Elastic and Efficient Virtual Network Provisioning for Cloud-Based Multi-Tier Applications

Meng Shen*, Ke Xu†, Fan Li*, Kun Yang‡ §, Liehuang Zhu*, and Lei Guan¶
* Beijing Engineering Research Center of High Volume Language Information Processing and Cloud Computing Applications
School of Computer Science, Beijing Institute of Technology, P. R. China
†Department of Computer Science, Tsinghua University, P. R. China
‡School of Computer Science & Electronic Engineering, University of Essex, UK
§The ISN National Key Lab, XiDian University, P. R. China
¶School of Management & Economics, Beijing Institute of Technology, P. R. China
{shenmeng, fli, liehuangz, guanlei}@bit.edu.cn; xuke@mail.tsinghua.edu.cn; kunyang@essex.ac.uk

***Abstract*—The multi-tier architecture is prevalently adopted by cloud applications, such as the three-tier web applications. It is highly desirable for both tenants and cloud providers to have an efficient and elastic approach for virtual network provisioning, where tenant applications can automatically scale in or out with varying workloads and providers can accommodate as many requests as possible in the underlying network. However, due to potential conflicts between efficiency and elasticity, it is challenging to achieve these two goals simultaneously, in abstracting tenant requirements and designing corresponding provisioning algorithms.**

In this paper, we propose an efficient and elastic virtual network provisioning solution called EasyAlloc, which is comprised of an *elasticity-aware* abstraction model and a virtual network provisioning algorithm. To accurately capture the tenant requirement and maintain the provisioning simplicity for providers, the elasticity-aware model leverages two types of decoupling, *i.e.*, *always-on* VMs for normal load and *on-demand* VMs for dynamic scaling, and the bandwidth requirement of each VM for intra- and inter-tier communications. Then we formulate the virtual network provisioning as an overhead minimization problem, where the objective simultaneously considers the bandwidth and elasticity overhead. Due to computational complexity of this problem, we leverage two heuristics, slot reservation and tier iteration, to obtain an efficient algorithm. Extensive simulation results show that compared with a typical elasticity-agnostic method under a heavy load, EasyAlloc enables a 9% increase of request acceptance rate and a 16.8% improvement of the successful extension rate. To the best of our knowledge, this is the first work targeting at the elastic virtual network provisioning.

I. Introduction

Recent years have witnessed a growing trend in migrating applications onto public cloud platforms [12], such as Amazon EC2 [2], Microsoft Azure [3] and Aliyun [1]. Many of these applications are complex combinations of multiple service components, which form multi-tier virtual networks. The number of tiers varies by business and application requirements, and a three-tier architecture is commonly used as shown in Figure 1. The Presentation tier is responsible for managing the user interaction and providing an easy-to-operate front end. The business rules are managed by the business logic tier, which controls and operates the entire application framework. The underlying data is stored and served by the data storage tier. Each tier consists of one or more virtual instances running on individual virtual machines (VMs).

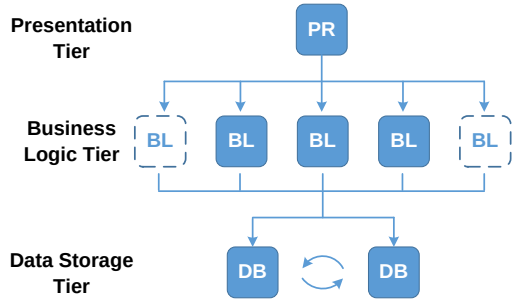


Fig. 1. An example of a three-tier application

Two problems are of great importance to the provisioning of virtual networks for multi-tier applications. From the perspective of application owners (also known as tenants), their applications should avoid performance degradation in the face of workload variations, including either predictable load changes or unforceable burst load. Therefore, there has long been an expectation from tenants to establish the ***elasticity*** of their virtual networks, where a virtual network is capable of automatically scale in or out with varying loads. For instance, as shown in Figure 1, a three-tier web application relies on three business logic (BL) instances for normal load, and automatically launches two additional BL instances marked in dashed boxes for heavy load (*e.g.*, the average CPU utilization of the three existing instances is larger than 80%). From the perspective of cloud operators, they always want to improve the ***efficiency*** of virtual resources utilization in the cloud, *e.g.*, improving the request acceptance rate, or increasing the revenue-to-cost ratio so as to maximize the profit of virtual network provisioning [10].

However, it is challenging to achieve these two goals simultaneously due to their inherent conflicts in the following two aspects. As the first step towards virtual network provisioning, there should be an appropriate *abstraction model* that is capable of describing tenants' virtual network requirements and facilitating the resource allocation for cloud providers. Since applications belonging to different tenants compete for the limited bandwidth and thereby suffer from unpredictable task completion time, the abstraction model usually specifies the VM and bandwidth requirements for each virtual network request. Several abstraction models have been

proposed in recent years, which can be roughly classified as the provider-oriented [4, 5, 26] and the application-oriented [18]. The former designs hose-based models that resemble the underlying datacenter topology. These models can offer cloud providers the simplicity in resource allocation, but prevent them to accurately computing the required bandwidth and lead to inefficient bandwidth provisioning. The latter derives a model based on application communication structure, but fails to facilitate the provisioning process. Besides, none of the prior work pays attention to the elastic requirement of tenants. Therefore, it remains a challenge to fill in the gap between precisely capturing elastic application requirements and efficiently provisioning in the underlying network.

The second challenge lies in designing a virtual network provisioning algorithm. Many existing algorithms aim to improve the efficiency for static and deterministic virtual network requirements, by globally optimizing the VM placement and bandwidth allocation strategies, such as Octopus [4]. However, the elasticity of virtual networks might make these algorithms inefficient. For instance, colocation is a typical optimization strategy for virtual network provisioning, which attempts to accommodate as many VMs as possible in a local area and thereby reduces the cross-core network bandwidth consumption. In the scenario of elastic virtual networks, however, the colocation strategy is likely to make the extended virtual components located far away from the existing components (*e.g.*, in a rack under another aggregation switch), because slots in the same rack are already assigned to other tenants. As a result, providers have to reserve even more bandwidth on core links to guarantee the communications between components of the same application.

In this paper, we present a new virtual network provisioning solution called *EasyAlloc*, which achieves efficiency and elasticity at the same time. EasyAlloc is comprised of an *elasticity-aware* abstraction model and a virtual network provisioning algorithm. To the best of our knowledge, this is the first work targeting at the elastic virtual network provisioning problem.

To capture the elastic nature of multi-tier applications, we classify VMs in each tier as *always-on* VMs for normal load and *on-demand* VMs for dynamic scaling with certain predefined launching conditions. Based on observations of communication characteristics for multi-tier applications, the elasticity-aware model divides the bandwidth requirement for each VM into intra- and inter-tier requirements, respectively. Such a decoupling makes a balance between exiting provider- and application-oriented abstractions, which keeps the simplicity for resource allocation and avoids the bandwidth over-provisioning.

Given a request described by the elasticity-aware model, we formulate the virtual network provisioning as an optimization problem that attempts to minimize the provisioning overhead. The metrics of overhead are carefully designed to consider the interference of a newly arrived request with future extensions of existing requests. The algorithm devised based on the mathematic formulation can maintain benefits of colocation while satisfying elastic requirements of requests.

In this paper, we make the following contributions.

- We propose a new elasticity-aware model to abstract the virtual network requirement for multi-tier applications. This model enables tenants to express their elastic network requirements, and offers providers the simplicity and efficiency for provisioning.
- We formulate the virtual network provisioning as an overhead minimization problem. Since provider efficiency and request elasticity can be conflicting goals, we propose a metric to quantify the overhead of elasticity, and jointly optimize the bandwidth and elasticity overhead in the formulation. Due to the NP-completeness of this problem, we leverage two heuristics, slot reservation and tier iteration, to obtain an efficient algorithm.
- We evaluate the benefits of the EasyAlloc through extensive simulations using synthetic application requests. Experiment results show that, compared with a typical elasticity-agnostic method in [4] under a heavy datacenter load, EasyAlloc enables a 9% increase of request acceptance rate and 16.8% more requests to successfully scale as expected.

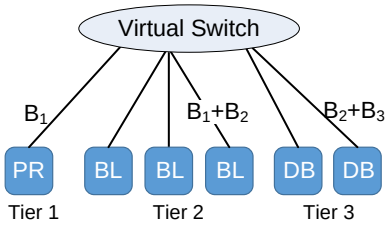


Fig. 2. Characterizing the virtual network requirement for the multi-tier application using the virtual cluster model in Octopus [4].

The rest content of this paper is organized as follows. We present the new elasticity-aware abstraction model in Section II, based on which the virtual network provisioning problem is mathematically formulated in Section III. The algorithm is devised in Section IV followed by the performance evaluation in Section V. After briefly summarizing the related work in Section VI, we conclude this paper in Section VII.

II. Virtual Network Abstraction

A. Design Goals

In public cloud datacenters, tenants request virtual networks with a certain amount of virtual resources, *e.g.*, CPU, memory and storage. For ease of exposition, existing literatures [4, 26] usually abstract the non-network resources in the form of VMs, which is also consistent with the case in commercial public platforms such as Amazon EC2 [2]. In order to capture bandwidth requirements of tenants, the hose model is incorporated into the virtual network abstraction to provide a desired amount of bandwidth guarantee for each VM. However, existing hose model is not suitable for abstracting the elastic virtual networks because of low efficiency and lack of elasticity.

Now we use the application in Figure 1 as an example to show the inefficiency of existing models. Here we only consider the virtual instances in solid boxes. Assume that B_1 represents the bandwidth demand between each VM in the presentation tier (Tier 1) and each VM in the business logic tier (Tier 2), and B_2 is the bandwidth demand between each VM

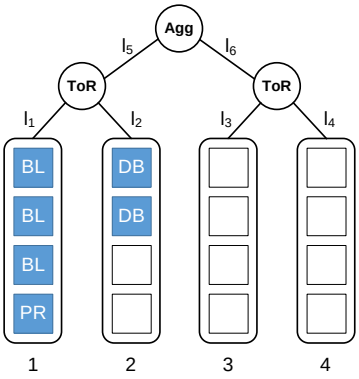


Fig. 3. An allocation example that shows the virtual cluster model might lead to bandwidth over-provisioning.

in the logical tier and each VM in the data storage tier (Tier 3). Besides, there is also a bandwidth demand of B_3 between two VMs in the data storage tier. In the virtual cluster model in Octopus [4], each VM is connected to a central virtual switch by a dedicated link (*i.e.*, hose) with its desired bandwidth demand. Accordingly, the virtual network abstraction is exhibited in Figure 2, where the link capacity denotes the aggregate bandwidth demand for each VM, irrespective of inter-tier or intra-tier communication. Such an abstraction can be easily accommodated in a tree-like topology, as illustrated in Figure 3, where Tier 1 and Tier 2 are placed in Server 1 and Tier 3 is placed in Server 2.

Take link l_2 for example, the total bandwidth requirement of VMs that reside in its subtree (*i.e.*, Server 2) is $l_2^{in} = 2B_2 + 2B_3$, while the total bandwidth requirement of VMs that reside outside its subtree is $l_2^{out} = B_1 + 3(B_1 + B_2) = 4B_1 + 3B_2$. To satisfy the bandwidth requirement expressed in the hose model, the amount of bandwidth reserved on link l_3 should be the minimum of l_2^{in} and l_2^{out}. Assume that l_2^{in} is less than l_2^{out}, so the bandwidth reserved for the data storage tier is $2B_2 + 2B_3$. However, since B_3 is used for communication between VMs in the same tier, it is unnecessary to reserve B_3 on link l_2. The reason of the over-provision problem lies in that the hose-based abstraction emulates MapReduce-like all-to-all traffic patterns, and thus does not fit the multi-tier communication pattern.

Another shortcoming of existing abstractions is that they are designed for static and deterministic virtual network requests and thus fail in capturing the elastic requirement of tenants. For instance, a parameter N is employed in the virtual cluster model to denote the number of VMs requested, which keeps unchanged during the lifetime of a virtual network.

Based on the above observations, the virtual network abstraction is guided by the following three design goals:

Simplicity. The abstraction should provide simplicity that allows 1) tenants to express their requirements easily and to reason in an intuitive way about their application performance, and 2) providers to carry on virtual network provisioning and resource enforcement according to tenant requirements.

Elasticity. The abstraction should be capable of capturing the elasticity of virtual networks, *i.e.*, automatically scale in or out with varying workloads.

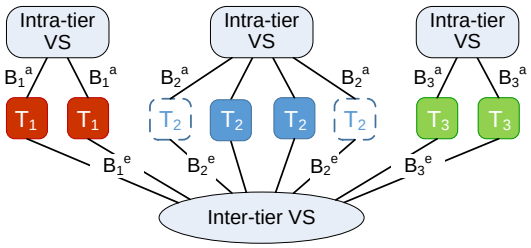


Fig. 4. An example of the elasticity-aware model that characterizes the virtual network requirement for a three-tier application. Tiers are in different colors.

Efficiency. Providers should be able to satisfy tenant requirements with as few physical resources as possible. The less the over-provisioning occurs, the more providers benefit.

B. The Elasticity-Aware Model

To achieve the design goals, we propose a new abstraction model for elastic multi-tier applications called the *elasticity-aware* model.

As shown in the previous example, the communication pattern in multi-tier applications consists of the communication between VMs in the same tier (*i.e.*, intra-tier communication), and the communication between VMs across tiers (*i.e.*, inter-tier communication). In contrast to the aggregate bandwidth requirement in Octopus, we decouple the intra-tier bandwidth from the inter-tier bandwidth for each VM. By decoupling, the elasticity-aware model can accurately capture the bandwidth requirement and mitigate the over-provisioning.

In order to meet the requirement of elasticity, we classify the VMs in each tier into two categories, *i.e.*, *always-on* VMs and *on-demand* VMs. The always-on VMs is used by applications to deal with the normal workload, which can be viewed as the minimum VM requirement to guarantee the worst-case performance. While the on-demand VMs serve as supplementary VMs that can be dynamically launched for load balancing during peak hours, or in the case of burst loads.

The virtual network requirement of a k-tier application can be represented by $T = \{T_i, i \in [1, k]\}$, where T_i for the i-th tier is a quintuple that takes the form of $\langle N_i^s, N_i^d, B_i^a, B_i^e, S_i \rangle$. The meaning of each element is as follows

- N_i^s denotes the number of the *always-on* VMs.
- N_i^d denotes the number of the *on-demand* VMs.
- B_i^a is the bandwidth requested for the *intra-tier* communication.
- B_i^e is the bandwidth requested for the *inter-tier* communication.
- S_i describes policies or conditions for the dynamic scaling. For instance, it can be *scaling out when the average CPU utilization is larger than 80%*. In other words, N_i^d VMs will be launched when S_i is satisfied.

To better understand this model, we illustrate how to express the virtual network requirement for a three-tier application, as shown in Figure 4. The three tiers are denoted by T_1, T_2 and T_3, respectively. Each tier requests two always-on VMs, and T_2 requests two on-demand VMs. In Figure 4, tiers

are distinguished by colors. Each VM in a tier is connected to an intra-tier virtual switch (VS) by a dedicated virtual link with the bandwidth guarantee for its intra-tier communication. For instance, two VMs in T_1 are connected to the intra-tier VS with the bandwidth of B_1^a. All VMs in the three tiers are connected to an inter-tier virtual switch by dedicated virtual links with the capacity of their respective inter-tier bandwidth guarantee. Note that the on-demand VMs are represented by dashed boxes, which indicates that these VMs are valid when the pre-defined scaling conditions are met. Here we assume the on-demand and always-on VMs in the same tier have the same bandwidth requirement. This is reasonable because VMs in the same tier usually perform similar logical functions and thus have a stable per-VM bandwidth requirement [18].

Let's revisit the example in Figure 3. By using the proposed elasticity-aware model, the bandwidth requirement of a VM at the DB tier is now divided into the intra-tier bandwidth (*i.e.*, B_3) and the inter-tier bandwidth (*i.e.*, B_2). Therefore, the bandwidth reserved on link l_2 for VMs in Server 2 is $2B_2$, which is less than $l_2^i n$ derived from the virtual cluster model.

The elasticity-aware model enables tenants to express their requirements, in terms of VM, bandwidth and dynamic scaling. And it has the potential to meet tenant requests with less underlying resources.

III. Virtual Network Provisioning Formulation

In this section, we mathematically formulate the virtual network provisioning problem with a joint consideration of provisioning efficiency and request elasticity.

A. Shortcoming of Typical Provisioning Goal

Upon receiving a virtual network request expressed by the elasticity-aware model, the provider should decide whether to accept and how to accommodate such a request. A virtual network request can be accepted if there exists a provisioning solution that satisfies the following constraints:

- Bandwidth constraints: the bandwidth requirement on each physical link should not exceed the residual capacity of that link.
- VM constraints: the VM requirement on each physical server should not exceed the available empty slots in that server.

A solution that meets the above constraints is a *feasible* solution. Since the available physical resources in datacenters usually far more than what is requested in an individual virtual network, the feasible solution might not be unique. Therefore, our task is to find the *best* solution from the possibilities. However, the criteria of best are closely related to the provisioning goals, which should be determined by considering concerns of the stakeholders.

The fundamental goal of cloud operators is to improve the efficiency of physical resources, *i.e.*, accommodating as many application requests as possible running atop the physical infrastructure. The bandwidth oversubscription is prevalent in today's datacenters [4], which makes the bandwidth become the bottleneck for accepting future requests. According to the measurement results reported in [18], the bandwidth is usually well provisioned at the server level, but not at the ToR or aggregation level due to the oversubscription. Therefore, operators prefer to a solution that can achieve the most bandwidth savings, especially at higher levels in a tree-like topology.

TABLE I. Virtual network requirements described by the elasticity-aware model for two requests

Request Index	Tier Index	Requirement
Request P	Tier 1	$< 1, 0, 0, 150, null >$
	Tier 2	$< 3, 2, 0, 200, s_1 >$
	Tier 3	$< 2, 0, 100, 250, null >$
Request Q	Tier 1	$< 1, 0, 0, 100, null >$
	Tier 2	$< 1, 0, 0, 200, null >$

The goal of tenants is to get the amount of virtual resources specified by the elasticity-aware model, and to ensure their required virtual networks can dynamically scale with the time-varying workloads.

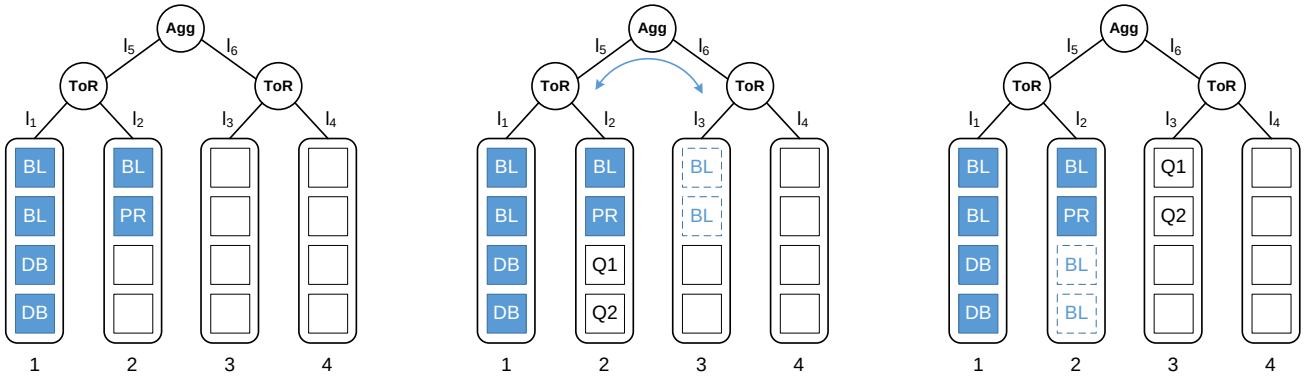
The provisioning efficiency and request elasticity might be conflicting goals. For example, due to the elastic nature of virtual networks requested by applications, an efficient solution right now might turn to be inefficient after launching the on-demand VMs. Here we use the application request in Table I for illustration, where the bandwidth is in the unit of Mbps. We also use the same topology as shown in Figure 3, where each two servers are connected to a ToR switch and two ToR switches are then connected to an aggregation switch. Assume that the capacity of each physical link is 1 Gbps.

At time t_0, the Request P arrives and its always-on VMs should be placed in the empty slots. There are multiple feasible placement solutions, of which the one with the least bandwidth requirement is shown in Figure 5(a), where two BL VMs and two DB VMs are located in Server 1 and the rest VMs are located in Server 2, respectively. The bandwidth reserved on links l_1 and l_2 are both 350 Mbps. (See Appendix for enumeration and comparison of feasible solutions.)

At time t_1, Request Q arrives which requires only two always-on VMs. Since Server 2 has exactly two empty slots after hosting Request P, intuitively, VMs for Request Q will be accommodated in Server 2. This placement can fully utilize the currently loaded servers without increasing bandwidth requirement on physical links. However, at time t_2 when the on-demand VMs for Request P needs to be launched, they should be accommodated in a new server, *e.g.*, Server 3, as shown in Figure 5(b). Then, cross-network traffic are generated due to the communication between VMs on both sides of the aggregation switch. As a result, 400 Mbps of bandwidth should be reserved on links l_3, l_5 and l_6.

In contrast, as shown in Figure 5(c), an alternative placement at time t_1 is hosting VMs for Request Q in a new server, *e.g.*, Server 3. Then, at time t_2 the two on-demand VMs for Request P can be accommodated in Server 2. Such a placement only leads to an increase of bandwidth reserved on links l_1 and l_2, without any influence on links l_3, l_5 and l_6.

By comparing two solutions exhibited in Figures 5(b) and (c), we can find that a "best" solution without considering the



(a) The initial state of the topology in use (b) A greedy placement without considering elasticity (c) An elasticity-aware placement

Fig. 5. Comparison of different VM placements for two applications in Table I. The greedy placement fails to reduce bandwidth consumption.

TABLE II. NOTATIONS FOR VIRTUAL NETWORK PROVISIONING

Notation	Definition
T	A request for a multi-tier application
V	The set of servers in datacenters
L	The set of links in datacenters
r_v	The number of empty slots in server $v \in V$
Ω_v	The set of requests having VMs in server v
c_l	The total capacity of link $l \in L$
r_l	The residual capacity on link $l \in L$
b_l	The bandwidth requirement on link l
f_{iv}	The number of VMs for Tier i in server v
F	The provisioning solution of the request
$\mathbb{C}(F)$	The resource overhead function of the solution F
$\mathbb{E}(F)$	The elasticity overhead function of the solution F
$\Phi(F)$	The aggregate objective function of the solution F

elastic nature of request might leads to the local optimum. However, it is challenging to achieve the global optimum, which requires operators to not only consider the newly arrived requests but also foresee its elasticity in the future.

B. Modeling Virtual Network Provisioning

Given a k-tier tenant application request denoted by $T = \{T_i, i \in [1,k]\}$ with $T_i = \langle N_i^s, N_i^d, B_i^a, B_i^e, S_i \rangle$, operators seeks for the globally optimal provisioning solution. Assume the datacenter underlying network is denoted by $G = (V, L)$, where V and L are the server set and the link set, respectively. Notations are summarized in Table II. Now we are in the position of designing the corresponding mathematical formulation.

Resource Overhead. The typical provisioning goal focuses on minimizing the overhead in terms of the cost of bandwidth reservation on all physical links. Given a provisioning solution $F = \{f_{iv}\}$, the resulting bandwidth reservation on each link, b_l, is determined. Therefore, the overhead denoted by $\mathbb{C}(F)$ is a function of F, which is defined as

$$\mathbb{C}(F) = \sum_{l \in L} w_l \frac{b_l}{c_l} \tag{1}$$

where w_l is the weight of link l and c_l is the total capacity of link l. Here we use the increased link utilization, rather than the increased bandwidth consumption, as the bandwidth overhead metric, because the normalization (*i.e.*, divided by c_l) make it comparable for links with different capacities. We can force the solution to use as little bandwidth of core links as possible by assigning them higher weights. Since the calculation of b_l for the multi-tier application is more complicated than that for the flat application [4, 26], we leave it later in this section.

Elasticity Overhead. As illustrated in Figure 5, when making provisioning decisions for a newly-arrived application request, providers should attempt to guarantee the elasticity of applications that are already deployed. The basic idea is to select a solution that does not interference too much with applications that potentially scale out in the future. We have to make this concept of interference more concrete.

Consider the virtual network requirement for Request Q. There are two empty slots in Server 2, and the application accommodated in Server 2 has exactly two on-demand VMs to be potentially launched in the future. So, if the empty slots are partially or completely allocated to Request Q, then there is not enough room for the scaling out of Request P. We define the amount of interference on existing applications (*e.g.*, Request P) due to the virtual network provisioning of a newly arrived request as the summation of the percentage of the on-demand VMs affected in each server.

Assume that D_T denotes the set of tiers in Request T that only launch the always-on VMs currently. Let Ω_v be the set of requests that have VMs accommodated in server v. The percentage of the on-demand VMs affected at server v, denoted by δ_v, is calculated as follows

$$\delta_v = (1 - \frac{r_v - f_{iv}}{\sum_{T \in \Omega_v} \sum_{j \in D_T} N_j^d}, 0)^+ \tag{2}$$

where $(para, 0)^+$ takes the larger value between $para$ and 0 and thus ensures a non-negative value of δ_v. Now we give the physical explanations of δ_v. If $r_v - f_{iv} \geq \sum_{T \in \Omega_v} \sum_{j \in D_T} N_j^d$, the number of empty slots after allocated to the newly arrived request remains large enough to handle the scaling requirement for on-demand VMs, which means that the new request has no influence on other application requests. So, the resulting δ_v is 0. Whereas if the available VM slots after accommodating the new request is not enough, δ_v reflects the percentage of VMs shortage over the total number of on-demand VMs.

We use the elasticity overhead as a measurement of the total interference across all servers, which is denoted by

$$\mathbb{E}(F) = \sum_{v \in V} \delta_v \tag{3}$$

It is straightforward to derive the aggregate objective function for virtual network provisioning from the resource overhead and the elasticity overhead, *i.e.*,

$$\Phi(F) = \mathbb{C}(F) + \mathbb{E}(F) \tag{4}$$

Based on the above analysis, we formulate the virtual network provisioning problem as an optimization which attempts to find a solution that minimizes the provisioning overhead

$$\text{minimize} \quad \Phi(F) \tag{5a}$$

$$\text{subject to} \quad b_l \leq r_l, \forall l \in L \tag{5b}$$

$$\sum_{i=1}^{k} f_{iv} \leq r_v, \forall v \in V \tag{5c}$$

$$\sum_{v} f_{iv} = N_i, \forall i \in [1, k] \tag{5d}$$

Remarks: Constraint sets in Equations (5b) and (5c) define the bandwidth and VM constraints, respectively. Constraint sets in Equation (5d) ensure that the requested number of VMs for each tier are accommodated in servers.

C. Characterising Bandwidth Requirement

Recall that the calculation of the bandwidth requirement on a link, *i.e.*, b_l in Eq. (5), remains unaddressed. We now focus on characterising the bandwidth requirement given a provisioning solution F.

To accommodate a virtual network request described by the elasticity-aware model, the providers should ensure the bandwidth guarantee specified in the model via allocating the desired amount of bandwidth on physical links. In a datacenter tree topology, a link l bridges the subtree under this link denoted by l^{In} and the rest of tree denoted by l^{Out}. For each tier i in a new application T, assume there are N_i^{In} and N_i^{Out} VMs reside in l^{In} and l^{Out}, respectively. The bandwidth requirement on link l is comprised of two types of communication across this link, *i.e.*, the intra-tier communication for each tier and the inter-tier communication for all tiers. For each type of communication, only the minimum requirement in l^{In} and l^{Out} should be satisfied, which is captured as follows

$$\begin{aligned} b_l = &\sum_{i \in T} \min(N_i^{In} B_i^a,\ N_i^{Out} B_i^a) + \\ &\min(\sum_{i \in T} N_i^{In} B_i^e,\ \sum_{i \in T} N_i^{Out} B_i^e) \end{aligned} \tag{6}$$

where the left and right parts are the intra-tier and inter-tier communication requirements, respectively.

IV. Algorithm Design

Since there is an implicit integer constraint on f_{iv} in Eq. (5), the optimization problem becomes computationally intractable for large-scale datacenters (*e.g.*, with thousands of servers or more). We resort to heuristics to approximately search for the optimal solution.

Algorithm 1: Virtual Network Provisioning

Input: A tree topology $G = (V, L)$ with a $root$ link and a k-tier application request $T = \{T_i\}(1 \leq i \leq k)$ where $T_i =< N_i^s, N_i^d, B_i^a, B_i^e, S_i >$

Output: Virtual network provisioning solution F for Request T

```
1  T' ← the k tiers in T ranked in a descending order by their
     bandwidth requirements, i.e., N_i^s(B_i^a + B_i^e)
2  for each tier T_i ∈ T' do
3      for each link l ∈ L do
4          Calculate its VM feasibility vector FV_l and Q_l
       // start from the root link
5      if alloc (T_i, root, N_i^s) != N_i^s then
6          return false
7  return true

   // Assign n VM slots in the subtree with the
      uplink l to request r
8  Function alloc (r, l, n)
9  if ( level (l) == 0 ) then
       // l is the uplink of a server
10     count ← max { j | FV_l(j) = 1 & j ≤ n }
11 else
       // set(l) is the set of links at the level of
          level(l) − 1 in the subtree of l
12     base ← n − Σ_{l' ∈ set(l)} Q_{l'}
13     for each link l' do
14         n_{l'} ← Q_{l'} + base;    tmp ← alloc(r, l', n_{l'})
15         if tmp != -1 then
16             base ← base − (tmp − Q_{l'})
17             count ← count + tmp
18         else
19             return -1
20 if count ∉ FV_l then
21     return -1
22 return count
```

The challenges for design a heuristic algorithm lie in that, for a k-tier application, the number of possible VM allocation options in a subtree with n empty slots is at a magnitude of $(k+1)^n$, because each slot can be either allocated to one of the k tiers or kept empty. The key to reduce the feasible solution space is to make the base (*i.e.*, $k+1$) and exponent (*i.e.*, n) as small as possible.

We attempt to achieve this goal via two heuristics. The first heuristic is *slot reservation*, which reserves enough VM slots in each server for the extension of existing request before accommodating the new request. In other words, slot reservation is to ensure the value of δ_v in Eq. (2) for each server v being 0. It helps to decrease the exponent. And the second one is *tier iteration*, which ranks the k tiers in a descending order of the bandwidth requirement and then processes one tier at a time. The bandwidth requirement of the i-th tier is measured by the bandwidth requested by the always-on VMs in that tier, *i.e.*, $N_i^s(B_i^a + B_i^e)$.

Algorithm 1 exhibits the procedure to solve the virtual network provisioning problem. The basic idea is to start from the root link of the tree topology and recursively call the function `alloc` to accommodate as many VMs as possible in the subtrees. According to the description of tier iteration, the procedure fist sorts the tiers according to their bandwidth requirements (line 1) and then processes them in turns (lines 3-7). If the requested VMs for all tiers are successfully placed

in the topology, without violating the VM and bandwidth constraints, the procedure ends with the desired provisioning solution.

In a flat application with just one tier, for a link l, it is always feasible to place 0 VM in its the subtree, because the resulting bandwidth requirement on link l is 0. However, this property is no longer valid for the multi-tier application, due to the inter-dependency among tiers as shown by the inter-tier bandwidth requirement in Figure 4. Therefore, we associate each link l with a feasible vector, FV_l, to indicate the number of VMs that can be accommodated in its subtree [23]. In FV_l, the minimum feasible number is recorded in Q_l, which means that at least Q_l VMs should be placed in its subtree.

Since the available slots and residual capacity of links change after each iteration, the procedure updates FV_l and Q_l before processing the next tier (line 4). For the i-th tier, FV_l has N_i^a+1 bits, where $FV_l(j)(1 \leq j \leq N_i^a+1)$ is set to be 0 for infeasibility, or 1 for feasibility. Eq. (6) can be used for the feasibility computation. Besides, it is straightforward to realize the slot reservation using FV_l, *i.e.*, by setting corresponding bits for the number of VMs to be reserved as 0.

In the function `alloc`, if link l is the uplink of a server, we should accommodate as many VMs as possible to take the advantage of colocation [18]; Otherwise, we move to the lower level links. Since each lower-level link l' might have a non-zero $Q_{l'}$, the procedure excludes such VMs (line 12) and allocate the remaining VMs in the subtree of link l' (lines 13-19). The final check at line 20 is to ensure that the aggregate number of VMs allocated in l's subtrees remains feasible.

Now we analyze the algorithm complexity in a tree topology with the link size of $|L|$. For a k-tier application, Algorithm 1 iteratively processes every tier, which invokes a separate top-level recursion each time. The complexity of calculating FV_l for each link l at line 4 is $O(\bar{N})$, and the complexity for each recursion is also $O(\bar{N})$, where $\bar{N}$ is the mean tier size. Therefore, the overall complexity becomes $O(k\bar{N}|L|)$.

V. Performance Evaluation

In this section, we evaluate the proposed algorithm for virtual network provisioning by simulation experiments.

A. Experiment Settings

We simulate a 3-level tree topology inspired by similar settings in exiting literatures [4, 18], with 4,000 servers. For simplicity, we assume a homogeneous configuration (*e.g.*, CPU and memory) across all servers and each server hosts 4 VMs. A rack consists of 40 servers with a 1 Gbps link connected to the corresponding ToR switch. Every 10 ToR switches are connected to an aggregation switch, which are then connected to a unique core switch. The oversubscription ratio is 4 in default at each level.

The synthetic application requests in our simulation are randomly generated by following the metrics in recent literatures [5, 7, 12, 18]. The number of VMs per request is exponentially distributed with a mean size of 50, and the percentages of the always-on and on-demand VMs are roughly 80% and 20%. We assume the largest number of tiers in applications is 3 and only one tier is associated with on-demand VMs. Among all application requests, the percentages of 1-tier (*i.e.*, flat), 2-tier and 3-tier requests are around 20%, 40% and 40%. For each VM at tier i in a request, the bandwidth requirements for inter-tier communications (*i.e.*, B_i^e) and intra-tier communications (*i.e.*, B_i^a) are uniformly distributed in [150, 300] and [50, 100], respectively. Particularly, VMs in applications with just one tier only have bandwidth requirement for intra-tier communication. Each request is also associated with an execution interval. For simplicity, we randomly assign a scaling time to each application request, which indicates the time to launch its on-demand VMs.

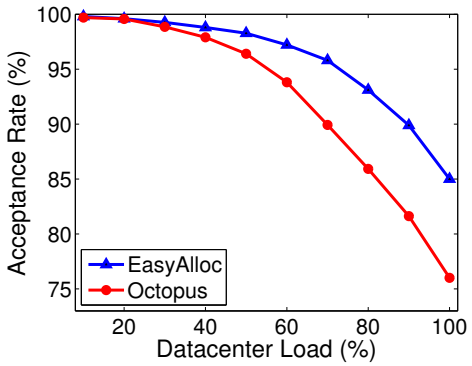


Fig. 6. Request acceptance rate with varying datacenter loads.

Most existing virtual network provisioning solutions leverage colocation techniques to reduce the cross-core network traffic in datacenters. These algorithms pay less attention to the elastic nature of virtual network requests. Therefore, we select a typical *elasticity-agnostic* solution, Octopus [4], as the counterpart for comparison with EasyAlloc. Octopus consists of a hose-based virtual cluster (VC) model that abstracts the application requirement as shown in Figure 2, and a VM placement algorithm that greedily colocates the VMs of a new request with existing requests as illustrated in Figure 5(b). Since Octopus is not designed for elastic applications, the greedy algorithm is changed slightly to consider the dependency of on-demand VMs on existing always-on VMs when seeking for optimal placements.

We simulate altogether 5,000 application requests that arrives follows a Poisson process with a mean rate λ. If an application request cannot be accommodated upon its arrival, it is rejected immediately. By varying the value of λ, we can control the load on the datacenter in terms of VM occupancy denoted by $\frac{\lambda \bar{N} \bar{T}}{4000 \times 4}$, where $\bar{N}$ and $\bar{T}$ are the mean size and the mean execution interval of an application request.

B. Evaluation Results

Based on the above description of experiment settings, we evaluate and compare the two methods in several metrics.

Request Acceptance Rate. The primary goal of cloud providers is to allow as many requests as possible run concurrently in the datacenter. Figure 6 plots the ratio of the accepted requests for two methods with varying datacenter loads. Generally, the acceptance rate for both methods decreases with the increase of the load, because an increasing number of requests cannot be accommodated due to the shortage of bandwidth and VMs under a heavier load.

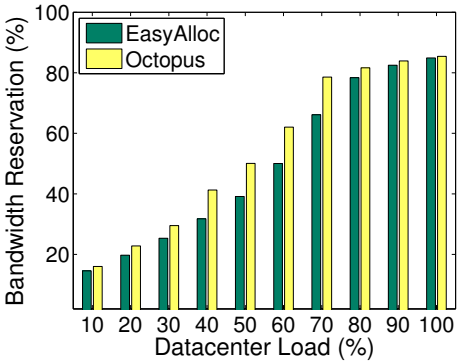


Fig. 7. Bandwidth reservation on core links with varying datacenter loads.

The acceptance rate for EasyAlloc is always higher than that for the elasticity-agnostic method, especially when the load is larger than 70%. At a relatively light load (*e.g.*, 10%-30%), the two methods have a similar performance that seldom rejects requests. As the load increase, the gap between two curves gradually enlarges, which indicates EasyAlloc utilizes the virtual resources more efficiently.

The reason lies in that the elasticity-aware method can potentially accommodate requests with less bandwidth consumption. First, the decoupling of the intra-tenant bandwidth requirement from the inter-tenant requirement mitigates the bandwidth over-provisioning in the virtual cluster abstraction, as illustrated in Figure 2. Second, Algorithm 1 considers the elasticity of requests and reserves rooms for the on-demand VMs, which results in better per-request locality and thus reduces the cross-network bandwidth consumption.

Bandwidth Reservation. Now we attempt to verify the performance gap is largely determined by the efficiency of bandwidth utilization. In an oversubscribed datacenter, the bandwidth of core links (*i.e.*, links connected to the core switch) is ease of becoming the bottleneck in accepting new requests. Therefore, we focus on exploring the bandwidth reserved on core links. Figure 7 exhibits the average link utilization for two methods with varying datacenter loads.

When the load is 80%, the bandwidth utilization for both two methods reaches a high level at around 80%. And then with the increase of load, the bandwidth reservation changes slightly, which indicates the bandwidth is almost fully utilized. Considering that the bandwidth reservation is calculated over the time scale, it cannot reach an exact 100% utilization. This observation confirms that when the load is heavy, bandwidth becomes the dominant factor for request rejection.

By comparing the two methods, the advantage of EasyAlloc over its counterpart, in terms of bandwidth reservation, is significant under the medium datacenter loads (*i.e.*, 40%-70%). The two methods have similar bandwidth reservation under heavy loads. However, when making an combination of the Figures 6 and 7, it is clearly that with a similar bandwidth reservation, the EasyAlloc is capable of accommodating more requests than Octopus.

Successful Extension Rate. In our simulation, we assume all accepted requests only scale out once at the predefined time during their lifetime. If the algorithm can find a feasible placement for on-demand VMs of an up-to-extend request, the extension is marked as successful; Otherwise, the request maintains the current status without further attempt to launch its on-demand VMs.

TABLE III. COMPARISON OF SUCCESSFUL EXTENSION RATES FOR TWO METHODS WITH DIFFERENT LOADS

Load	EasyAlloc	Octopus
20%	100%	100%
50%	92.4%	87.6%
80%	80.1%	63.3%

Table III lists the successful extension rates for two methods, where representative loads are selected for comparison, namely light (20%), medium (50%) and heavy (80%). There exists a gap between two methods under the medium load, and an even larger difference (16.8%) under the heavy load. That is because the bandwidth saving in EasyAlloc enables more requests to scale out as expected. The results confirm the effectiveness of the elasticity-aware abstraction model and the provisioning algorithm.

C. Summary

EasyAlloc aims to achieve efficiency and elasticity simultaneously, based on an elasticity-aware abstraction model and carefully considerations in finding appropriate VM placements. Through simulation results, we can conclude that the proposed EasyAlloc outperform the elasticity-agnostic method, in terms of request acceptance rate and the successful extension rate. Therefore, EasyAlloc bridges the elasticity requirement from tenants and the resource efficiency expectation from cloud providers.

VI. RELATED WORK

The virtual resource allocation in clouds has attracted many research attentions in recent years. Here we briefly summarize existing achievements on request abstraction, virtual network embedding and resource allocation.

Request abstraction. The most commonly used abstractions are the variants of the hose model which is originally designed for VPNs [9]. In a hose model abstraction, all VM are connected to a central virtual switch by a dedicated link that has a minimum bandwidth guarantee. The authors in [4] propose Octopus to expose tenants' virtual network requirements to cloud providers. Octopus consists of two types of abstractions, *i.e.*, *virtual cluster* and *virtual oversubscribed cluster*. The first assumes that all VMs are connected to a single non-oversubscribed virtual switch. It is designed for the all-to-all traffic pattern, which is the common case in MapReduce-like data-intensive applications. The second one is an enhanced hose model by organizing VMs into a two-tier cluster. Each cluster in the lower layer is an individual instance of the virtual cluster, and then connected with other clusters together. This model emulates the applications that prefer local communications.

Xie *et al.* [26] extend the hose model to captures the time-varying nature of bandwidth requirement of cloud applications.

Zhu *et al.* [29] make another extension to handle tenants with heterogeneous bandwidth demands. Ballani *et al.* [5] explicitly consider bandwidth guarantees for both inter- and inter-tenant communications, and propose a hierarchical hose model. The authors in [23] propose a dual-hose model to further decouple the guarantees for a tenant's inter-tenant traffic from those for its intra-tenant traffic, which enables cloud providers to accommodate more requests and reduces the completion time of tenants' jobs.

Since the hose-based abstraction cannot accurately express the requirement for applications composed of multi-tier components with complex traffic interactions, Lee *et al.* [18] present a network abstraction called Tenant Application Graph (TAG). TAG is derived based on application communication structure, and thus sacrifices the simplicity in accommodating virtual networks in the underlying network.

Virtual network embedding. Upon receiving a virtual network request that is expressed by a certain abstraction model, cloud providers have to decide the placement of such a virtual network. The mapping from virtual network onto the underlying network is usually referred to as virtual network embedding, which is firstly proposed in the field of conventional Internet service provider (ISP) networks [10]. In cloud datacenters, a virtual network takes the form of a virtual datacenter (VDC) that consists of VMs connected through virtual switches, routers and links with guaranteed bandwidth.

Zhang *et al.* [28] present a availability-aware VDC embedding framework that considers the reliability aspect of the embedding, in terms of heterogeneous hardware failure rates and dependencies among virtual components. In case of server failures, the live migration of VMs is unavoidable. Zhang *et al.* [27] theoretically analyze the required bandwidth value to satisfy the performance metrics of a live VM migration. Hao *et al.* [13] propose a generalized resource placement methodology that can work across different cloud architectures, with real-time request arrivals and departures.

Virtual resource allocation strategies. In datacenters, virtual computation resource (*i.e.*, VMs) are in better isolation than virtual bandwidth. Therefore, the resource allocation is usually divided into two steps [4, 18, 26], *i.e.*, fist finding an efficient VM placement according to operational goals, and then designing a runtime bandwidth allocation mechanism to enforce the application bandwidth requirements.

Two typical operational goals for the VM placement are maximizing the available bandwidth in core networks and minimizing the VM fragmentation in physical servers. Both of them aim to make an efficient resource utilization [8]. On the standpoint of tenants, Kuo *et al.* [15] propose an algorithm to minimize the maximum access latency among all pairs of a data node and its assigned computation node.

Since todays datacenters are often oversubscribed, many work-conserving bandwidth sharing mechanisms are proposed to provide fair sharing of networks for tenants, such as Seawall [24] and Netshare [17]. However, they do not provide deterministic bandwidth guarantees, thereby making it difficult for tenants to predict their worst-case performance. Guo [11] propose a novel distributed rate allocation algorithm based on the Logistic model under the control-theoretic framework, which provides bandwidth guarantees. Gatekeeper [21], EyeQ [14], FairCloud [19] and ElasticSwitch [20] simultaneously achieve minimum bandwidth guarantees and work-conservation. Particularly, Gatekeeper and EyeQ require congestion-free core networks.

In a work-conserving environment, the fairness of resource sharing among tenants is of great importance. Shen *et al.* [22] propose a new pricing model that sets different unit prices for reserved bandwidth, the bandwidth on congested links and on uncongested links, which can lead to a win-win situation for both tenants and cloud providers. Wang *et al.* [25] generalize the notation of dominant resource fairness from one server to multiple heterogenous servers, and design a multi-resource allocation mechanism for cloud computing systems.

The novelty of this paper. Based on the requirement characteristics of multi-tier applications in cloud networks [6, 12, 16], we design an elasticity-aware abstraction model, which makes the classification of always-on and on-demand VMs and decouples the bandwidth requirement for intra-tier communication from that for inter-tier communication. It is the first model that considers elastic nature of virtual networks. We also design an elasticity metric to quantify the interference of accepting a new request on the future scaling of existing requests. This metric enables us to jointly optimize the bandwidth and elasticity overhead when seeking for a best VM placement.

VII. Conclusion

In this paper, we propose an efficient and elastic virtual network provisioning solution called EasyAlloc, which is comprised of an *elasticity-aware* abstraction model and a virtual network provisioning algorithm. To accurately capture the tenant requirement and maintain the provisioning simplicity for providers, the elasticity-aware model enables two types of decoupling, *i.e.*, *always-on* VMs for normal load and *on-demand* VMs for dynamic scaling, and the bandwidth requirement of each VM for intra- and inter-tier communications. Then we formulate the virtual network provisioning as an overhead minimization problem, where the objective simultaneously considers the bandwidth and elasticity overhead. Due to the NP-completeness of this problem, we leverage two heuristics, slot reservation and tier iteration, to obtain an efficient algorithm. Extensive simulation results show that EasyAlloc outperform a typical elasticity-agnostic method, in terms of request acceptance rate. We leave more evaluations of EasyAlloc as future work.

Acknowledgment

This work has been supported in part by NSFC Project (61170292, 61472212, 61370192, 61432015), 973 Project of China (2012CB315803), 863 Project of China (2013AA013302, 2015AA010203, 2015AA01A705), UK EPSRC Project NIRVANA (EP/L026031/1), EU MARIE CURIE ACTIONS EVANS (PIRSES-GA-2013-610524), Multidisciplinary Fund of Tsinghua National Laboratory for Information Science and Technology, and Beijing Institute of Technology Research Fund Program for Young Scholars.

References

[1] Aliyun cloud. [Online]. http://www.aliyun.com/.

[2] Amazon web services. [Online]. http://aws.amazon.com/.
[3] Microsoft azure. [Online]. http://azure.microsoft.com/.
[4] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron. Towards predictable datacenter networks. *ACM SIGCOMM Computer Communication Review*, 41(4):242–253, 2011.
[5] H. Ballani, K. Jang, T. Karagiannis, C. Kim, D. Gunawardena, and G. O'Shea. Chatty tenants and the cloud network sharing problem. In *Proceedings of USENIX NSDI*, pages 171–184, 2013.
[6] T. Benson, A. Akella, and D. A. Maltz. Network traffic characteristics of data centers in the wild. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, IMC '10, pages 267–280, New York, NY, USA, 2010. ACM.
[7] P. Bodík, I. Menache, M. Chowdhury, P. Mani, D. A. Maltz, and I. Stoica. Surviving failures in bandwidth-constrained datacenters. In *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '12, pages 431–442, New York, NY, USA, 2012. ACM.
[8] L. Chen and H. Shen. Consolidating complementary vms with spatial/temporal-awareness in cloud datacenters. In *INFOCOM, 2014 Proceedings IEEE*, pages 1033–1041, April 2014.
[9] N. G. Duffield, P. Goyal, A. Greenberg, P. Mishra, K. K. Ramakrishnan, and J. E. van der Merive. A flexible model for resource management in virtual private networks. *ACM SIGCOMM Computer Communication Review*, 29(4):95–108, 1999.
[10] L. Gong, Y. Wen, Z. Zhu, and T. Lee. Toward profit-seeking virtual network embedding algorithm via global resource capacity. In *INFOCOM, 2014 Proceedings IEEE*, pages 1–9, April 2014.
[11] J. Guo, F. Liu, X. Huang, J. Lui, M. Hu, Q. Gao, and H. Jin. On efficient bandwidth allocation for traffic variability in datacenters. In *INFOCOM, 2014 Proceedings IEEE*, pages 1572–1580, April 2014.
[12] M. Hajjat, X. Sun, Y.-W. E. Sung, D. Maltz, S. Rao, K. Sripanidkulchai, and M. Tawarmalani. Cloudward bound: Planning for beneficial migration of enterprise applications to the cloud. *SIGCOMM Comput. Commun. Rev.*, 40(4):243–254, Aug. 2010.
[13] F. Hao, M. Kodialam, T. Lakshman, and S. Mukherjee. Online allocation of virtual machines in a distributed cloud. In *INFOCOM, 2014 Proceedings IEEE*, pages 10–18, April 2014.
[14] V. Jeyakumar, M. Alizadeh, D. Mazieres, B. Prabhakar, C. Kim, and W. Azure. Eyeq: Practical network performance isolation for the multi-tenant cloud. In *Proceedings of USENIX HotCloud*, 2012.
[15] J.-J. Kuo, H.-H. Yang, and M.-J. Tsai. Optimal approximation algorithm of virtual machine placement for data latency minimization in cloud systems. In *INFOCOM, 2014 Proceedings IEEE*, pages 1303–1311, April 2014.
[16] K. LaCurts, S. Deng, A. Goyal, and H. Balakrishnan. Choreo: Network-aware task placement for cloud applications. In *Proceedings of the 2013 Conference on Internet Measurement Conference*, IMC '13, pages 191–204, New York, NY, USA, 2013. ACM.
[17] T. Lam and G. Varghese. Netshare: Virtualizing bandwidth within the cloud. Technical report, UCSD, 2009.
[18] J. Lee, Y. Turner, M. Lee, L. Popa, S. Banerjee, J.-M. Kang, and P. Sharma. Application-driven bandwidth guarantees in datacenters. *SIGCOMM Comput. Commun. Rev.*, 44(4):467–478, Aug. 2014.
[19] L. Popa, G. Kumar, M. Chowdhury, A. Krishnamurthy, S. Ratnasamy, and I. Stoica. Faircloud: sharing the network in cloud computing. In *Proceedings of ACM SIGCOMM*, pages 187–198, 2012.
[20] L. Popa, P. Yalagandula, S. Banerjee, J. C. Mogul, Y. Turner, and J. R. Santos. Elasticswitch: practical work-conserving bandwidth guarantees for cloud computing. In *Proceedings of ACM SIGCOMM*, pages 351–362, 2013.
[21] H. Rodrigues, J. R. Santos, Y. Turner, P. Soares, and D. Guedes. Gatekeeper: Supporting bandwidth guarantees for multi-tenant datacenter networks. In *Proceedings of USENIX WIOV*, 2011.
[22] H. Shen and Z. Li. New bandwidth sharing and pricing policies to achieve a win-win situation for cloud provider and tenants. In *INFOCOM, 2014 Proceedings IEEE*, pages 835–843, April 2014.
[23] M. Shen, L. Gao, K. Xu, and L. Zhu. Achieving bandwidth guarantees in multi-tenant cloud networks using a dual-hose model. In *Proceedings of IEEE IPCCC*, pages 1–8, 2014.
[24] A. Shieh, S. Kandula, A. Greenberg, C. Kim, and B. Saha. Sharing the data center network. In *Proceedings of USENIX NSDI*, pages 23–23, 2011.
[25] W. Wang, B. Li, and B. Liang. Dominant resource fairness in cloud computing systems with heterogeneous servers. In *INFOCOM, 2014 Proceedings IEEE*, pages 583–591, April 2014.
[26] D. Xie, N. Ding, Y. C. Hu, and R. Kompella. The only constant is change: incorporating time-varying network reservations in data centers. *ACM SIGCOMM Computer Communication Review*, 42(4):199–210, 2012.
[27] J. Zhang, F. Ren, and C. Lin. Delay guaranteed live migration of virtual machines. In *INFOCOM, 2014 Proceedings IEEE*, pages 574–582, April 2014.
[28] Q. Zhang, M. Zhani, M. Jabri, and R. Boutaba. Venice: Reliable virtual data center embedding in clouds. In *INFOCOM, 2014 Proceedings IEEE*, pages 289–297, April 2014.
[29] J. Zhu, D. Li, J. Wu, H. Liu, Y. Zhang, and J. Zhang. Towards bandwidth guarantee in multi-tenancy cloud computing networks. In *IEEE International Conference on Network Protocols (ICNP)*, pages 1–10, 2012.

APPENDIX

Now we focus on analyzing different placements for the always-on VMs in Tenant P as listed in Table I. The topology in use as shown in Figure 5(a) has two types of switches, *i.e.*, ToR and aggregation switches. In general, a placement should use as less high-level links as possible. This implies that it is better to accommodate the always-on VMs within a single rack. Without loss of generality, we select servers 1 and 2 as the host servers.

TABLE IV. FEASIBLE PLACEMENTS AND CORRESPONDING BANDWIDTH REQUIREMENTS FOR THE REQUEST IN FIGURE 5

Index	Server 1	Server 2	BW Requirement (Mbps)
1	PR, 2BL	2DB	$\min(750, 500) = 500$
2	2BL, 2DB	PR, BL	$\min(900, 350) = 350$
3	3BL, DB	PR, DB	$\min(950, 500) = 500$
4	PR, BL, 2DB	2BL	$\min(850, 400) = 400$
5	3BL	PR, 2BL	$\min(600, 650) = 600$
6	PR, 2BL	BL, 2DB	$\min(550, 700) = 550$
7	PR, BL, DB	2BL, DB	$\min(700, 750) = 700$

Due to the homogeneity of VMs in the same tier and servers in the same rack, we can enumerate all possible placements in Table IV, where server 1 and server 2 are interchangeable. The first 4 items (*i.e.*, placements 1-4) accommodate 4 VMs and 2 VMs in each server, respectively. And the rest 3 items (*i.e.*, placements 5-7) place 3 VMs in each server. The placement with the least bandwidth requirement is the 2nd item, which is the same as the one in Figure 5(a).

The multi-tier nature of applications and the hierarchical datacenter topology makes the number of feasible placements increase superlinearly with the number of VMs. Based on carefully analysis on Table IV, we can get some tricks that helps us to efficiently obtain the desired placement:

1) accommodating as many VMs in a single server as possible for the same request (*e.g.*, a 4+2 placement is commonly better than a 3+3 placement), and

2) colocating VMs with more bandwidth requirements together (*e.g.*, two DB instances should be placed together).