers. In *Proc. the 30th Annual IEEE International Conference on Computer Communications*, Apr. 2011, pp.71–75.

[27] Meng X, Pappas V, Zhang L. Improving the scalability of data center networks with traffic-aware virtual machine placement. In *Proc. the 29th Annual IEEE International Conference on Computer Communications*, Apr. 2010, pp.1–9.

[28] Zhang Q, Zhani M F, Boutaba R *et al.* Harmony: Dynamic heterogeneity-aware resource provisioning in the cloud. In *Proc. the 33rd International Conference on Distributed Computing Systems*, Jul. 2013, pp.510–519.

[29] Wei G, Vasilakos A V, Zheng Y *et al.* A game-theoretic method of fair resource allocation for cloud computing services. *The Journal of Supercomputing*, 2010, 54(2): 252–269.

[30] Zhang H, Li B, Jiang H *et al.* A framework for truthful online auctions in cloud computing with heterogeneous user demands. In *Proc.*

the 32nd Annual IEEE International Conference on Computer Communications, Apr. 2013, pp.1510–1518.

[31] Di S, Wang C L. Minimization of cloud task execution length with workload prediction errors. In *Proc. the 20th International Conference on High Performance Computing*, Dec. 2013, pp.69–78.

[32] Di S, Wang C L. Dynamic optimization of multiattribute resource allocation in self-organizing clouds. *IEEE Transactions on Parallel and Distributed Systems*, 2013, 24(3): 464–478.

[33] Rahimi M R, Ren J, Liu C H *et al.* Mobile cloud computing: a survey, state of art and future directions. *Mobile Networks and Applications*, 2013, 19(2): 133–143.

[34] Rahimi M R, Venkatasubramanian N, Mehrotra S *et al.* Mapcloud: mobile applications on an elastic and scalable 2-tier cloud architecture. In *Proc. the 5th IEEE/ACM International Conference on Utility and Cloud Computing*, Nov. 2012, pp.83–90.

[35] Rahimi M R, Venkatasubramanian N, Vasilakos A V. Music: Mobility-aware optimal service allocation in mobile cloud computing. In *Proc. the 6th IEEE International Conference on Cloud Computing*, Jun. 2013, pp.75–82.

[36] Gao P X, Curtis A R, Wong B *et al.* It's not easy being green. In *Proc. ACM SIGCOMM 2012 Conference*, Aug. 2012, pp.211–222.

[37] Wang L, Zhang F, Aroca J A *et al.* GreenDCN: a general framework for achieving energy efficiency in data center networks. *IEEE Journal on Selected Areas in Communications*, 2014, 32(1): 4–15.

[38] Al-Fares M, Loukissas A, Vahdat A. A scalable, commodity data center network architecture. In *Proc. ACM SIGCOMM 2008 Conference*, Aug. 2008, pp.63–74.

[39] Guo C, Wu H, Tan K *et al.* Dcell: a scalable and fault-tolerant network structure for data centers. In *Proc. ACM SIGCOMM 2008 Conference*, Aug. 2008, pp.75–86.

[40] Guo C, Lu G, Li D *et al.* Bcube: a high performance, server-centric network architecture for modular data centers. In *Proc. ACM SIGCOMM 2009 Conference*, Aug. 2009, pp.63–74.

[41] Zhou X, Zhang Z, Zhu Y *et al.* Mirror mirror on the ceiling: flexible wireless links for data centers. In *Proc. ACM SIGCOMM 2012 Conference*, Aug. 2012, pp.443–454.

[42] Ballani H, Costa P, Karagiannis T *et al.* The price is right: Towards location-independent costs in datacenters. In *Proc. the 10th ACM Workshop on Hot Topics in Networks*, Nov. 2011, pp.23–28.

[43] Ricci R, Alfeld C, Lepreau J. A solver for the network testbed mapping problem. *ACM SIGCOMM Computer Communication Review*, 2003, 33(2): 65–81.

[44] Lischka J, Karl H. A virtual network mapping algorithm based on subgraph isomorphism detection. In *Proc. the 1st ACM workshop on Virtualized infrastructure systems and architectures*, Mar. 2009, pp.81–88.

[45] Cheng X, Su S, Zhang Z *et al.* Virtual network embedding through topology-aware node ranking. *ACM SIGCOMM Computer Communication Review*, 2011, 41(2): 38–47.

Sheng Zhang received his BS and PhD degrees from Nanjing University in 2008 and 2014, respectively. Currently, He is an assistant lecturer in the Department of Computer Science and Technology, Nanjing University. He is also a member of the State Key Laboratory for Novel Software Technology. His research interests include network virtualization, cloud computing, and mobile networks. To date, he has published more than 15 papers, including those appeared in IEEE Transactions on Parallel and Distributed Systems, IEEE Transactions on Computers, ACM MobiHoc, and IEEE INFOCOM. He received the Best Paper Runner-Up Award from IEEE MASS 2012.

Zhu-Zhong Qian is an associate professor at the Department of Computer Science and Technology, Nanjing University. He is also a member of the State Key Laboratory for Novel Software Technology. He received his PhD. Degree in computer science in 2007. Currently, his research interests include cloud computing, distributed systems, and pervasive computing. He is the chief member of several national research projects on cloud computing and pervasive computing. He has published more than 30 research papers in related fields. He is a member of CCF and IEEE.

Jie Wu is the chair and a Laura H. Carnell professor in the Department of Computer and Information Sciences at Temple University. He is also an Intellectual Ventures endowed visiting chair professor at the National Laboratory for Information Science and Technology, Tsinghua University. Prior to joining Temple University, he was a program director at the National Science Foundation and was a Distinguished Professor at Florida Atlantic University. His current research interests include mobile computing and wireless networks, routing protocols, cloud and green computing, network trust and security, and social network applications. Dr. Wu regularly publishes in scholarly journals, conference proceedings, and books. He serves on several editorial boards, including IEEE Transactions on Service Computing and the Journal of Parallel and Distributed Computing. Dr. Wu was general co-chair/chair for IEEE MASS 2006, IEEE IPDPS 2008, and IEEE ICDCS 2013, as well as program co-chair for IEEE INFOCOM 2011 and CCF CNCC 2013. Currently, he is serving as general chair for ACM MobiHoc 2014. He was an IEEE Computer Society Distinguished Visitor, ACM Distinguished Speaker, and chair for the IEEE Technical Committee on Distributed Processing (TCDP). Dr. Wu is a CCF Distinguished Speaker and a Fellow of the IEEE. He is the recipient of the 2011 China Computer Federation (CCF) Overseas Outstanding Achievement Award.

Sang-Lu Lu received her Ph.D. degree in computer science from Nanjing University in 1997. She is currently a professor in the Department of Computer Science and Technology and the State Key Laboratory for Novel Software Technology. Her research interests include distributed computing, wireless networks, and pervasive computing. She has published over 80 papers in referred journals and conferences in the above areas. She is a member of CCF and IEEE.