Resource Optimization for Survivable Embedding of Virtual Clusters in Cloud Data Centers

Biyu Zhou*†, Jie Wu‡, Fa Zhang*, and Zhiyong Liu*
*State Key Laboratory of Computer Architecture, ICT, CAS
†University of Chinese Academy of Sciences, Beijing, China
‡ Center for Networked Computing, Temple University, USA
{zhoubiyu, zhangfa, zyliu}@ict.ac.cn, jiewu@temple.edu

***Abstract*—With the popularity of cloud computing, optimizing cloud resource consumption while providing predictable cloud service has become one of the focuses of research in recent years. In order to ensure a predictable performance, the requests from tenants are abstracted as Virtual Clusters, which not only specify the computing demands, but also establish the communication requirements among virtual machines. While much work has been done on virtual cluster embedding under a variety of goals, very few people have studied this issue in consideration of service survivability, which also plays a vital role in ensuring the performance in cloud data centers. In this paper, we study the resource optimization for survivable embedding of virtual clusters and aim to minimize the consumption of cloud resources in terms of server and bandwidth, while ensuring that both the resource constraints and the survivability constraints are not violated. We formally define this problem and analyze its complexity, and design efficient algorithms to solve the problem. Comprehensive experimental results verify that the overall resource consumption can be significantly reduced by applying our proposals.**

***Index Terms*—Virtual Cluster, resource optimization, survivability, bandwidth guarantee, cloud data center.**

I. Introduction

Virtualization has already become a widely adopted technology for sharing resources in cloud computing [1, 2]. By multiplexing physical resources such as server and network bandwidth through properly scheduling the requests from different tenants, the resource usage can be reduced while the service level agreement (SLA) of each tenant is satisfied. A simple interface is maintained between tenants and cloud providers: cloud tenants submit their resource requests in the form of virtual machines (VMs) to the cloud provider. The cloud provider then decides the allocation of the physical resources to each of the requested virtual machines. Simple as it is, the shared nature of the network in multi-tenant cloud data centers determines that the network performance for different tenants can vary significantly. This has stimulated the recent research on providing service abstractions that guarantee the bandwidth in cloud resource sharing.

Zhiyong Liu is the corresponding author. This work is supported partially by the National Natural Science Foundation of China (NSFC) Project 61520106005, 61521092, National Key Research and Development Program of China 2017YFB1010000, NSF grants CNS 1629746, CNS 1564128, CNS 1449860, CNS 1461932, CNS 1460971, CNS 1439672, and CNS 1301774.

The *Virtual Cluster* (VC) abstraction introduced in [3] is one of the most popular service abstractions due to its simplicity and efficiency [3–9]. The VC request allows each tenant to specify a topology rather than only the number of VMs. The topology specifies both the number of VMs required and the minimum aggregate bandwidth capacity requirements between any two VMs. The VC abstraction adopts the hose model [3] to specify the bandwidth requirements between each VM pair. When a VC request is admitted to access, the cloud provider will place all the VMs of the tenant on physical servers that have enough empty VM slots and reserve sufficient bandwidth in the network to guarantee the bandwidth requirements according to the hose model. In this way, the network performance of the tenant is guaranteed. The resource allocation process for VC requests is also named Virtual Cluster Embedding (VCE).

In addition to bandwidth, another equally important performance attribute that should be ensured when providing cloud service to tenants is the service availability in the event of a burst physical server failure. Unfortunately, many existing proposals have failed to take ensuring service availability as a goal when dealing with resource optimization for VCE. In a large-scale cloud data center, physical server failure can happen frequently [10]. When such a failure happens, all VMs in the failed server will stop working. Even a single element failure can cause severe revenue losses for those tenants [11]. According to Cerin's study [12], software failures have led to up to tens of millions of dollars being lost in current mainstream cloud systems including Amazon, Windows Azure, Google App, etc. In order to reduce the losses caused by these failures, a lot of survivability mechanisms that exploit the redundancy of the resources in cloud data centers are proposed [13] to enable a quick service recovery when failures happen. One survivability mechanism [10] that has been widely adopted in cloud data centers works in a proactive manner. This mechanism is based on the observation that some techniques, such as Hardware Predicted Failure Analysis alerts, can provide an advanced warning of expected hardware failure [14]. Such an alert can trigger the system to move the VMs on a server that is about to fail to backup resources that have been reserved prior to the actual failure, at the cost of an underutilized resource when no failure happens. In this way, the continuation of cloud service is guaranteed.

In this paper, we study the problem of resource optimization for VCE in cloud data centers under the proactive survivability constraint. Particularly, we consider VCE under 1-survivability constraint, which means that a tenant service can recover from an arbitrary single-server failure [10]. However, our solution can be easily extended to a k-survivability case. We consider the model that each physical server involves several uniform VM slots, each representing one resource unit that has a certain amount of CPU, memory, disks, etc. A slot can be used to place an arbitrary VM. Given a fixed data center network topology (typically a tree-like network architecture [15]) and a set of VC requests, our aim is to find a feasible embedding plan for all the VC requests in the cloud data center network that minimizes both the server resource consumption (the number of slots that should be reserved) and the bandwidth resource consumption (the total bandwidth capacity that should be reserved) without violating the 1-survivability constraint.

However, there is a conflict between simultaneously minimizing the bandwidth resource consumption and server resource consumption under the given survivability constraint. Intuitively, we can increase the the fault tolerance by spreading VMs of a tenant across many servers, thus reducing the server resource consumption (by reducing the number of backup VM slots). However, this allocation requires more bandwidth resource in the network (as illustrated in Figure 5(a)). On the other hand, to reduce the bandwidth resource consumption of a tenant, we can assemble all the VMs in a subtree of the network as much as possible. This allocation, however, reduces the fault tolerance of a tenant, and thus increases the server resource consumption (as illustrated in Figure 5(b)).

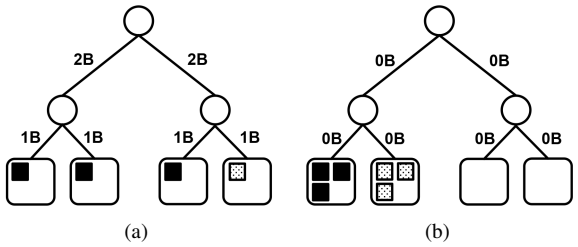


Fig. 1. The survivable embedding of the VC request $< 3, 1B >$ in a simple network topology with three switches and four physical servers illustrates the trade-off between server usage (spread VMs across servers, 5(a)) and bandwidth usage (pack VMs together, 5(b)). A black box indicates a slot has been allocated to a primary VM. A dotted box indicates a slot has been allocated to a backup VM. The value on each link indicates the number of bandwidth units that should be reserved in the plan. It can been seen that (a) needs the least slots (4) at the cost of 8 units of bandwidth; (b) consumes no bandwidth at the cost of 6 slots.

One challenge in this scheduling problem is achieving a good balance between server resource consumption and bandwidth resource consumption. To tackle the resource optimization problem, we first give a detailed description of the service model adopted in this paper. Then, a formal definition of the problem that combines two objectives (i.e., server resource optimization and bandwidth resource optimization) into a unified objective is presented. We then apply optimization techniques to compute an efficient VCE plan in terms of minimizing the unified objective for a single VC request. Finally, we design an effective heuristic scheduling algorithm that packs multiple VC requests to cloud data center networks to minimize the unified objective.

The main contributions are summarized as follows.

- This paper models and defines the VC request embedding problem under the survivability constraint. It formulates the resource optimization problem in terms of both server resource and bandwidth resource into an bi-objective optimization problem. Furthermore, we analyze that the problem is NP-hard in general.
- We design a novel scheduling algorithm with the following features: i) it exploits the trade-off between server resource usage and bandwidth resource usage to reduce the unified objective; ii) it provides a simple method for approximating the maximum load on the subtree rooted at an arbitrary node in the network topology.
- We conduct comprehensive simulations to evaluate the performance of our algorithms, and experimental results verify that our proposal can achieve a significant reduction of resource consumption.

The remainder of this paper is organized as follows. Section II provides the background and related work. Section III presents the models and problem statement. Section IV presents the optimization framework and algorithm details. Section V shows the experimental results and Section VI concludes the paper.

II. Background and Related work

A. Virtual Cluster Embedding

Virtual cluster (VC) is one of the most popular service abstractions for a multi-tenant cloud computing environment. It offers a bandwidth guarantee among all the VMs [3]. The VC request, $r :< N, B >$, allows tenants to specify a topology rather than only the number of VMs. The topology specifies both the number of VMs required (N) and the minimum aggregate bandwidth capacity requirement (B) between any two VMs (as illustrated in Figure 2). The VC abstraction adopts the hose model [3] to specify the bandwidth requirement between each VM pair. For each switch, v, in the network, assume that the number of VMs within the subtree rooted at v is N_v, then, the bandwidth reserved on outbound link of switch v for VC request r should be no less than $min\{n_v, N - n_v\} \cdot B$. The VC abstraction is simple and flexible, and thus has been widely used in both academia and industry.

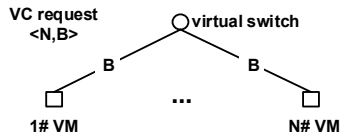


Fig. 2. An example of virtual cluster provisioning.

When a VC request arrives, the cloud provider then places all the VMs on physical servers that have enough VM slots and reserves sufficient bandwidth in the network to guarantee

the bandwidth requirement according to the hose model. In this way, the network performance of the tenants is guaranteed. The resource allocation process for VC requests is also named Virtual Cluster Embedding (VCE). A feasible VCE indicates that the resource capacity constraint is not violated.

B. Survivable VCE

Survivability plays a vital role in cloud service provision. The topic of providing survivable virtual cloud services has been studied extensively, focusing on either survivable VM hosting or virtual infrastructure hosting. However, ensuring a high survivability in VCE efficiently remains a problem. The only relevant work on this topic is conducted by Yu *et al.* in [16], where the authors study the problem of providing a survivable VCE with minimal resource consumption as we do. However, they focus on a simple case where only the server resource in terms of the VM slot is considered for a single VC request scenario, while our work simultaneously optimizes the consumption of both the server and bandwidth resources for multiple VC requests scenario.

This work considers a proactive 1-survivability mechanism. When a VC request is accepted, the cloud provider will not only place the primary VMs of the VC request in the network, but will also reserve enough backup resources for this VC request to ensure that there are enough backup VM slots and bandwidth resources in the event of an arbitrary single physical server failure under the hose communication model. We consider 1-survivability due to the fact that, while single-server failures are frequent in cloud data centers, simultaneous multi-server failures are rare [17]. Once an advanced warning of expected server failure is generated, the system moves all the VMs on the failing server to backup slots for sustained service. To avoid confusion, the VMs requested by tenants are called *primary VM*s and the VM slots reserved for backup use are called *backup VM*s. Specifically, a survivable VCE is defined as an allocation of server and bandwidth resources to the VC request such that during any single-server failure, a feasible VCE exists in the resilient network.

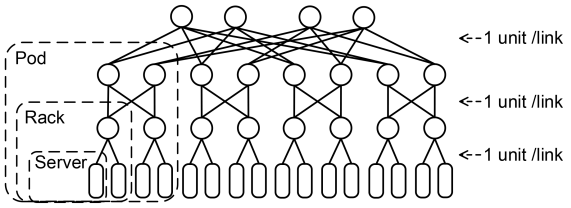
Figure 1 depicts two survivable VCE plans for the VC request $r :< 3, 1B >$. In Figure 1(a), compared to a VCE plan that uses only 3 VM slots and $5B$ units of bandwidth, the survivable VCE plan consumes 4 VM slots and $8B$ units of bandwidth. In Figure 1(b), compared to a VCE plan that consumes only 3 VM slots and 0 units of bandwidth, the survivable VCE plan consumes 6 VM slots and 0 units of bandwidth. With these redundant resources, the service of r can be recovered from any single-server failure.

III. Models and problem statement

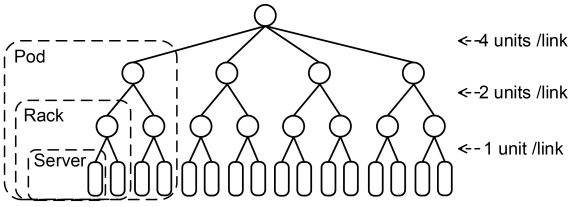
In this paper, we study the resource optimization problem when providing embedding plans for virtual clusters in cloud data centers under the survivability constraint. We first describe the scenarios and introduce some notations. Then, we provide a formal definition for the survivable VC embedding problem and formulate this problem into an optimization problem. Finally, we analyze the complexity of the problem.

A. Network Service Models

The data center is portrayed as a large-scale computing system made up with thousands of servers connected by a specifically designed network topology. Let $\mathcal{H}$ denote the set of all physical servers, the total number of servers in the network is $M = |\mathcal{H}|$. The data center network architecture studied in this paper can be abstracted as a three-level single-rooted tree, $T = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of switches and $\mathcal{E}$ is the set of network links. It is composed of three layers of network switches, namely access, aggregate, and core layers. We assume that all servers in the network and all links connecting the servers are identical. The bandwidth of the uplink of each switch is the summation of the bandwidth of the downlinks. This is reasonable since the networks are generally built with identical commodity switches and uniform servers in modern cloud data centers. We consider this type of network architecture because many of the multi-rooted tree based network architectures used today can be equivalently abstracted as a single-rooted tree by aggregating multiple equal paths into a single one. Figure 3 shows an example of the equivalent abstraction of a 4-ary FatTree topology [18].



(a) Original 4-ary FatTree topology.



(b) Equivalent abstraction of a 4-ary FatTree topology.

Fig. 3. A 4-ary FatTree topology. In (a), the bandwidth of all links is 1 unit. In (b), the bandwidth of the links associated with the root switch is 4 units; The bandwidth of the links connecting with servers is 1 unit; The bandwidth of the remaining links is 2 units.

We use *slot* to represent one resource unit that has a certain amount of CPU, memory, disks, etc. Each slot can host one virtual machine (VM). For each physical server, $h_i \in \mathcal{H}$, we define c as the total number of VM slots on h_i. For each uplink out from a physical server, the bandwidth capacity of this link is denoted by b. Tenants use VC abstraction to specify their computing demands. We consider a set of m independent VC requests, $\mathcal{R} = \{r_i | 1 \leq i \leq m\}$, as the input demands. This hypothesis makes sense because a typical strategy for handling online job scheduling is to divide time into equal time slots and schedule newly arrived tasks in a time slot as a batch. We attempt to find survivable VCE plans for all requests in $\mathcal{R}$.

B. Resource Optimization

Our aim is to determine a feasible and survivable embedding plan for a given set of VC requests (i.e., $\mathcal{R} = \{r_i | 1 \leq i \leq m\}$) that minimizes the total resource consumption. The problem is important for cloud data centers, as it can help maximize the data center's future ability to accommodate tenants' requests. We describe an assignment of VMs by the variables $\{x_{k,i}\}$. The variable $x_{k,i}$ indicates the number of VMs of the VC request r_k (including both primary VMs and backup VMs) is placed on server h_i. The types of resources to be optimized in this paper are listed as follows.

Server resource. We use the average number of VM slots that are reserved and occupied on all physical servers as the overall measure of the server resource consumption in the network. Let **SC** denote the average server resource consumption. Correspondingly, SC involves both the slots occupied by primary VMs from tenants and the backup VMs reserved to ensure service survivability. Then, SC can be formally defined as

$$SC(X) = \frac{1}{m} \sum_{k=1}^{m} \sum_{i=1}^{M} x_{k,i}$$

Bandwidth resource. We use average bandwidth units that are reserved on all links as the overall measure of the bandwidth resource consumption in the network. Let **BC** denote the average network resource consumption. Correspondingly, BC includes the bandwidth reserved to guarantee the communication requirements among all the primary VMs and backup VMs. Let $F(k)$ denote the total number of both primary and backup VMs of VC request r_k. We use $I(v, k)$ to denote the number of VMs (both primary and backup VMs) located in the subtree rooted at v. Then, BC can be formally defined as

$$BC(X) = \frac{1}{m} \sum_{k=1}^{m} \sum_{v \in \mathcal{V} \cup \mathcal{H}} \min\{I(v,k), F(k) - I(v,k)\} \cdot B_k$$

We combine the above two objectives into a single objective using a scalarization function. Specifically, the total resource consumption considered in this paper is defined as the sum of the bandwidth resource consumption and the server resource consumption. Let **TC** denote the total resource consumption, it can be formally defined as

$$TC = (1 - \alpha) \cdot BC + \alpha \cdot SC \tag{1}$$

where $\alpha \in [0, 1]$ is a tunable positive weight. This scalarization function accommodates a flexible trade-off between BC and SC, expressed through the choice of α. In extreme cases (i.e., α equals 0 or 1), the above objective degenerates into a single optimization.

Given the network topology with an initial assignment of resources to services and a set of tenant VC demands as the input, the problem we study in this paper is finding the survivable embedding of all these VC requests that uses the minimum total resources in cloud data centers. The unified total resource (TC) considered here indicates both the bandwidth and server resources. Formally, the problem is defined as the following optimization problem.

Definition 1: Given network $T = (\mathcal{V}, \mathcal{E})$ and a set of tenant VC requests, $R = \{r_i | 1 \leq i \leq m\}$, the goal of the *Resource Optimization for Survivable Virtual Cluster Embedding* (RO-SVCE) is to find an embedding plan for all the requests in the network such that TC is minimized while the constraints on either survivability or resource capacity are not violated.

Not surprisingly, the general RO-SVCE problem is NP-hard. [3] This can be easily proven by a reduction from the bin packing problem [19]. The key challenge of RO-SVCE is to deal with the trade-off between server resource optimization and bandwidth resource optimization. As illustrated in Figure 1, a solution that uses fewer slot resources will result in more bandwidth consumption (as in Figure 5(a)), while a solution that uses fewer bandwidth resources will result in more VM slot consumption. In the following section, we will provide efficient algorithms to deal with this issue.

IV. Resource optimization for virtual clusters under survivability constraint

In view of the hardness of the RO-SVCE problem, we propose efficient heuristic algorithms to tackle this issue in this section.

A. Solution for single VC requests

We start by designing an efficient algorithm that solves this problem for a single VC request (i.e., $r :< N, B >$). Before going into details, we first present two observations. Let N_b denote the minimum number of backup slots needed to guarantee the survivability constraint. Let q_i denote the number of VC request r's primary VMs allocated to the physical server, $h_i \in \mathcal{H}$.

Observation 1: N_b is determined by the maximum number of primary VMs in a single physical server, i.e., $N_b = \max_{h_i \in \mathcal{H}}\{q_i\}$. The total server resource consumption can be formulated as $SC = N + N_b$. Thus, minimizing the server resource usage is equivalent to minimizing the value of N_b. This observation motivates us to control the server resource consumption by adjusting the value of N_b, which can be easily adjusted in a linear manner.

Observation 2: A minimum-bandwidth survivable VCE should allocate all VMs within a subtree as much as possible. This observation motivates us to control the bandwidth resource consumption by packing VMs together.

Based on the two observations, we design an algorithm (S_Algo) to tackle the RO-SVCE problem for a single VC request. The intuition is that the consumption of the server resource is linearly related to the value of N_b, while the consumption of the bandwidth resource has a monotonic relationship with the level of aggregation of the VMs. Given the value of N_b, an embedding plan that consumes minimum bandwidth resource can be find by searching the smallest subtree that can accommodate the VC request. By adjusting the value of N_b, we can achieve a balance between the consumption of server and network resources.

S_Algo is illustrated in Algorithm 1. The algorithm works in a greedy manner. Initially, the number of backup VMs

Algorithm 1 S_Algo(r).

Require: Data center topology T.
Ensure: Allocation for request VC $r :< N, B >$.

$TC^* \leftarrow \infty$, $v^* \leftarrow v_0$, $l^* \leftarrow \infty$.
for $N_b \in \{1, \cdots, N\}$ **do**
  for $h_i \in \mathcal{H}$ **do**
    $c_i \leftarrow \min\{c_i, N_b\}$.
  /*Level 0 stands for the edge, level 1 represents the aggregation, level 2 means the core*/
  for $l \in \{0, 1, 2\}$ **do**
    Cal_Capacity(l).
    for each switch v in Level l **do**
      /*n_v is the capacity of node v*/
      if $n_v \geq (N + N_b)$ **then**
        Cal_Embeding(v).
        Compute the total resource usage TC.
        if $TC > TC^*$ **then**
          Continue.
        $TC^* \leftarrow TC$, $v^* \leftarrow v$.
if $TC^* = \infty$ **then**
  No feasible solution exists.
else
  Cal_Embeding(v, l); Return TC^*.

Algorithm 2 Cal_Capacity(l).

Require: Data center topology, T.
Ensure: The capacity of the node, v.

for each switch v in Level l **do**
  /*$\mathcal{S}_v$ is the set of node in level $l-1$ that connects to switch v, i.e., the set of all children of v*/
  $\mathcal{O} \leftarrow \emptyset$.
  for $v_i \in \mathcal{S}_v$ **do**
    if $v_i \in \mathcal{H}$ **then**
      $n_i \leftarrow c_i$.
    if $n_i \geq \lfloor \frac{b_i}{B} \rfloor$ **then**
      $\mathcal{O} \leftarrow \mathcal{O} \cup v_i$.
    $\Delta_i \leftarrow \min\{n_i, \lfloor \frac{b_i}{B} \rfloor\}$.
  $i^* \leftarrow argmax_{v_i \in \mathcal{O}} \lfloor \frac{b_i}{B} \rfloor$.
  if $(\sum_{v_i \in \mathcal{S}_v} \Delta_i - \Delta^{i^*}) \leq \lfloor \frac{b_{i^*}}{B} \rfloor$ **then**
    $\Delta_{i^*} \leftarrow n_{i^*}$.
  $n_v \leftarrow \sum_{v_i \in \mathcal{S}_v} \Delta_i$.

(i.e., N_b) is set to 1. Given the value of N_b, the number of available slots in each server is upper bounded by N_b accordingly due to the survivability constraint. Then, we use a procedure, Cal_Capacity(l) (as illustrated in Algorithm 2), to compute the number of available VMs that can be located in the subtree rooted at each switch, v, in the network from the bottom to the top (also referred to as the capacity of the switch, i.e., n_v). The Cal_Capacity(l) procedure works in a bottom-to-top manner. Once a switch, v, whose subtree has enough capacity to hold the VC request, is found (without loss of generality, let's say that v is in level l), a procedure, Cal_Embedding(v, l) (as illustrated in Algorithm 3), is called to find an embedding plan for this request in the subtree. Then, we compute the total resource consumption under this new embedding plan. If the new embedding plan reduces the total resource consumption, this new embedding plan is recorded as a potential embedding plan. By increasing the value of N_b, we can achieve a point where the total resource consumption cannot be reduced by increasing the number of backup slots. At this point, the algorithm terminates. The final embedding plan is the output of the algorithm. It can be verified that the embedding plan obtained by S_Algo achieves a good balance between the server resource usage and the bandwidth resource usage.

The number of available VMs that can be located in the subtree rooted at each switch in level l in the network cannot be calculated using an easy expression. In fact, the number of available VMs that can be located in the subtree rooted at switch v when it is chosen as a root in the embedding plan (denoted by n_v) is not the same as when it is not a root (denoted by Δ_v). We use a simple procedure, Cal_Capacity(l) (as illustrated in Algorithm 2), to compute approximate values for both n_v and Δ_v. The algorithm works in a bottom-to-top manner. Let $\mathcal{S}_v$ denote the set of nodes in level $l-1$ that connect to switch v (i.e., the set of all children of v). We first show the calculation of the value of Δ_i for node v_i. This value is computed by its parent node. The computation is as follows. If the child is a physical server, the capacity of this child is initialized to the number of its available slots. For each switch, v, the number of available VMs that can be located within the subtree of its child, $v_i \in \mathcal{S}_v$, is limited by the capacity and the uplink of v_i. Thus, its capacity can be computed by $\Delta_i = \min\{n_i, \lfloor \frac{b_i}{B} \rfloor\}$.

Now, we show the computation of the value of n_v for node v. Intuitively, the value of n_v should be the summation of the values of Δ_i of all its child nodes. However, due to the specific features of the hose communication model adopted by the VC abstraction, the actual value of n_v may be slightly larger than the sum. Consider a simple case in Figure 4, assume that the VC request is $< 12, 1B >$. For the left child, v_L, we have $\Delta_L = \min\{4, 5\} = 4$. For the right child, v_R, we have $\Delta_R = \min\{6, 4\} = 4$. Thus, we have $n_R = 4 + 4 = 8$. However, according to the hose communication model, it is easy to know $n_R = 4 + 6 = 10$. To tackle this issue, the Cal_Capacity(l) algorithm first finds out all the children that satisfy $n_i \geq \lfloor \frac{b_i}{B} \rfloor$ (denoted by set $\mathcal{O}$). Then, it finds the child node with the maximum value of $\lfloor \frac{b_i}{B} \rfloor$ in $\mathcal{O}$ (this child is referred to as v_{i^*}). After that, it determines whether the child v_i^* is bandwidth-constrained or server-constrained. If it is bandwidth-constrained (i.e., $(\sum_{v_i \in \mathcal{S}_v} \Delta_i - \Delta^{i^*}) \geq \lfloor \frac{b_{i^*}}{B} \rfloor$), the maximum number of VMs that can be located in the subtree rooted at v_i^* is $\min\{n_i, \lfloor \frac{b_i}{B} \rfloor\}$ (i.e., $\Delta_{i^*} = \min\{n_i, \lfloor \frac{b_i}{B} \rfloor\}$). Otherwise, the maximum number of VMs that can be located in the subtree rooted at v_i^* is n_i^* (i.e., $\Delta_{i^*} = n_i^*$). By summing up all the values of the maximum capacities of all the child nodes, the algorithm outputs the capacity of the root node v (i.e., $n_v = \sum_{v_i \in \mathcal{S}_v} \Delta_i$). It can be easy to verify that the

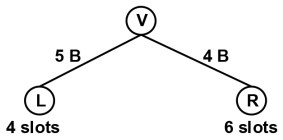


Fig. 4. A simple example of a three node topology.

computed values of n_v and Δ_i are equal or a slightly smaller than the actual values, and thus are feasible.

Once a switch, v, whose subtree has enough capacity to hold the VC request is found, the Cal_Embedding(v, l) procedure will be called to calculate an embedding plan for this request in the subtree. The procedure works in a up-to-bottom manner. Let $\mathcal{Q}$ denote the set of nodes that are waiting to be processed (i.e., candidate set). The candidate set is initialized to $\{v\}$. Let δ_v denote the number of VMs waiting to be assigned at node v. For each node, u, in the candidate set, it first sorts all its children (i.e., $v_i \in \mathcal{S}_u$) in descending order of their capacities (i.e., Δ_i). Then, it fills each child, v_i, with VMs in sequence until the δ_u VMs have been allocated. The number of VMs assigned to each node v_i is denoted by δ_i. After the end of a round, the candidate set is updated. The process is executed recursively in each level. When all the VMs are allocated to physical servers, the algorithm terminates.

Algorithm 3 Cal_Embedding(v, l).

Require: Data center topology, T.
Ensure: Allocation for VC request, r.
1: $\mathcal{Q} \leftarrow \{v\}$, $\delta_v \leftarrow N$.
2: **while** $l \geq 0$ **do**
3: $\quad \mathcal{Q}' = \emptyset$.
4: $\quad$ **for** $u \in \mathcal{Q}$ **do**
5: $\quad\quad$ Sort all children (i.e., $v_j \in \mathcal{S}_u$) in descending order of their capacities (i.e., Δ_i).
6: $\quad\quad$ Assign each child v_i with δ_i VMs to fill up its capacity Δ_i in sequence until the δ_u VMs have been allocated.
7: $\quad\quad$ $\mathcal{Q}' \leftarrow \mathcal{Q}' \cup \mathcal{S}_u$,
8: $\quad$ $\mathcal{Q} \leftarrow \mathcal{Q}'$, $l \leftarrow l - 1$.

B. Solution for multiple VC requests

Based on the solution for a single VC request, we can now present an algorithm (M_Algo) to solve the RO-SVCE problem for the multi-tenant case. M_Algo is described as follows. Let w_i ($w_i = N_i * B_i$) denote the weight of the request, r_i. Firstly, we sort all the VC requests in descending order of their weights. The intuition is that the embedding of the VC request with a larger weight has a greater impact on the final performance. Thus, it should be proceeded earlier. After the order has been determined, all the VC requests are embedded one-by-one according to S_Algo. The algorithm is also illustrated in Algorithm 4.

C. Discussions

1) Acceleration: By searching the value space of N_b, the algorithm will compare up to N minimum bandwidth consumption embedding plans under each value of N_b in order to find the optimal plan that achieves a good balance between bandwidth resource usage and server resource usage. Due to the monotonic relationship between the two metrices, a binary search method can be applied to accelerate the process of the algorithm (i.e., the number of comparisons that should be conducted can be reduced to up to $log(N)$ times).

Algorithm 4 M_Algo$(\mathcal{R})$.

Require: Data center topology, T.
Ensure: Allocation for VC request set, $\mathcal{R}$.
1: **for** $r_i \in \mathcal{R}$ **do**
2: $\quad w_i \leftarrow N_i * B_i$.
3: Sort all requests in descending order of their weights.
4: **for** $r_i \in \mathcal{R}$ **do**
5: $\quad$ S_Algo(r_i).

2) Extension: Although we use the FatTree topology in this paper to facilitate the description of our solution, S_Algo can be extended to other typical data center network topologies that can be abstracted as a single-rooted tree following the method illustrated in Figure 3. In addition, although this paper deals with the 1-survivability case, the proposal can be easily extended to the k-survivability case by setting the number of backup VM slots to the summation of the first k-minimum number of primary VM slots in the servers.

V. Evaluation

The effectiveness and efficiency of our proposed algorithms are evaluated through extensive simulations. In this section, we will give a detailed summary of our simulation findings.

A. Evaluation Settings

1) The data center network: The data center network topology adopted here is a typical 8-ary FatTree topology. The topology contains 8 pods, each pod includes 4 aggregation switches and 4 edge switches. Each aggregation switch connects 4 core switches and 4 edge switches. There are 16 core switches in total. Each edge switch connects 4 servers. In total, 128 servers are connected by these edge switches. Each server involves 10 VM slots. All switches and servers are connected by $10Gbps$ links. We use the single-rooted equivalent abstraction of the 8-ary FatTree topology to conduct the following evaluations. The solution for the single-rooted equivalent abstraction topology can be easily transformed into a solution for the typical multi-rooted FatTree by adopting the Equal-Cost Multi-Path routing (ECMP) principle, while the resource consumed before and after conversion remains unchanged.

2) The scenarios: In this evaluation, in order to simulate the real data center load, we randomly generate some loads on each server and each link following a normal distribution given by $\mathcal{N}(2, 0.5)$ and $\mathcal{N}(1, 0.2)$, respectively. We consider two scenarios: the single VC request scenario and the multiple VC requests scenario. In the single VC request scenario, only a VC request is provided as the input. This scenario can be

used to describe a kind of online task arrival mode, where tasks arrive one by one. Once arrived, the task will enter the queue and wait to be processed. We test two settings for a VC request in this scenario, i.e., $< 8, 4 >$ and $< 16, 2 >$, to present how our proposal deals with the inherent trade-off between the server resource consumption and the bandwidth resource consumption. In the multiple VC requests scenario, multiple VC requests are provided as the input. This scenario can be used to describe anther kind of online task arrival mode, where tasks arrive in batches. Tenant requests arrive in the form of $< N, B >$, where N follows a normal distribution given by $\mathcal{N}(10, 2)$ and B follows a normal distribution given by $\mathcal{N}(1, 0.5)$. We generate 50 tenant requests to form a batch in the following evaluation.

3) The benchmarks: We compare our algorithm (referred to as TRO) to two benchmark algorithms. The first one is a shadow-based [10] algorithm (referred to as EXP-1). Specifically, the shadow-based solution is a widely-adopted solution for VM management, which reserves a backup VM slot for each of the original VMs. We use the heuristic algorithm proposed in [16] to employ the shadow-based solution to the VC abstraction. We also compare our algorithm to a newly proposed heuristic algorithm (referred to as EXP-2) in [16] that aims to minimize the number of total VM slots used when providing survivable VC embedding.

4) Evaluation metrics: The performance of the above benchmark algorithms is evaluated in terms of the total resource consumption (TC), the server resource consumption (SC), and the bandwidth resource consumption (BC).

B. Simulation Results

1) Single VC request scenario: In this subsection, we present the simulation results of our algorithm on a single VC request scenario. Table I illustrates the values of TC, SC and BC. Note that all the results are averaged among 2 independent tests (i.e., $< 8, 4 >$ and $< 16, 2 >$). It can be observed that a setting that increases the bandwidth resource savings will undermine the server's resource-saving performance and vice versa. When $\alpha = 0$, SC is 18.5 while BC is 16. When $\alpha = 1$, SC is 13 while BC is 25.5. By changing the value of α, we can control the trade-off between SC and BC. To apply the TRO algorithm to real data centers, the manager can choose an optimal setting for α that can achieve the most desirable performance.

TABLE I
PERFORMANCE COMPARISON WITH DIFFERENT VALUES OF α, EACH VALUE IS AVERAGED AMONG 2 INDEPENDENT TESTS.

α	TC	SC	BC
0	34.5	18.5	16
0.25	33.5	16	17.5
0.5	34.5	15	19.5
0.75	37.5	14.5	23
1	38.5	13	25.5

We then compare the performance of the TRO algorithm and two benchmark algorithms in the single VC request scenario. In this simulation, we set the value of α as 0.3, which has been proven to achieve a good balance between the server resource consumption and the bandwidth resource consumption in our simulations. We evaluate 5 settings for the VC request (i.e., $r_1 =< 8, 2 >$, $r_2 =< 8, 4 >$, $r_3 =< 12, 3 >$, $r_4 =< 16, 2 >$, and $r_5 =< 16, 4 >$) to see the impact on the different sizes of VC requests to the performance. Figure 5 shows the performance comparison of the three algorithms. It can be seen that the TRO algorithm outperforms the newly proposed EXP-2 algorithm in terms of bandwidth resource consumption (i.e., BC) while achieving a similar performance in terms of server resource consumption (i.e., SC). This is because TRO optimizes the total resource consumption while EXP-2 only optimizes the number of VM slots. In addition, EXP-2 works better than EXP-1. This is because EXP-1 always uses twice the number of actual demanded VM slots while the bandwidth resource consumption is not optimized.

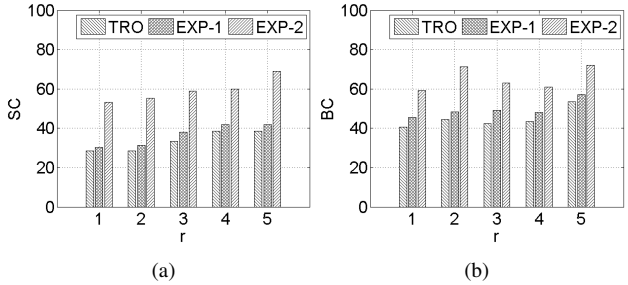


Fig. 5. The performance comparison in terms of a) the server resource consumption and b) the bandwidth resource consumption among three algorithms under 5 settings for VC requests (i.e., $r_1 =< 8, 2 >$, $r_2 =< 8, 4 >$, $r_3 =< 12, 3 >$, $r_4 =< 16, 2 >$ and $r_5 =< 16, 4 >$).

2) Multiple VC requests scenario: In this subsection, we show the performance of our algorithms in the multiple VC requests scenario. We also set the value of α as 0.3. We vary the size of a batch to see the impact on the performance. All the results are averaged among m independent tests, where m is the size of the batch. Results are illustrated in Figure 6. The simulation results are similar to those from the single VC request scenario. From Figure 6(a) we can conclude that, in terms of server resource consumption, TRO outperforms EXP-2 and is close to EXP-1. From Figure 6(b) we can see that TRO outperforms EXP-1 and EXP-2 in terms of bandwidth resource consumption. However, there are some special features in the multiple VC requests scenario. It can be seen that the gap between TRO and the other two benchmark algorithms is bigger than in single VC request scenario. This implies that TRO has a better request serving capacity in the batch scenario. The reason is that the design of TRO considers task scheduling while the other two algorithms fail to consider the optimization in the multiple requests scenario.

VI. RELATED WORK

Recently, there has been much interest in designing service abstraction that can provide bandwidth guarantee for multi-

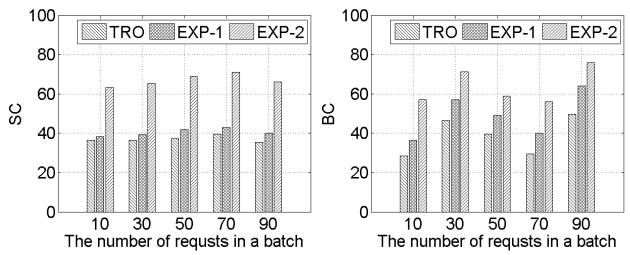


Fig. 6. The performance comparison in terms of a) the server resource consumption and b) the bandwidth resource consumption in the multiple VC requests scenario under 5 settings for the size of a batch (i.e., the size of a batch is set as 10, 30, 50, 70 and 90, respectively).

tenant cloud. The virtual cluster (VC) abstraction proposed by Ballani *et al.* in [3] is the most popular abstraction for batch-processing applications. Based on this abstraction, the authors also design a system named Oktopus to implement the proposed abstraction. After that, some other proposals that extend the VC abstraction are presented, such as TIVC [4], SVC [5], DCloud [20] and CloudMirror [21].

Much work has focused on providing virtual cluster embedding under a variety of goals. Zhu *et al.* [6] proposed an optimal algorithm to embed a VC request in the network with minimum bandwidth consumption. Their method is based on dynamic programming, and the time complexity is high. Rost *et al.* [7] discuss the computational complexity of star-topology embedding and hose embedding, and propose the HVC-ACE heuristic to benefit the embedding in terms of acceptance ratio and resource footprint. Some other work focuses on dynamic scaling VCs [9, 22]. However, few work has been done on ensuring a high survivability in VCE. The only relevant work on this topic is conducted by Yu *et al.* in [16], where the authors study the problem of providing a survivable VCE with minimal resource consumption as we do. However, they focus on a simple case where only the server resource in terms of the VM slot is considered for a single VC request scenario, while our work simultaneously optimizes the consumption of both the server and bandwidth resources for multiple VC requests scenario.

VII. Conclusion

In this paper, we study the resource optimization for the survivable embedding of virtual clusters in cloud data centers. Our aim is to minimize the consumption of resources in terms of server and bandwidth, while ensuring that both the resource constraints and the survivability constraints are not violated. We formally define this problem and analyze its complexity. Through a thorough understanding of the trade-off between the server resource usage and the bandwidth resource usage, we design efficient algorithms to obtain the resource optimization for both the server and bandwidth resources. Comprehensive experimental results verify that the overall resource consumption can be significantly reduced by applying our proposals.

References

[1] M. Mishra and A. Sahoo, "On theory of VM placement: Anomalies in existing methodologies and their mitigation using a novel vector based approach," in *IEEE CLOUD*, 2011.

[2] Q. Liu, G. Wang, J. Wu, and W. Chang, "User-controlled security mechanism in data-centric clouds," in *HPCC, NY, USA, August 24-26*, 2015, pp. 647–653.

[3] H. Ballani, P. Costa, T. Karagiannis, and A. I. T. Rowstron, "Towards predictable datacenter networks," in *ACM SIGCOMM, Toronto, ON, Canada, August 15-19*, 2011, pp. 242–253.

[4] D. Xie, N. Ding, Y. C. Hu, and R. R. Kompella, "The only constant is change: incorporating time-varying network reservations in data centers," in *ACM SIGCOMM , Helsinki, Finland - August 13 - 17*, 2012, pp. 199–210.

[5] L. Yu and H. Shen, "Bandwidth guarantee under demand uncertainty in multi-tenant clouds," in *IEEE ICDCS , Madrid, Spain, June 30 - July 3*, 2014, pp. 258–267.

[6] J. Zhu, D. Li, J. Wu, H. Liu, Y. Zhang, and J. Zhang, "Towards bandwidth guarantee in multi-tenancy cloud computing networks," in *IEEE ICNP , Austin, TX, USA, October 30 - Nov. 2,*, 2012, pp. 1–10.

[7] M. Rost, C. Fuerst, and S. Schmid, "Beyond the stars: Revisiting virtual cluster embeddings," *Computer Communication Review*, vol. 45, no. 3, pp. 12–18, 2015.

[8] W. Shi, C. Wu, and Z. Li, "An online mechanism for dynamic virtual cluster provisioning in geo-distributed clouds," in *IEEE INFOCOM, San Francisco, CA, USA, April 10-14*, 2016, pp. 1–9.

[9] L. Yu and Z. Cai, "Dynamic scaling of virtual clusters with bandwidth guarantee in cloud datacenters," in *IEEE INFOCOM, San Francisco, CA, USA, April 10-14*, 2016, pp. 1–9.

[10] E. Bin, O. Biran, O. Boni, E. Hadad, E. K. Kolodner, Y. Moatti, and D. H. Lorenz, "Guaranteeing high availability goals for virtual machine placement," in *IEEE ICDCS , Minneapolis, Minnesota, USA, June 20-24*, 2011, pp. 700–709.

[11] H. A. Alameddine, S. Ayoubi, and C. Assi, "Offering resilient and bandwidth guaranteed services in multi-tenant cloud networks: Harnessing the sharing opportunities," in *ITC , Würzburg, Germany, September 12-16*, 2016, pp. 1–9.

[12] M. Gagnaire, F. Diaz, C. Coti, C. Cerin, K. Shiozaki, Y. Xu, P. Delort, J.-P. Smets, J. Le Lous, S. Lubiarz *et al.*, "Downtime statistics of current cloud solutions," *International Working Group on Cloud Computing Resiliency, Tech. Rep*, 2012.

[13] S. Loveland, E. M. Dow, F. LeFevre, D. Beyer, and P. F. Chan, "Leveraging virtualization to optimize high-availability system configurations," *IBM Systems Journal*, vol. 47, no. 4, pp. 591–604, 2008.

[14] IBM, "Hardware designed for manageability - ibm predictive failure analysis," *IBM Internet site: http://www-05.ibm.com/hu/termekismertetok/xseries/dn/pfa.pdf, IBM Predictive Failure Analysis - by IBM Systems Director.*, 2010.

[15] cisco, "Cisco data center infrastructure 2.5 design guide," 2010.

[16] R. Yu, G. Xue, X. Zhang, D. Li, and Z. Cai, "Survivable and bandwidth-guaranteed embedding of virtual clusters in cloud data centers," in *IEEE INFOCOM*, 2017, pp. 1–9.

[17] P. Yalagandula and S. Nath, "Beyond availability: Towards a deeper understanding of machine failure characteristics in large distributed systems," in *WORLDS, San Francisco, CA, USA, December 5*, 2004.

[18] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *ACM SIGCOMM, Seattle, WA, USA, August 17-22*, 2008, pp. 63–74.

[19] W. Knödel, "A bin packing algorithm with complexity o(n log n) and performance 1 in the stochastic limit," in *Mathematical Foundations of Computer Science, Strbske Pleso, Czechoslovakia, August 31 - September 4*, 1981, pp. 369–378.

[20] D. Yu, C. Hu, H. C. Jiau, and K. Ssu, "dcloud: a document link provision cloud for software extension tasks," in *International Conference on Computer Science & Software Engineering, Porto, Portugal - July 10 - 12, 2013*, 2013, pp. 86–94.

[21] J. Lee, Y. Turner, M. Lee, L. Popa, S. Banerjee, J. Kang, and P. Sharma, "Application-driven bandwidth guarantees in datacenters," in *ACM SIGCOMM , Chicago, IL, USA, August 17-22*, 2014, pp. 467–478.

[22] C. Fuerst, S. Schmid, P. L. Suresh, and P. Costa, "Kraken: Online and elastic resource reservations for multi-tenant datacenters," in *IEEE INFOCOM , San Francisco, CA, USA, April 10-14*, 2016, pp. 1–9.