SEA: Stable Resource Allocation in Geographically Distributed Clouds

Sheng Zhang[†], Zhuzhong Qian[†], Jie Wu[§], and Sanglu Lu[†]
[†]State Key Lab. for Novel Software Technology, Nanjing University, China
[§]Department of Computer and Information Sciences, Temple University, USA
[†]zhangsheng@dislab.nju.edu.cn, {qzz,sanglu}@nju.edu.cn, [§]jiewu@temple.edu

***Abstract*—Today's public cloud providers typically deploy their small sized data centers in multiple geographically different locations, so as to improve data center power usage effectiveness and locate resources closer to users. A major challenge is resource allocation. Many results have been reported regarding this issue from the perspectives of virtual machine consolidation, network-aware virtual machine placement, traffic engineering, dynamic capacity provisioning, and so on. However, there has not been any focus on stable resource allocations, where no resource request or data center has any migration incentives. To the best of our knowledge, this paper is the first attempt at gaining a better understanding of the structure of the Stable rEsource Allocation (SEA) problem. We introduce a formal problem statement and develop two algorithms for the 1-dimensional (1-D) and 2-D cases, respectively. Simulation results show that the proposed algorithms have good scalability and convergence.**

I. Introduction

Over the past few years, cloud computing has been gaining more and more traction in both academia and industry. It is changing the way we access and retrieve information [1]. In traditional business applications, not only are the amount and variety of hardware and software required to run them daunting, but a large team of experts is also employed to install, configure, update, and secure them. Cloud computing helps us get rid of those headaches by shifting them to cloud providers.

In this paper, we consider resource allocation in distributed clouds. Most of today's public cloud providers deploy their computing resources in a number of geographically different locations in a wide-area network. Such a distribution has several desirable properties. First, it helps cloud providers utilize heterogeneities of both carbon footprints and electricity costs to achieve energy-efficient request routing and traffic engineering [2]. Second, it can locate resources closer to users and increase availability of cloud-computing resources [3].

There has been extensive research on allocating resources in cloud computing environments [2–6]. For example, Wang *et al.* [4] consolidate virtual machines based on bin packing; Zhang *et al.* [6] characterize workload and machine heterogeneities for dynamic capacity provisioning; and so on. However, most of prior work focuses on energy saving, and seldom consider the stability issue in geographically-distributed clouds.

When there are multiple distributed small sized data centers and multiple resource requests, it cannot be avoided that each one has a preference list over the opposite entities. That is, for a particular data center, it may prefer one resource request over another. For example, a data center with ample bandwidth favors communication-intensive requests; a high-performance data center prefers computation-intensive requests. A resource request may prefer data centers with small latencies. If we allocate resources without respecting these preferences, the allocation would be not stable, in the sense that a pair of data center and resource request may be appealing to each other. Then, such a data center may want to release one or more resource requests that are placed in it, as to admit the attractive request; such a resource request may want to migrate to or be reallocated in the attractive data center. Furthermore, a single release or migration may have a cascading effect: more releases and migrations arise. Such instability not only wastes (and crowds) the precious infrastructure resources, but also elongates the completion times of tenants' applications.

In this paper, we study the stable resource allocation problem between multiple geographically distributed clouds and multiple resource requests from tenants. To better understand the structure of the SEA problem, we first consider 1-D SEA in which only the CPU resource requirement is taken into account, and we design a polynomial-time solution based on the famous *proposal* algorithm for the stable marriage problem. We further incorporate virtual network embedding techniques into our solution to deal with 2-D SEA. Experimental results confirm the scalability and convergence of the proposed solutions. We believe it is important to explore how we can deal with resource allocation to best meet realistic needs. The contributions of this paper are twofold.

1) To the best of our knowledge, this is the first attempt that considers stability in allocating resources between multiple tenants and data centers. We provide a detailed and proper (from our perspectives) problem definition.
2) We develop two solutions for 1-D and 2-D SEA, respectively. Simulation results demonstrate the scalability and good convergence of the proposed algorithms.

The remainder of this paper is organized as follows. We go over existing related work in Section II. Section III introduces the notations and problem formulation. Then, in Sections IV and V, we present our solutions for 1-D and 2-D SEA, respectively. Simulation results are presented in Section VI. Section VII concludes the paper.

II. Related Work

The research in this paper builds on prior work related to resource allocation in clouds, virtual network embedding, and the stable marriage problem.

Resource allocation in clouds. The ability to provide efficient resource allocation is central to cloud computing. A large body of resource allocating algorithms have been proposed in the past [2–6]. Bin packing-based virtual machine consolidation with dynamic bandwidth demand was investigated in [4]. Network-aware virtual machine placement was considered in [5] and [3], where the authors leveraged graph partition and cluster techniques to minimize average traffic latency caused by network infrastructure. *HARMONY* [6] is a heterogeneity-aware resource management system for dynamic capacity provisioning in cloud environments. *FORTE* [2] focuses on traffic engineering to maximize data center power usage effectiveness; it dynamically controls the fraction of user-generated traffic directed to each data center in response to changes in carbon footprint and electricity price, which are location specific. Different from prior work, we try to explore the stability issue in resource allocation in clouds.

Virtual network embedding. Virtual network embedding [7] is to embed virtual networks with resource constraints in substrate networks, so as to efficiently utilize substrate resources. To cope with its NP-completeness [8], meta-heuristic-based algorithms were designed in [9, 10]. The study of embedding with unlimited substrate resources was conducted in [11], in which load balancing and reconfiguration are also considered. Substrate supports for path splitting were envisioned in [12]. A subgraph isomorphism detection-based embedding algorithm was proposed in [13]. Linear programming and deterministic/randomized rounding-based algorithms were developed in [14] to deal with virtual networks with location constraints. Topology-awareness was incorporated into embedding in [15, 16]. Opportunistic resource sharing-based embedding was investigated in [17]. Some other works [18, 19] have designed several excellent virtual network models which allow tenants to explicitly specify networking requirements, along with CPU needs, to achieve predictable application performance in clouds. Existing virtual network embedding algorithms can serve as basic sub-procedures in the 2-D version of our problem.

Stable marriage problem. In the classic stable marriage (SM) problem [20] with n men and n women, each person is associated with a strictly ordered preference list containing all the members of the opposite sex. A matching is stable if and only if there is no man-woman pair, both of whom have the incentive to elope. In its extensions, the preference list could be incomplete or contain ties. Two major variants of SM are the stable roommates problem and hospitals/residents problem [21]. For more details about SM, see [22].

III. Problem Statement

In this section, we first present the distributed cloud architecture and resource requests, then we introduce an important notation—*preference list*; finally, we formulate the Stable rEsource Allocation (SEA) problem.

A. Distributed Cloud Architecture

As we mentioned earlier, today's public cloud providers typically deploy their small sized data centers in multiple geographically different locations. Compared with centralized cloud architectures, such a distribution is appealing to both cloud providers and tenants.

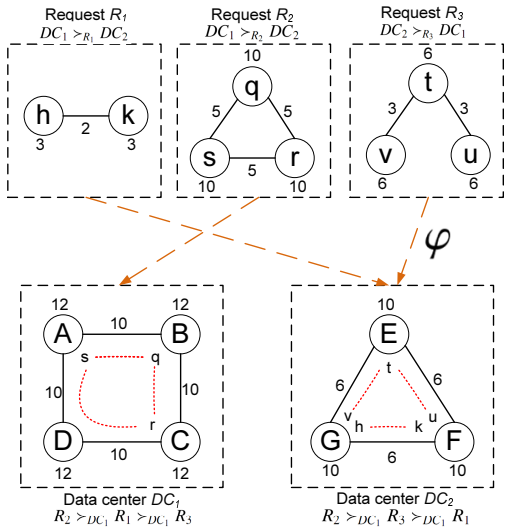


Fig. 1. An example of distributed clouds, resource requests, and an allocation plan. The resource capacity/demand is written next to the respective node or link that represents it. The dashed brown lines indicate a stable allocation plan. The dashed red lines represent the details of placing resource requests in data centers.

For cloud providers, instead of storing and running all tenants' applications in one large data center, they can place different kinds of resource demands in different small data centers to achieve energy-efficiency. For example, given a resource demand that is computation-intensive, a cloud provider may want to allocate the corresponding amount of resources in a data center which has the smallest carbon/computation ratio.

For tenants, the access latency is the main factor that influences their experiences. Through allocating resources in a data center that is closest to a tenant, distributed cloud architecture indeed reduces the tenant-perceived access latency.

In this paper, we assume that there are N small sized data centers, say DC_1, DC_2, and DC_N. Each small data center DC_i is modeled as a weighted undirected graph, $DC_i = (N_i^p, E_i^p, C_i^p, B_i^p)$, where N_i^p and E_i^p are the sets of physical machines (PMs) and communication links, respectively; C_i^p is the set of CPU capacities, and B_i^p is the set of bandwidth capacities. We would regret not mentioning that our model can be translated into several novel intra-data-center architectures, *e.g.*, fat-tree [23], VL2 [24], and BCube [25].

In Fig. 1, there are 2 small data centers: DC_1 is composed of 4 physical machines and 4 links, while DC_2 contains 3 physical machines and 2 links. The CPU (resp. bandwidth) capacity of each machine (resp. link) is written next to the respective node (resp. edge) that represents it. For example, the CPU capacity of physical machine B is 12.

B. Resource Requests

Following prior work in predictable data center networking [18, 19] and network virtualization [12, 14], we allow tenants to specify not only their CPU demands but also networking demands. In other words, the resource demands from a tenant form a virtual network.

We assume that there are M resource requests, say R_1, R_2, and R_M. Each resource request R_j is represented as a weighted

undirected graph, $R_j = (N_j, E_j, C_j, B_j)$, where N_j and E_j are the sets of virtual machines (VMs) and virtual communication links, respectively; C_j is the set of computational resource demands, and B_j is the set of networking resource demands.

Take R_1 in Fig. 1 for example, it consists of 2 virtual machines interconnected by a virtual link. The resource demand of each virtual machine (resp. link) is also written next to the respective node (resp. edge) that represents it.

C. Preference List

A high-performance data center prefers computation-intensive requests; a data center with ample bandwidth favors communication-intensive requests. A resource request prefers data centers with small latencies. In this paper, we capture such a kind of preference in the notation of *preference list*.

Each resource request R_j is associated with a strictly ordered preference list P_{R_j} containing all the N data centers. The rank of DC_i in the preference list of R_j is denoted as $P_{R_j}(DC_i)$. If R_j prefers DC_h to DC_k, we write $DC_h \succ_{R_j} DC_k$. For example, in Fig. 1, if $P_{DC_1} = \{R_2, R_1, R_3\}$, then we have $P_{DC_1}(R_2) = 1$, and $R_1 \succ_{DC_1} R_3$. A similar notation is used for data centers' preferences.

D. The Stable rEsource Allocation (SEA) Problem

Before presenting the SEA problem, we first give two formal definitions: allocation plan and blocking pair.

***Definition** 1: (Allocation Plan).* Let DC be the set of N data centers and R be the set of M requests; an allocation plan is a many-to-one correspondence $\varphi : R \rightarrow DC$. When $\varphi(R_j) = DC_i$, we say R_j is placed in DC_i.

Fig. 1 shows an allocation plan, where $\varphi(R_1) = DC_2$, $\varphi(R_2) = DC_1$, and $\varphi(R_3) = DC_2$.

***Definition** 2: (Blocking Pair).* A pair (R_j, DC_i) of resource request and data center is a blocking pair in an allocation plan φ if (*i*) R_j is not placed in DC_i, (*ii*) R_j is either unplaced or prefers DC_i to $\varphi(R_j)$, and (*iii*) DC_i is either under-utilized or prefers R_j to the worst request placed in it.

Then, SEA can be formulated as follows: given N data centers and M resource requests, find an allocation plan without any blocking pairs.

The reader could check that the allocation plan in Fig. 1 is stable, as there is no blocking pair. Let us look at another allocation plan φ': $\varphi'(R_1) = DC_1$, $\varphi'(R_2) = DC_2$, and $\varphi'(R_3) = DC_1$. In φ', (R_2, DC_1) forms a blocking pair, because (*i*) R_2 is not placed in DC_1 in φ', (*ii*) R_2 prefers DC_1 to DC_2, and (*iii*) DC_1 prefers R_2 to the worst request placed in it, *i.e.*, R_3.

We make three remarks. First, following the existing work [12, 14], we assume that one VM maps to one PM. When a tenant wants to deploy multiple VMs in one PM, we can treat all these VMs as one large VM by summing up their requirements. Second, the resource capacity should not be violated when placing resource requests in data centers. For example, after we placed R_2 in DC_1 (*i.e.*, s is placed in A, q is placed in B, and r is placed in C), there is no way to place either R_1 or R_3 in DC_1 unless violating resource capacities. Third, if there is not sufficient physical resources, some requests may end up with no placement.

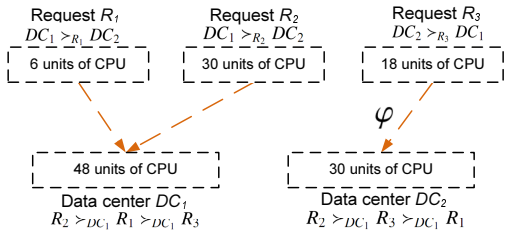


Fig. 2. An illustration of 1-D SEA. The dashed brown lines indicate a stable allocation plan. Note that this plan is different from that in Fig. 1.

In order to have a better understanding of SEA, we first study the 1-dimensional (1-D) case, where only CPU resource/demand is considered, then we study the 2-D case with the experience obtained from 1-D SEA.

IV. 1-D SEA

A. The Intuition

The 1-D SEA only considers the computing resource/demand, therefore, we further define $residual_i$ as the amount of residual computing resource in DC_i, that is, $residual_i = \sum_{n \in N_i^p} C_i^p(n)$; similarly, denote by $demand_j$ the amount of computing resource demand from R_j, that is, $demand_j = \sum_{n \in N_j} C_j(n)$. Fig. 2 shows the 1-D version of the example in Fig. 1. For instance, $residual_2 = 30$, and $demand_3 = 30$.

Different from the classical stable marriage problem [20] and its variant hospital/resident (HR) problem [21], 1-D SEA has an additional resource constraint. In this sense, the HR problem can be seen as a special (homogeneous) case of 1-D SEA where all requests have the same resource demand.

For the SM problem with n men and n women, Gale and Shapley designed the famous *proposal* algorithm [20], which can be briefly stated as follows. All men and women are initialized to be unmarried. In each step, an arbitrary unmarried man m proposes to his highest ranked woman w to whom he has not yet proposed. If w is also unmarried or w prefers m to her current partner m', then they become engaged (and m' becomes unmarried). Since each man proposed to each woman at most once, the algorithm terminates in n^2 proposals, at most, in the worst case.

However, the *proposal* algorithm does not apply to our 1-D SEA. This is due to heterogeneity—different resource requests have different amounts of resource demands. For a data center, the release of a low-priority resource request may allow the data center to admit not only the current request but also the previously rejected requests. We illustrate this using an example. Suppose that the preference list of a data center DC with a capacity of 10 is $R_c \succ_{DC} R_a \succ_{DC} R_b$; the resource demands of three requests R_a, R_b, and R_c are 9, 5, and 2, respectively. After R_a is placed in DC, we find that R_b cannot be placed at DC, because (*i*) the residual resource in DC is not sufficient for R_b, and (*ii*) DC prefers R_a to R_b. Then, we try to place R_c in DC. Since $R_c \succ_{DC} R_a$, we should release R_a and place R_c in DC. If we directly apply the *proposal* algorithm to

Algorithm 1 Allocation algorithm for 1-D SEA

1: **Input:** P_{DC_i} and $residual_i$ for $i = 1, ..., N$;
P_{R_j} and $demand_j$ for $j = 1, ..., M$
2: **Output:** φ that is stable
3: let $\varphi(R_j) \leftarrow 0$ for each R_j
4: let $ap_i \leftarrow \varnothing$ store the placed requests in each DC_i
5: let $changed \leftarrow true$
6: **while** $changed$ **do**
7: $changed \leftarrow false$
8: **for** $j = 1$ to M **do**
9: **if** $\varphi(R_j) \neq 0$ **then continue**
10: **for** $k = 1$ to N **do**
11: let DC_{idx} be $P_{R_j}[k]$
12: **if** $residual_{idx} > demand_j$ **then**
13: insert R_j into the ordered ap_{idx}
14: $residual_{idx} \leftarrow (residual_{idx} - demand_j)$
15: $\varphi(R_j) \leftarrow idx$, $changed \leftarrow true$
16: **break**
17: **else**
18: $h \leftarrow$ **findASubset**(ap_{idx}, R_j)
19: **if** $h \leq 0$ **then continue**
20: let $S \leftarrow \{ ap_{idx}[h], ap_{idx}[h+1], ..., ap_{idx}[ap_{idx}.length] \}$
21: $ap_{idx} \leftarrow ap_{idx} \setminus S$
22: insert R_j into the ordered ap_{idx}
23: $residual_{idx} \leftarrow (residual_{idx} - demand_j + \sum_{R_x \in S} demand_x)$
24: $\varphi(R_x) \leftarrow 0$ for all $R_x \in S$
25: $\varphi(R_j) \leftarrow idx$, $changed \leftarrow true$
26: **break**
27: **end if**
28: **end for**
29: **end for**
30: **end while**

31: **Sub-procedure: findASubset**(ap_{idx}, R_j)
32: let $release \leftarrow 0$
33: **for** $h = ap_{idx}.length$ to 1 **do**
34: $released \leftarrow released + (demand\ of\ ap_{idx}[h])$
35: **if** DC_{idx} prefers $ap_{idx}[h]$ to R_j **then return** 0
36: **if** $released + residual_{idx} \geq demand_j$ **then return** h
37: **end for**
38: **return** 0

our case, this is the final result, because the algorithm tries to place each request in each data center at most once. However, we notice that the previously rejected request, namely, R_b, could be also placed together with R_c in DC.

B. The Solution

Based on the aforementioned argument, we design the allocation algorithm for 1-D SEA, as shown in Alg. 1. Generally speaking, Alg. 1 runs in multiple rounds; the flag variable *changed* indicates whether there is any change in the allocation plan, and the algorithm terminates when there are no changes between the allocation plans of two consecutive rounds.

Specifically, in the initialization phase (lines 3-5), all requests are set to be unplaced; the ordered set ap_i stores the requests that are placed in DC_i; we denote by $ap_i[h]$ and $P_{R_j}[i]$ the h-th element (*i.e.*, resource request) in the ordered set ap_i and the i-th element (*i.e.*, data center) in the preference list of R_j, respectively. During the main loop (in each round), for each unplaced request, we check whether it could be placed in each data center in its preference list. For R_j and DC_{idx}, if the residual resource in DC_{idx} is more than the demand of R_j (line 12), we insert R_j into the ordered DC_{idx}, and make some necessary updates (lines 13-15). Otherwise, we check whether R_j could be placed in DC_{idx} through releasing a subset of placed requests, which are less favorable than R_j to DC_{idx} (lines 18-25). The sub-procedure **findASubset** returns such a subset, if it exists (lines 31-38). As ap_{idx} is ordered, **findASubset** first checks (*i*) whether DC_i prefers $ap_{idx}[ap_{idx}.length]$ to R_j, and (*ii*) whether the remove of $ap_{idx}[ap_{idx}.length]$ releases sufficient resources; then, **findASubset** incrementally adds $ap_{idx}[ap_{idx}.length-1]$, $ap_{idx}[ap_{idx}.length-2]$, ..., $ap_{idx}[1]$ into the subset for checking. After such a subset is identified, we remove this subset of requests from ap_{idx} (lines 20-21), insert R_j into the ordered DC_{idx}, and make some necessary updates (lines 22-25).

Correctness. We can briefly prove the correctness of Alg. 1 by contradiction. Suppose that (R_j, DC_{idx}) is a blocking pair in the resulting φ. According to our algorithm, we check whether R_j can be placed in DC_{idx} before checking whether R_j can be placed in $\varphi(R_j)$ (lines 10-11), but DC_{idx} cannot admit R_j, which implies that, DC_{idx} prefers the worst request placed in it to R_j, contradicting the definition of blocking pair.

Complexity. The algorithm terminates when there are no changes between the allocation plans of two consecutive rounds, which implies that the number of unplaced resource requests decreases by at least one in a round. So the total number of rounds is at most M. In each round, for each request, the algorithm checks whether each data center in its ordered preference list could admit the request: the insertion (line 13 or 21) takes $O(M)$ time, and the sub-procedure takes also $O(M)$ time. Putting them together, the total time complexity of Alg. 1 is $O(M \cdot M \cdot N \cdot (max(M, 2M))) = O(M^3 N)$.

V. 2-D SEA

In 2-D SEA, we denote by $RC^p(n)$ and $RB^p(e)$ the residual resource of a physical machine n and a physical link e, respectively. For example, in Fig. 1, after placing R_2 in DC_1, we have $RC^p(A) = RC^p(B) = RC^p(C) = 2$, $RC^p(D) = 12$, and $RB^p(\overline{AB}) = RB^p(\overline{BC}) = RB^p(\overline{CD}) = RB^p(\overline{DA}) = 5$.

2-D SEA allows tenants to specify their networking requirements along with CPU demands. The resources requirements from a tenant then form a virtual network. We are no longer able to place or release a resource request by addition or subtraction. The key challenge is how to place a 2-D resource request (in the form of a virtual network) into a data center. This is similar to virtual network embedding problem [7], which is proven to be NP-Complete [8].

Following our previous work [16, 17], we propose a simple greedy approach for placing a 2-D resource request in a data center, shown in Alg. 2. The basic idea is to first deploy virtual machines, then use the shortest paths between physical machines to embed virtual links. In the virtual machine embedding phase (lines 3-12), we first sort virtual and physical machines in Q and T, respectively; then, we try to place each virtual machine from the head to the end of Q in the first

Algorithm 2 Placing a 2-D request in a data center

1: **Input:** a 2-D request $R_j = (N_j, E_j, C_j, B_j)$, and the current state of a data center $DC_i = (N_i^p, E_i^p, RC_i^p, RB_i^p)$
2: **Output:** a successful placement or a rejection
3: **for all** $n^p \in N_i^p$ **do** $unused(n^p) \leftarrow 1$ **end for**
4: let $Q \leftarrow$ sorted N_j with increasing $C(n)$
5: let $T \leftarrow$ sorted N_i^p with increasing $RC^p(n^p)$
6: **for** $k = 1$ to $Q.length$ **do**
7: let $h \leftarrow 1$
8: **while** $RC^p(T[h]) \cdot unused(T[h]) < R(Q[h])$ **do** h++ **end while**
9: **if** $h > T.length$ **do return** a rejection
10: place $Q[i]$ in $T[i]$
11: $unused(T[i]) \leftarrow 0$
12: **end for**
13: **for all** $e \in E_j$ **do**
14: place e in the shortest path between the respective physical hosts with a sufficient bandwidth
15: **end for**
16: **return** a successful placement

physical machine that has a sufficient amount of resource. In the virtual link embedding phase (lines 13-15), we place each virtual link in the shortest path between the respective physical hosts with a sufficient bandwidth.

The allocation algorithm for 2-D SEA is similar to Alg. 1; we only have to replace lines 12 and 17 in Alg. 1 with Alg. 2-based codes. We briefly summarize Sections IV and V by providing a few remarks.

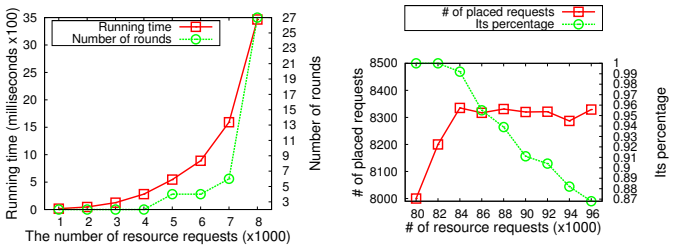
Tenant-optimality. Based on the research results about SM [20], our algorithms are actually tenant-optimal, since we can see the algorithms as sequences of proposals from tenants to cloud providers. By 'tenant-optimal' we mean that, every tenant gets the best possible data center among all stable allocation plans. Reversing the directions of proposals, we can get provider-optimal algorithms.

Provider-tenant-equalness. Given a SEA instance, the number of stable allocation plans could be exponentially many [26]. A natural question arises: how to evaluate a plan? There are several criteria, including regret, egalitarian, and equalness. We are interested in provider-tenant-equalness, where we want to find an allocation plan that fairly treats providers and tenants. The provider-tenant-equalness problem minimizes the following objective:

$$equalness(\varphi) = \sum_{(R,\varphi(R)\in\varphi)} P_R(\varphi(R)) - \sum_{(R,\varphi(R)\in\varphi)} P_{\varphi(R)}(R)$$

And this optimization problem is NP-hard [27].

Incomplete preference list. In a distributed clouds environment, a single entity cannot obtain all the global information. Thus, a tenant's preference list may contain only a part of all small sized data centers. In this case, we can slightly modify our definitions of block pair and stable allocation plan without major changes. The proposed solutions, with a slight modification, can be applied to find a stable allocation.



(a) Running time and number of running rounds

(b) Number of successfully placed requests and its percentage

Fig. 3. Scalability evaluation

VI. Numerical Results

In this section, we evaluate the performance of the proposed algorithms and make some remarks.

A. Simulation Setup

We assume that there are $N = 100$ small sized data centers. We vary the number of resource requests (*i.e.*, M) from 1,000 to 10,000, and see the impact of M on the running time and number of running rounds. The preference list of each data center and each resource requests is a random permutation of the opposite entities. In 1-D SEA, the capacity of each data center is uniformly generated between 400 and 500, and the demand of each resource request is uniformly generated between 2 and 10.

In 2-D SEA, we construct the topology of each data center as follows: the number of physical machines is determined by a uniform distribution between 50 and 100, and each pair of physical machines is connected with a probability of 0.6; after we generate such a topology, we check whether it is connected; if it is not, we just regenerate it until we get a connected topology. The topology of each resource request is obtained in a similar way, except that the number of virtual machines is between 2 and 10, and each pair of virtual machines is connected with a probability of 0.5. Both CPU and bandwidth capacities in data centers are generated uniformly from the interval between 50 and 100. Both CPU and bandwidth demands in resource requests are generated uniformly from the range between 2 and 10.

B. Simulation Results

We provide the results from 1-D SEA. The results of 2-D SEA are similar, and are omitted due to space limitations.

Running time and number of rounds. Fig. 3(a) shows how the running time and the number of running rounds changes by varying the number of resource requests. We note that, the running time increases slowly when M is relatively small, and fast when M is larger than 7,000. There are two main reasons behind this phenomenon. One is that, the time complexity of Alg. 1 is upper bounded by $O(M^3N)$, and the other is that, since N is fixed, when the number of requests increases, the number of running rounds also increases, as shown by the green line in the same figure. These results demonstrate the good scalability and quick convergence of the proposed algorithm. This property provides us with a randomized Monte Carlo algorithm for 1-D SEA: we just truncate Alg. 1 after a fixed amount of running time, then return the allocation plan.

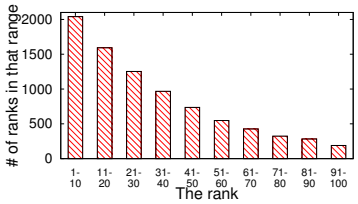


Fig. 4. Quality of allocation from the perspective of requests

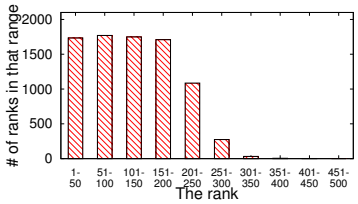


Fig. 5. Quality of allocation from the perspective of data centers

Number of successfully placed requests and its percentage. In Fig. 3(b), we see that the number of resource requests goes up from 8,400 to 9,600, and the number of successfully placed requests stays almost unchanged. We also find that, the percentage of successfully placed requests decreases when the number of requests increases from 8,000 to 9,600, because the fixed amount of physical resources can only admit about 8,400 resource requests.

Quality of allocation. We evaluate the quality of allocation from two perspectives. From the viewpoint of requests, we want to see how many requests are finally placed at their most favorable data centers, respectively? How many requests are finally placed at their second most favorable data centers, respectively? and so on. Fig. 4 shows the simulation results where there are 9,000 requests. We notice that, about 2,000 requests are placed at their top-10 most favorable data centers; about one-third of all requests are placed at their top-30 most favorable data centers. Similarly, Fig. 5 shows the results from the viewpoint of data centers. We see that, about 20%/80% of requests are placed at data centers which consider these requests as their respective top-50/top-200 most favorable requests. These results confirm the high quality of the allocation plans generated by the proposed algorithms.

Overall, we admit that comparing our solutions to alternatives is necessary, and a more precise definition of quality of allocation is required. We leave them as future work.

VII. Conclusions

In this paper, we give a formal problem statement and propose scalable solutions for 1-D and 2-D SEA. Numerical results show that our algorithms converges quickly. In future work, we intend to take the quality of stability (*e.g.*, egalitarian, equalness, etc.) into account, and will also attempt to incorporate incomplete preference lists into our scenario.

Acknowledgements

We thank the ICC reviewers for feedback on earlier drafts of the paper. This work was supported in part by NSFC Grants (61073028, 61202113, 61321491, and 91218302), Key Project of Jiangsu Research Program Grant (BE2013116), Jiangsu NSF Grant (BK2011510), Research and Innovation Program for Jiangsu Graduates Grant (CXZZ12_0055), Program A for Outstanding PhD candidate of Nanjing University (201301A08), US NSF grants (ECCS 1128209, CNS 1065444, CCF 1028167, CNS 0948184, and CCF 0830289), and EU FP7 IRSES MobileCloud Project Grant (612212).

References

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica *et al.*, "A view of cloud computing," *ACM CACM*, vol. 53, no. 4, pp. 50–58, 2010.

[2] P. X. Gao, A. R. Curtis, B. Wong, and S. Keshav, "It's not easy being green," in *ACM SIGCOMM 2012*.

[3] M. Alicherry and T. Lakshman, "Network aware resource allocation in distributed clouds," in *IEEE INFOCOM 2012*.

[4] M. Wang, X. Meng, and L. Zhang, "Consolidating virtual machines with dynamic bandwidth demand in data centers," in *IEEE INFOCOM 2011*.

[5] X. Meng, V. Pappas, and L. Zhang, "Improving the scalability of data center networks with traffic-aware virtual machine placement," in *IEEE INFOCOM 2010*.

[6] Q. Zhang, M. F. Zhani, R. Boutaba, and J. L. Hellerstein, "Harmony: Dynamic heterogeneity-aware resource provisioning in the cloud," in *IEEE ICDCS 2013*.

[7] N. Chowdhury and R. Boutaba, "A survey of network virtualization," *Computer Networks*, vol. 54, no. 5, pp. 862–876, 2010.

[8] D. G. Andersen, "Theoretical approaches to node assignment," Dec. 2002, Computer Science Department, Carnegie Mellon University.

[9] R. Ricci, C. Alfeld, and J. Lepreau, "A solver for the network testbed mapping problem," *ACM SIGCOMM CCR*, vol. 33, no. 2, pp. 65–81, 2003.

[10] S. Zhang, Z. Qian, S. Guo, and S. Lu, "FELL: A flexible virtual network embedding algorithm with guaranteed load balancing," in *IEEE ICC 2011*.

[11] Y. Zhu and M. Ammar, "Algorithms for assigning substrate network resources to virtual network components," in *IEEE INFOCOM 2006*.

[12] M. Yu, Y. Yi, J. Rexford, and M. Chiang, "Rethinking virtual network embedding: substrate support for path splitting and migration," *ACM SIGCOMM CCR*, vol. 38, no. 2, pp. 17–29.

[13] J. Lischka and H. Karl, "A virtual network mapping algorithm based on subgraph isomorphism detection," in *ACM VISA 2009*.

[14] M. Chowdhury, M. Rahman, and R. Boutaba, "ViNEYard: Virtual network embedding algorithms with coordinated node and link mapping," *IEEE/ACM TON*, vol. 20, no. 1, pp. 206–219, 2012.

[15] X. Cheng, S. Su, Z. Zhang, H. Wang, F. Yang, Y. Luo, and J. Wang, "Virtual network embedding through topology-aware node ranking," *SIGCOMM CCR*, vol. 41, pp. 38–47, April 2011.

[16] S. Zhang, Z. Qian, J. Wu, and S. Lu, "An opportunistic resource sharing and topology-aware mapping framework for virtual networks," in *IEEE INFOCOM 2012*.

[17] S. Zhang, Z. Qian, J. Wu, S. Lu, and L. Epstein, "Virtual network embedding with opportunistic resource sharing," in *IEEE TPDS, DOI: 10.1109/TPDS.2013.64*.

[18] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron, "Towards predictable datacenter networks," in *ACM SIGCOMM 2011*.

[19] D. Xie, N. Ding, Y. C. Hu, and R. Kompella, "The only constant is change: incorporating time-varying network reservations in data centers," in *ACM SIGCOMM 2012*.

[20] D. Gale and L. S. Shapley, "College admissions and the stability of marriage," *The American Mathematical Monthly*, vol. 69, no. 1, pp. 9–15, 1962.

[21] K. Iwama and S. Miyazaki, "A survey of the stable marriage problem and its variants," in *IEEE ICKS 2008*.

[22] D. Gusfield and R. W. Irving, *The stable marriage problem: structure and algorithms*. MIT press Cambridge, 1989.

[23] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *ACM SIGCOMM 2008*.

[24] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "Vl2: a scalable and flexible data center network," in *ACM SIGCOMM 2009*.

[25] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu, "BCube: a high performance, server-centric network architecture for modular data centers," in *ACM SIGCOMM 2009*.

[26] R. W. Irving and P. Leather, "The complexity of counting stable marriages," *SICOMP*, vol. 15, no. 3, pp. 655–667, 1986.

[27] A. Kato, "Complexity of the sex-equal stable marriage problem," *Japan Journal of Industrial and Applied Mathematics*, vol. 10, no. 1, pp. 1–19, 1993.