Traffic and Failure Aware VM Placement for Multi-tenant IaaS Cloud

Xin Li and Chen Qian
Department of Computer Science, University of Kentucky
Email: xin.li@uky.edu, qian@cs.uky.edu

***Abstract*—In an infrastructure as a service (IaaS) cloud, tenants want to receive reliable services and the cloud provider intends to reducing intra-network traffic in order to provide more services. Achieving the requirements of both sides is a challenging problem. Current tenant abstraction models cannot provide enough information for the cloud provider to optimize network traffic while satisfying reliability requirements. Based on the analysis of the traffic traces of a real multi-tenant cloud, we develop a new tenant abstraction model and design a novel virtual machine placement algorithm, called NETMAP. NETMAP optimizes the cost of network traffic incurred by tenant applications under the reliability constraints from the tenants. Trace-driven simulation results show that NETMAP outperforms a number of possible solutions. Even if there is traffic estimation error from the abstraction model, NETMAP still yields an optimized VM placement.**

I. Introduction

An infrastructure as a service (IaaS) cloud offers computing and storage resources to tenants by allowing tenants to deploy computing instances, i.e., virtual machines (VMs), to the physical machines of a cloud provider's data center network. In a multi-tenant cloud, tenants deploy their VMs by making VM requests to the cloud provider. VMs belonging to a same tenant will communication and form a *tenant network*. To meet the tenant requirements and to provide better performance and reliability, the cloud provider needs to optimize VM placement on physical machines towards various goals.

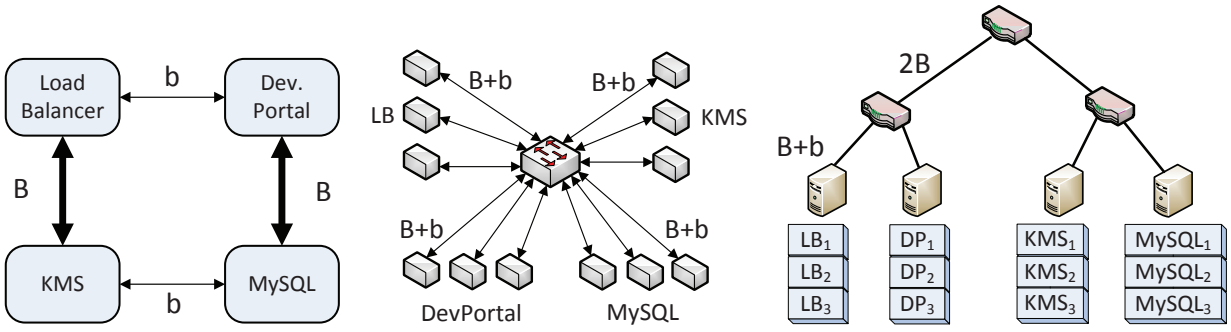
Many tenant applications are throughput-sensitive (such as MapReduce [9]) or latency-sensitive (such as search engines). As many data center networks today are bandwidth oversubscribed [3], these applications may become unavailable or unresponsive under network congestion or failures. Reducing the intra-network traffic reduces the probability of congestion, and thus provides shorter latency and larger bandwidth for VM communication. Therefore the cloud provider prefers to reduce the traffic within its data center network [23]. An intuitive solution is to place VM pairs with heavy traffic on servers that are close in the network. Furthermore, the cloud provider should guarantee a certain level of reliability under network failures. For example, a group of VMs performing a same function in a tenant network should not be placed to a single failure domain such as a server or a rack. Joint traffic-failure tolerance optimization is considered a complicated and hard problem [5]. It is mainly because reducing traffic requires VMs of a tenant to be placed together but reliability under failures requires VMs to spread out in the network, resulting in a dilemma.

978-1-4673-7113-1/15/$31.00 ©2015 IEEE

In this paper, we focus on the *VM placement strategy that can reduce the data center network traffic while providing failure tolerance for a multi-tenant cloud*. Unfortunately current solutions for VM placement may not be able to satisfy all these requirements. Traffic-aware VM placement such as TMVPP [23] and SecondNet [17] have been proposed to reduce network traffic. A recent work [5] considers both network traffic and failure constraints to improve existing VM placement. However these methods assume full knowledge of the traffic matrix of every tenant network, including the exact bandwidth usage of every VM pair. Such assumption is valid for enterprise production data center. However full traffic knowledge may not be provided by tenants to a multi-tenant cloud. On the other hand, network abstraction models are proposed for tenants to express bandwidth requirements without giving full traffic matrix, such as the hose model [10] and virtual cluster [4]. These models emulate physical topologies but can hardly reflect the communication patterns caused by the application structure of a tenant network. Hence they may result in sub-optimal VM placement in a multi-tenant cloud.

In this work, we analyze real traffic trace data of a multi-tenant data center network deployed by a giant telecommunication corporation in the USA. Based on our results we design a function-based abstraction model (FAM) for tenants to provide their network abstraction as well as reliability requirements. We then propose a model for the cloud provider to optimize network traffic. Our network distance model can effectively characterize the oversubscription facts in a data center network. We show that the optimization problem is NP-hard and propose a novel VM placement algorithm called NETMAP. Our contributions can be summarized as follows:

1) We design a new network abstraction model FAM for tenants. FAM provides function relationships which are important for both traffic estimation and reliability requirements.
2) We propose a new network distance model for any hierarchical tree topology which characterizes oversubscription in data center networks.
3) We design a novel VM placement algorithm that reduces the network traffic while preserving reliability



(a) Bandwidth usage (b) hose model (c) possible placement using the hose model

Fig. 1. VM placement using the hose model for a tenant network

requirements of tenants. Experimental results show that NETMAP can achieve good performance for both real and synthetic traffic data with or without traffic estimation errors.

The rest of this paper is organized as follows. Section II presents the observations from the measurement of real multi-tenant traffic data. We model the tenant network, physical network, and the optimization problem in Section III and design the NETMAP algorithm in Section IV. Section V shows the performance evaluation. Finally we conclude this work in Section VII.

II. Measurement Results and Observations

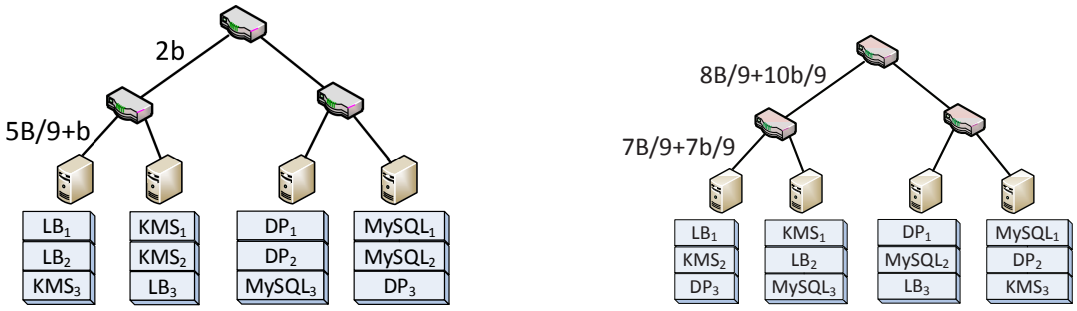
We analyzed the configuration and traffic data of a real multi-tenant data center network deployed at a giant telecommunication corporation in the USA, referred as "enterprise traffic trace data" in this paper. For privacy issues, the trace is not publicly accessible. We observed a number of important facts about the bandwidth usage of tenant networks. First, the application of a tenant network can be decomposed into a number of *functions*. Each VM serves only one function, but multiple VMs may serve a same function. Second, traffic between different VM pairs varies a lot, which is similar to the flow size distribution in the Internet [20]. However, if VMs x_1 and x_2 belong to a same function and y_1 and y_2 belong to another same function, then the traffic from x_1 to y_1 is very close to the traffic from x_2 to y_2. In other words, traffic between two functions is evenly contributed by each VM pair of the two functions. Third, traffic between a pair of VMs is relatively stable, which is also reported by Meng et al. [23]. Last, tenants do not communicate with each other.

Even though in the trace there is no intra-traffic within VMs in a same function, it is possible in other cases. For example, some distributed storage VMs need intra-traffic to keep the consistency of backup files [13]. When the traffic estimation of a tenant network is unavailable, the hose model is used to express the intra-traffic abstraction [19], which means that each VM talks to all other VMs at a very close rate.

Fig. 1(a) shows a typical example of functions in a tenant network we observed from the enterprise data (some functions are not shown due to the ease of illustration). The tenant network consists of four functions, namely the load balancer (LB), key management system (KMS), MySQL, and developer portal (DP). Three VMs serve each function. Hence there are 12 VMs in total. Suppose the bandwidth usage between LB and KMS is B in both directions. There are $3 \times 3 = 9$ pairs evenly contributing the bandwidth. Hence each LB VM communicates with each KMS VM using bandwidth $B/9$. The bandwidth usage between another pair of functions, say b for KMS and MySQL, maybe less than B by over an order of magnitude. Suppose $B = 10b$. If the tenant applies the commonly used the hose model, all VMs are connected to a virtual switch. Each VM has $(B+b)/3$ traffic to the virtual switch, as shown in Figure 1(b). This tenant network abstraction can only reflect the incoming and outgoing traffic for each VM, but hides the traffic pattern between different functions.

A possible VM placement using the hose model is shown in Fig. 1(c), where three VMs belonging to a same function are deployed to a same server. The physical topology is a simple tree structure, in which top-of-rack (ToR) switches connect physical servers by in-rack links and an aggregate switch connects ToRs by cross-rack links. The placement in Fig. 1(c) causes high network traffic especially for the cross-rack links. The overhead on a cross-rack link is $2B$ on both directions. Moreover, such placement has no failure tolerance. If the left rack fails, all VMs for LB and DP fail and the application of the tenant becomes unavailable. Compared to an optimized placement shown in Fig. 2(a), the overhead on a cross-rack link can be as low as $2b$, only 10% compared with that in Figure 1(c). If the placement should satisfy a reliability requirement that any rack failure does not remove all VMs serving a same function, a possible optimized placement is shown in Fig. 2(b). In this example the overhead on a cross-rack link is $\frac{8}{9}B + \frac{10}{9}b$, still much lower than (about 50%) that in Fig. 1(c).

In this paper we propose a new tenant network abstraction model based on the function relationship within an application and a VM placement method using this model.



(a) Optimized placement

(b) Optimized placement with failure tolerance

Fig. 2. Traffic and failure aware VM placement

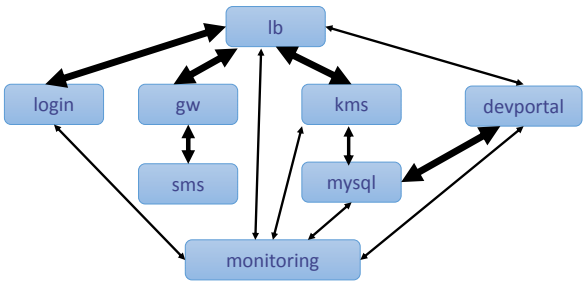


Fig. 3. Function-based abstraction model of a tenant network

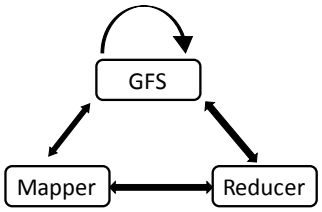


Fig. 4. Function-based abstraction model of MapReduce

III. Models and Problem Formulation

A. Function-based Abstraction Model

The function-based abstraction model (FAM) proposed in this paper is a graph, where each vertex is a function of the application and a link between two functions indicates they communicate. A vertex has two attributes, the number of VMs and reliability requirement. The reliability requirement indicates the maximum fraction of VMs of this function that can be placed to a failure domain. Suppose we only consider network failures and a rack is the biggest failure domain. If the number of VMs of function F is m and the reliability requirement is $r \in [0,1]$, then at most $\lfloor m * r \rfloor$ VMs can be deployed to a same rack. A link has an attribute representing the traffic amount. The traffic amount of FAM could be very coarse-grained, such as 0 or 1, and the tenant also need to give the estimated traffic for one unit of traffic. Suppose the traffic between functions is represented from 0 to 5, and the estimated traffic for one unit is t. In this case, the bandwidth usage presented by 3 is actually $3 * t$. The grain fineness completely depends on the information available of the tenant. An example FAM of a typical tenant network from a enterprise network is shown in Fig. 3.

Compared to two commonly used network abstractions, exact traffic matrix and the hose model, FAM does not require the tenant to provide the exact amount of traffic and can reflect the traffic pattern between different functions. To apply FAM to our VM placement algorithm, we need to obtain two important inputs, namely traffic estimation and network distance, which we next explain in Sec. III-B and Sec. III-C, respectively.

B. Traffic matrix estimation

A traffic matrix M of a tenant network is an $N \times N$ matrix, where N is the number of VMs of the tenant network and the element m_{xy} is the (estimated) traffic amount from VM x to VM y. M is served as an input of the VM placement algorithm of this paper. In a general FAM such as that in Fig. 3, the tenant may need to specify the traffic amount on each FAM link, and the traffic will be evenly divided by the total number of pairs between the two functions of the link. Suppose in Figure 3, the total traffic from LB to KMS is T and each function has 3 VMs. Then the estimated traffic value from an LB VM to a KMS VM is $T/9$ in the traffic matrix. In the performance evaluation in Sec. V-C, we show that we only need coarse-grained traffic values to achieve good VM placement.

In multi-tenant clouds, a large proportion of jobs deployed by tenants are data-intensive applications, which have paradigm frameworks, such as MapReduce [9], Dryad [18], Hive [1], and Pregel [22]. Their traffic patterns have been investigated in [6] and can be utilized by our work. Fig. 4 shows an illustration of the function-based abstraction model (FAM) of MapReduce [9]. A typical communication pattern can be presented as follows. Mappers and reducers are both clients of Google file system (GFS) [13]. Mappers first read the files from GFS, and output the intermediate result to the reducers. The final result is stored in GFS after being processed by the reducers. Within

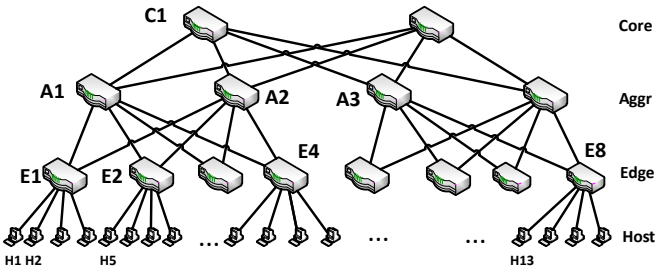


Fig. 5. Hierarchical topology example

GFS, two additional replicas are backed up in case of failure, where intra-traffic is introduced.

C. Network topology and distance

Suppose the traffic matrix is fixed. Placing two VMs in different locations will increase the bandwidth usage on different links. We present a method to model the network cost for different VM locations. We use a *position* to refer to the hardware resource (including CPU, memory and disk) allocated for a VM on a server. A server can have multiple positions and each position is allocated to only one VM. Let p_i and p_j be the positions allocated to VMs i and j. We use $F_{ij} * D_{p_i,p_j}$ to denote the network cost to support the communication between i and j, where D_{p_i,p_j} is the *network distance* between two positions p_i and p_j. Intuitively, if the network distance is larger, the traffic between the two positions are more likely to cause congestion [26]. Unlike some existing methods that only uses number of links on the path [23], our network distance model also considers the subscription ratio of each link on the path. The higher the subscription ratio of a link, the more traffic contend for the bandwidth on the link. In this sense, links of higher subscription ratio should have higher weight. Many data centers are oversubscribed. Some Facebook data centers has an subscription ratio as high as $40:1$ [11].

In this paper, we only consider *hierarchical tree based networks* which are the state-of-art data center topologies [7], including simple tree, FatTree [3], and VL2 [15]. A hierarchical multi-rooted tree topology can be built with commodity switches to provide multiple equal-cost paths between any pair of servers. We show an example of hierarchical multi-rooted tree in Fig. 5. A multi-rooted hierarchical tree topology has three vertical layers: edge layer, aggregate layer, and core layer. A *pod* is a management unit down from the core layer, which consists of a set of interconnected end hosts and a set of edge and aggregate switches that connect these hosts.

The network distance between two positions p_i and p_j is defined as:

$$\sum_{q \in \mathbb{Q}(p_i,p_j)} w_q \sum_{l \in q} r_l$$

where $\mathbb{Q}(p_i,p_j)$ is the set of all network pathes connecting p_i and p_j, q is one path among them, l is an upper-link on the path, w_q is the weight $\in [0,1]$ of path q, and r_l is the subscription ratio of link l. w_q is calculated by the likelihood of the traffic between p_i and p_j using path q. Hence $\sum_{q \in \mathbb{Q}(p_i,p_j)} w_q = 1$. Hence w_q can also be considered as the fraction of flows between two positions that use path q. Note that we only consider upper-link regarding the network distance, because the subscription ratio of a down-link is always 1 and can be ignored in our network distance calculation. Down-links can only be congested due to bad scheduling or unbalanced flows.

Consider an outgoing upper-link L of switch s, and let S_{in} denote the set of the set of incoming upper-links, S_{out} denote the set of outgoing upper-links, r_l denote a link's *subscription ratio*, and B_l denote a link's bandwidth. Then the subscription ratio of L is:

$$r_L = \frac{\sum_{l \in S_{in}} r_l * B_l}{\sum_{l \in S_{out}} B_l}$$

Note that the subscription ratio of a link connecting a server to its access switch is 1 and have no oversubscription.

Use the hierarchical multi-rooted tree in Fig. 5 as an example, where the capacity of every link is 1 Gbps. The network distance between two servers H_1 and H_2 is 1. It is because the subscription ratio of the upper-link is 1. Consider the distance between H_1 and H_5. There are two possible routing pathes, $H_1-E_1-A_1-E_2-H_5$ and $H_1-E_1-A_2-E_2-H_5$. Flows are evenly distributed on both paths. Thus the weight of each path is 0.5. As for path $H_1-E_1-A_1-E_2-H_5$, the upper-links are $l(H_1,E_1)$ and $l(E_1,A_1)$. $l(H_1,E_1)$'s subscription ratio is 1 and $l(E_1,A_1)$'s subscription ratio is

$$\frac{\sum_{l \in S_{in}} r_l * B_l}{\sum_{l \in S_{out}} B_l} = \frac{4 \times 1}{2} = 2$$

Therefore the network distance of $H_1-E_1-A_1-E_2-H_5$ is $1+2=3$. Likewise, $H_1-E_1-A_2-E_2-H_5$ has a same network distance. As a result, the network distance between H_1 and H_5 is $3 \times 0.5 + 3 \times 0.5 = 3$. The network distance between H_1 and H_{13} can be calculated similarly. There are 8 routing pathes. The subscription ratio of $l(A_1,C_1)$ is $4 \times 2/2 = 4$. Hence the network distance for each path is $1+2+4=7$. Since all paths have equal distances and equal flow distribution, the network distance between H_1 and H_{13} is also 7. When the topology is not strictly symmetric, distance computation is more complicated.

D. Optimization goal

The main goal of our algorithm presented in this paper is to place VMs onto physical machines using the network abstraction provided by the tenants, such that the network traffic is minimized and the reliability requirements are guaranteed. The cloud provider has an estimated traffic matrix M_t for every tenant t, in which $M_t(i,j)$ denotes the traffic between VMs i and j. We currently do not consider external traffic, because data center intra-traffic is much higher than Internet-facing traffic. For example, Facebook experiences "1000 times more traffic inside its data centers than it sends to and received from outside users" [2].

Notations	Explanation
t	tenant network t
$M_t(i,j)$	the traffic between VM i and VM j in tenant network t
$D(k,p)$	the network distance between position k and position p
x_{ij}	indicator that VM i is placed onto position j
$\mathbb{F}$	the set includes all functions
$\mathbb{R}$	the set includes all failure domains
S_f	a set of VMs with the same function f
P_r	a set of positions with the same failure domain r
α	reliability requirement

TABLE I
NOTATIONS IN THE OPTIMIZATION PROBLEM

The distance between each position k and position p can be presented by $D(k,p)$. What we actually want to find is the decision matrix X, in which element $x_{ip} = 1$ means that VM i is allocated to position p. The objective is to find a minimum cost allocation of VMs of all tenants into possible positions, where the cost is the sum of traffic and network distance products for all VM pairs. We formalize this problem as follows.

$$\text{minimize} \sum_t \sum_{i,j \in t} \sum_{k,p} M_t(i,j) D(k,p) x_{ik} x_{jp} \tag{1}$$

by satisfying the following constraints:

$$\sum_p x_{ip} = 1 \qquad \text{for a given VM } i \tag{2}$$

$$\sum_i x_{ip} = 1 \qquad \text{for a given position } p \tag{3}$$

$$x_{ip} \in \{0,1\} \tag{4}$$

Eq. (2) and Eq. (3) ensure that each VM is only assigned to one position, and each position only hosts one VM. If we have m VMs, n positions, $m < n$, we could just add $n-m$ dummy VMs, which do not communicate with any other VMs. The optimization result is not affected.

In addition to the above constraints, tenants may have reliability requirements, because data center network failures are very common [14]. Suppose S_f is a set of VMs of a same function $f \in \mathbb{F}$, where $\mathbb{F}$ is the set including all functions being placed. P_r is a set of positions in a same failure domain $r \in \mathbb{R}$, such as a rack, where $\mathbb{R}$ includes all failure domains. We do not want to place too many VMs in S_f to the positions in P_r for reliability purpose. Otherwise this function and hence the tenant will suffer the single-failure problem. Therefore, another set of constraints should be added to the problem formulation.

$$\forall f \in \mathbb{F}, \forall r \in \mathbb{R} \qquad \sum_{i \in S_f} \sum_{p \in P_r} x_{ip} \leq \alpha \cdot |S_f| \tag{5}$$

where α is the largest fraction of VMs in S that can be placed to one failure domain, which is given by the tenant and $\alpha \in (0,1]$.

IV. NETMAP ALGORITHM

In this section, we first show the NP-hardness of the optimization problem we want to solve. Therefore a heuristic algorithm is more practical for the cloud provider to find a good placement of VMs while reducing the traffic and satisfying the reliability requirements.

A. Complexity Analysis

Even if we do not consider the reliability constraint, minimizing the object function for a single tenant is a quadratic assignment problem (QAP) [23], which is proved to be NP-hard [21]. Furthermore, the reliability constraints such as Eq. (5) do not reduce the problem complexity, because the values of α are tenant-assigned and are arbitrary. Thus, QAP can be reduced to our optimization problem. If matrix D represents a 2-tier hierarchy tree topology, assigning VMs while optimizing the object function can be considered as the balanced minimal K-cut problem [24], which is also known to be NP-hard [23].

Since the input size of all tenants in a data center network is at the magnitude of hundreds or thousands VMs and physical machines, it is impractical to use an algorithm that finds a perfect solution.

B. Design of NETMAP

The proposed algorithm, NETMAP (tenant NETwork virtual MAchine Placement), consists of two phases when placing each tenant network. The aim of the first phase, which is called `Partition`, is to partition a tenant network into *blocks*. Each block's size is no bigger than that of a rack, since we would like to assign the block to one rack as a whole to reduce cross-rack traffic. Also, we suppose a rack is the biggest failure domain. The reliability requirements hold in each block. `Partition` aims to minimize cross-block traffic. The output of `Partition` is the input of the second phase, called `Place`. What we do in the second phase is trying to assign blocks to the physical machines while maintaining the reliability requirements. In the second phase, we try to minimize cross-pod traffic. NETMAP places tenant networks one by one in descending order of traffic volume, which is the sum of all link traffic in the tenant network. The intuition is that the cloud provider should first place those "big" tenant networks that contribute more network traffic. At the beginning, when there are sufficient network resources, the provider is likely to find good placement for a big tenant network. The "small" tenant networks with less traffic can be placed when the resources are partially occupied. Even with a relatively bad placement, a small tenant network will not incur heavy penalty to the physical network. Furthermore, we observe from the enterprise data center traffic that a tenant network with less traffic usually has smaller number of VMs. Hence the blocks with small sizes are easy to be assigned to racks as a whole.

The output of our algorithm NETMAP, shown in Algorithm 1 is the VM-to-position one-to-one mapping information, which is stored in *VMtoPo*. For each tenant network t, the function `Partition`$(VMList, M, R)$ first divides t into multiple blocks, stored in *BlockList*. The return value of function `Place`$(BlockList, VMtoPo, D, AvaRsc, R)$, *RscAlloc*, decides which part of resource should be allocated to the particular tenant network t. If *RscAlloc* is -1, it means that remaining physical machine resource is not enough to host all VMs of t, i.e. *VMList*, and the provider should stop placing t and try another one. Otherwise, *RscAlloc* is assigned a list

Algorithm 1 : NETMAP

Input: Traffic matrix M; Distance matrix D; Tenant network list $TNList$; Available physical resource $AvaRsc$; Reliability requirements R.
Output: VM-to-position mapping $VMtoPo$

$VMtoPo \leftarrow \emptyset$;
while $TNList$ is not empty **do**
  $t \leftarrow$ the tenant network with the most total traffic in $TNList$;
  $VMList \leftarrow$ the set of VMs in t;
  $TNList \leftarrow TNList - \{t\}$;
  $BlockList \leftarrow$ Partition$(VMList, M, R)$;
  $RscAlloc \leftarrow$ Place$(BlockList, VMtoPo, D, AvaRsc, R)$;
  if $RscAlloc \neq$-1 **then**
    Update$(AvaRsc, RscAlloc)$;
  else
    Mark t as unplaced;
  end if
end while

Algorithm 2 : Partition$(VMList, M, R)$

$BlockList \leftarrow \emptyset$;
$start\_node \leftarrow$ VM with most traffic in $VMList$;
$VMList \leftarrow VMList - \{start\_node\}$;
$blk_1 \leftarrow \{start\_node\}$;
$BlockList \leftarrow BlockList \bigcup \{blk_1\}$;
while $VMList$ is not empty **do**
  $v \leftarrow$ the VM in $VMList$ with most traffic with VMs in $BlockList$;
  $Valid\_blk\_set \leftarrow$ set of blocks in $BlockList$, such that putting v into any of them does not violate the resource and reliability constraints;
  if $Valid\_blk\_set \neq \emptyset$ **then**
    Put v into a block in $Valid\_blk\_set$ such that the cross-block traffic is minimized;
  else
    $New\_blk \leftarrow \{v\}$;
    $BlockList \leftarrow BlockList \bigcup \{New\_blk\}$;
  end if
end while
Return $BlockList$;

of positions which are assigned to all VMs in $BlockList$, the output of the first phase. Using Update$(AvaRsc, RscAlloc)$, the available resource is updated and reduced.

1) ***Partition phase****:* The purpose of the function Partition is to divide a tenant network into blocks and minimize cross-block traffic. It is intuitive to place VMs which communicate with each other in high traffic volume in a same block, and satisfy the reliability requirements in the meantime. We start from the VM that contributes the most traffic among all VMs in the tenant network, called the hottest VM. We put it into the first block. Then we keep selecting the VM that has the most traffic with the VMs in the first block, among all unselected VMs, and put it into the first block. Note whenever we find that the resource required for a VM is larger than the available resource of a rack, or putting the VM into the current block violates the reliability requirements, we generate a new block and put the VM into it. The process stops until all VMs are selected. An example is shown in Fig. 6 based on the enterprise data center traces. Two functions load balancer (lb) and gateway (gw) communicate a lot with each other, but VMs of a same function do not communicate. The reliability requirement is that no more than 99% VMs of a same function are placed in a same rack. Therefore at least one VM of each function should be put into an second block. The rest VMs are put into the first block to minimize cross-block traffic. The detailed pseudocode description is presented in Algorithm 2.

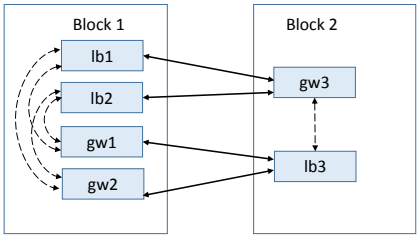


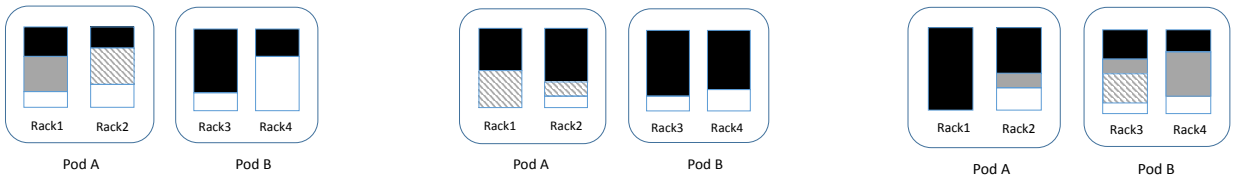
Fig. 6. Example of Partition

Algorithm 3 : Place$(BlockList, VMtoPo, D, AvaRsc, R)$

if resource for $BlockList$ is larger than $AvaRsc$ **then**
  return -1;
end if
while $BlockList$ is not empty **do**
  $B \leftarrow$ block with most intra-block traffic in $BlockList$
  $BlockList \leftarrow BlockList - \{B\}$;
  if no rack has resource more than that required by B while preserving reliability constraints **then**
    Assign B to racks with most free spaces;
    VirtualMigiration();
    Update $VMtoPo$ and $RscAlloc$;
  else
    Assign B to a rack such that the overall distance-traffic product is minimized and reliability constraints are preserved;
    Update $VMtoPo$ and $RscAlloc$;
  end if
end while
return $RscAlloc$;

The return value of the function Partition is a set of blocks, including VMs in each block. We would use this information as the input of the second phase.

2) ***Place phase****:* If the required resource of $BlockList$ is bigger than the available resource $AvaRsc$ in the cloud, there is not enough physical resource to host the tenant network. The function would return -1 to indicate that we are unable to place the tenant network. We place the blocks in descending order



(a) Normal (b) Split (c) Violation

Fig. 7. Example of `Place`. The dark area denotes the VMs from other tenant and the grey area denotes the VMs that have been placed from the same tenant. The strip area represent the block being placed. The white area denotes free spaces

of their intra-rack traffic. For each block, we assign it to the rack such that the overall distance-traffic product is minimized. If possible, a rack accommodates at most one block from a tenant network, and thus the reliability requirements are preserved, as shown in Fig. 7(a). If no rack has enough size to host the block while preserving reliability constraints, we have to split the block into sub-blocks and place them to racks with most free spaces, regardless of the reliability constrains, like the example in Fig. 7(b). Note that this process could potentially violate reliability constraints. One is example is shown in Fig. 7(c). When `place` algorithm tries to place one block, any rack is either full or contains blocks from the same tenant network, `place` algoirthm places the block to the rack with most free spaces, in order to reduce the probability of splitting but the reliability constrains may be violated. Therefore we run a function `VirtualMigration`() that "move" the placement of VMs to fix the constraints. Since VMs have not been physically placed, this process is called *virtual* migration. *NetMap achieves reliability requirements without real VM migration* and hence does not cost extra resource.

3) ***Virtual migration****:* The virtual migration algorithm is straight-forward. The cloud provider moves the part of VMs that violate the constraint and places them into a rack with free space. If no rack has free space, the provider swap the selected VMs with VMs in another rack. The algorithm continues until all reliability requirements are met. Note that virtual migration algorithm does not consume real resources since VMs have not deployed at the physical yet. The virtual migration algorithm guarantees the reliability requirements before VMs being actually placed. Due to page constraint, we skip the detailed algorithm description.

V. Performance Evaluation

A. Methodology

We analyzed the configuration and traffic data of a real multi-tenant data center network deployed at a giant telecommunication corporation in the USA carefully, based on the features of which we synthesized representative latency-sensitive enterprise data center traffic traces which include 512 VMs of 44 tenant networks. We conduct extensive experiments based on the synthetic latency-sensitive enterprise data center traffic traces and synthetic throughput-sensitive traffic traces of MapReduce applications. The network topologies are a FatTree [3] topology, one typical example of hierarchical tree, and a general hierarchical tree with oversubscription ratio at

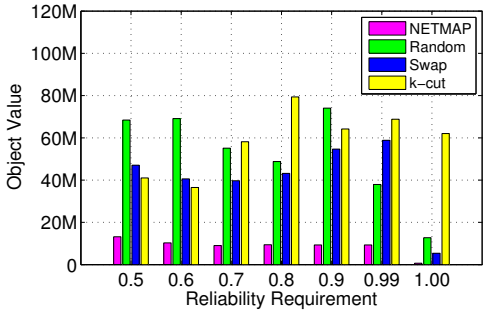


Fig. 8. Object value comparison of different algorithms

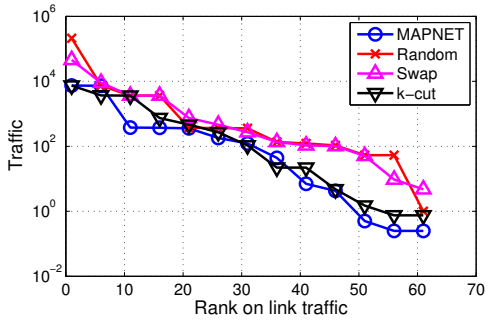


Fig. 9. Traffic distribution on links

every level. The FatTree has 4 pod, each containing 2 racks, and 4 physical machines reside in each rack. The general hierarchical tree is shown in Fig. 5. One physical machine can host 16 positions.

The traffic matrix of a tenant network as the input of an optimization algorithm is the *estimated traffic matrix* using the FAM model. However, when we evaluated the performance of various algorithms, we use the *exact traffic from traces to calculate the object value*.

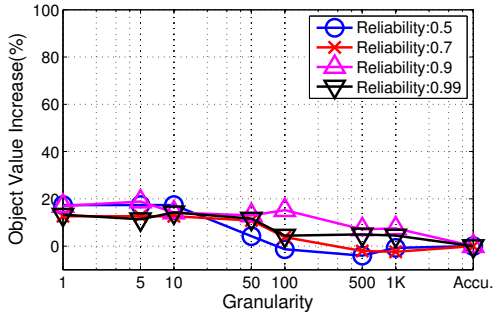
There is no existing algorithm that places VMs considering traffic while enforcing failure resistance. As a result, we compare the performance of NETMAP with the following naive algorithms. Note that virtual migration in NETMAP is used in all following algorithms to enforce reliability requirements.

Random:

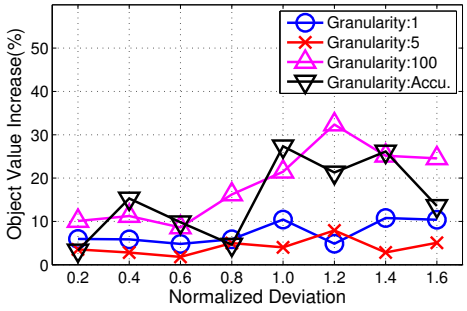
Each tenant network is allocated a set of free spaces which are close to each other. Each virtual machine is randomly assigned to a position in the allocated spaces. Then virtual migration is applied to meet the reliability requirement.

Swap:

The basic idea of this algorithm is similar to the steepest



(a) Impact of different granularity



(b) Impact of traffic variation

Fig. 10. Impact of traffic estimation granularity and deviation

decent method [12]. After assigning VMs to positions using the Random algorithm, we swap the positions of two randomly-selected VMs if the swapping yields a steepest decrease in object value. This algorithm is executed iteratively. We randomly choose 10 pairs in each iteration, and run 1000 iterations in total. Likewise, virtual migration is applied to meet the reliability requirement at last.

k-cut:

The algorithm k-cut [5] is based on the heuristic to solve the balanced minimal k-cut problem [24]. The k-cut algorithm tries to optimize all tenant networks as as a whole. Also, virtual migration is applied in the end to meet the reliability requirement.

B. Traffic cost comparison

Fig. 8 demonstrates the comparison of network cost between different algorithms in the FatTree topology. The x-axis is the reliability requirement. i.e. the maximum fraction of VMs of a same function can be placed in a rack. Specifically, $x = 1.00$ means that there is no reliability constraint, and $x = 0.99$ means that all VMs cannot be placed in a single rack. The y-axis is the object value mentioned in Eq. (1), calculated using the exact traffic from traces.

We find that NETMAP always achieves the best performance in the experiments using FatTree topology. It could also be seen that Random and Swap algorithms are slightly worse than NETMAP in optimizing the object value. Surprisingly Random has better performance than Swap. It is because virtual migration at the end of Swap significantly breaks the optimization in prior swaps. However k-cut's performance is not as good as the other three. The main reason is that k-cut does not take cross-pod traffic into consideration, which introduces heavy penalty to the objective value.

As we can see, when the reliability requirement becomes more strict (i.e., lower values), the gap between NETMAP and Swap becomes larger. When the reliability requirement is 0.5, The object value of Swap is 52.14% more than that of METMAP's. However, when the reliability requirement is 0.9, the object value of swap is only 33.8% more that that of METMAP's. The results can be explained as follows. When the reliability requirement becomes stricter, more VMs need to be virtually migrated in the Swap algorithm. More iterations of virtual migration will downgrade the prior optimization

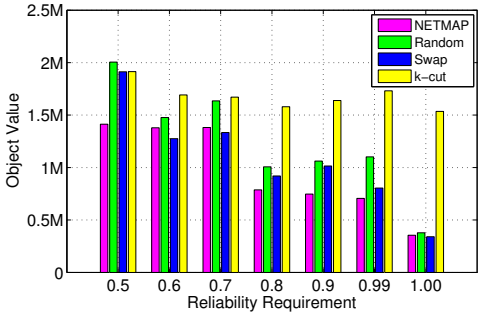


Fig. 11. Object value comparison of MapReduce experiments

results more severely. NETMAP tries to maintain reliability requirements in the first step `Partition`, alleviating the negative impact of virtual migration.

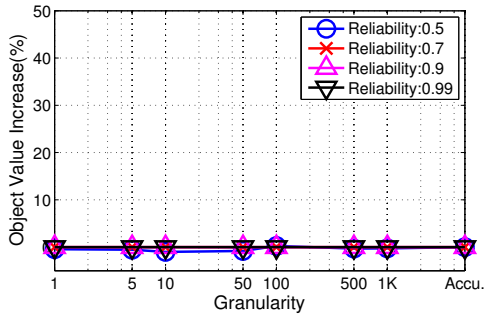
Fig. 9 shows the traffic distribution on cross-rack and cross-pod links. The y-axis is in logarithmic size. We find traffic on dominated links of random and swap is much bigger than that of the dominated links of NETMAP and k-cut. Compared to the results in Fig. 8, k-cut seems to work well in reducing traffic on links. However, the major problem of k-cut is that it does not distinguish cross-rack and cross-pod links, while the latter contribute much more penalty to the objective value shown in Fig. 8. Thus, the overall object value of k-cut is still higher than those of Random, Swap, and NETMAP.

C. Traffic matrix estimation

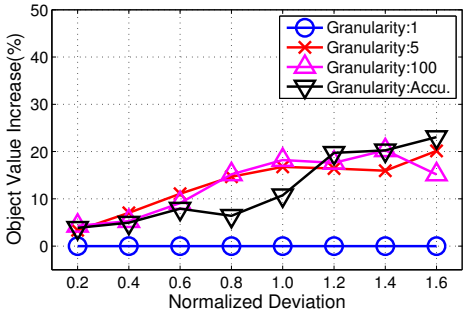
One input of our heuristic algorithm is the traffic matrix which is estimated based on the FAM model rather than using exact values. We will use experiments to show that even using estimated traffic matrix, NETMAP still yields relatively good results.

There are two factors that make the estimated traffic matrix different from accurate traffic matrix. One is *granularity* in an integer value. For example, if the granularity is 5, then only numbers from 0 to 5 are used to indicate the different volume level of traffic between a pair of VMs. The other one is traffic *deviation*. We *artificially add deviation* to the traffic between two VMs, normalized by the average traffic of communication between the same pair of functions.

Our experiment shows that the coarse-grained traffic matrix does not degrade the performance significantly. In Fig. 10(a),



(a) Impact of different granularity



(b) Impact of traffic variation

Fig. 12. Impact of traffic estimation of MapReduce experiments

TABLE II
OBJECT VALUE DECREASE WHEN DOUBLING THE NETWORK SIZE

Reliability Requirement	0.5	0.6	0.7	0.8	0.9	0.99
Object Value Decrease(%)	44.35	9.16	0.01	1.98	0	0

the *y*-axis is the object value increase of VM placement using estimated traffic matrix, compared to that using accurate traffic matrix, the *x*-axis is different degrees of granularity. Here Accu. means accurate traffic matrix. Different curves show different levels of reliability requirements. We find that even using coarse-grained traffic matrix, the object value increase is below 20% in all cases. We further observe that the object value increase less with the growth of granularity. However, levels of reliability requirements seem to have little impact to object value increase.

Fig. 10(b) shows the impact of traffic deviation when the reliability requirement is fixed to 0.5. The *x*-axis shows different normalized deviation values. Different curves represent different granularity levels. We find that when deviation is small (between 0 and 1), using estimated traffic matrix still gets similar results. We also find that high-deviated traffic has less impact on coarse-grained traffic matrix. It is mainly because coarse-grained values are not easy to be affected by deviations.

D. MapReduce Experiments

For this set of experiments, we simulate tenant networks running MapReduce applications. Each tenant network includes 5 mappers, 5 reducers, and a GFS of 20 chunkservers. Fig. 11 demonstrates the performance of different algorithms. We can see that the performance gap between each algorithm becomes smaller, because traffic in different tenants are even. NETMAP works well for both skewed and even traffic data.

Fig. 12 shows the impact of traffic estimation on MapReduce tenant networks. We find that the object value increase also remains at a very low level by varying both granularity and deviation.

E. Impact of network topologies

The underlying topology also influences the performance of VM placement. Fig. 13 shows the performance of different

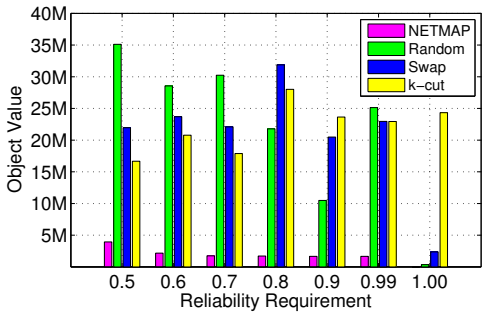


Fig. 13. Object value comparison on a general hierarchical tree topology

algorithms when the network topology is a general hierarchical tree shown in Fig. 5, which has higher level of oversubscription than a FatTree. We find that our algorithm outperforms others more significantly compared to FatTree experiments. It is because in the hierarchy tree, the cost on cross-pod link is more expensive due to multiple levels of oversubscription. Our algorithm, NETMAP, deliberately avoids cross-pod traffic, hence it outperforms others.

The size of the underlying topology also has an impact on the object value. If the size of the physical network grows, the object value is expected to decrease. In previous experiments, the number of VMs is the same as the available positions in the physical network. We double the size of physical positions. Table II shows the object value decrease in different reliability requirements. We could see that if the reliability is strict, say 0.5, the object value decreases significantly by 44.35% when the physical network size increases. When the reliability requirements are loose, doubling the network size does not make a big impact.

F. Computation time

Another advantage of our algorithm is that it is fast. Table III and Table IV show the comparison between different algorithms on the VM placement computation time of the enterprise traces and MapReduce traces respectively. NETMAP's computation is the fastest in both circumstances.

TABLE III
COMPUTATION TIME FOR ENTERPRISE TRAFFIC

Algorithm	Calculating Time(s)						
	Reliability Requirement						
	0.5	0.6	0.7	0.8	0.9	0.99	1.0
NETMAP	0.068	0.068	0.067	0.065	0.065	0.065	0.062
Swap	8.24	4.82	3.76	3.61	3.54	3.43	3.40
Random	14.7	6.87	0.036	0.426	0.074	0.091	0.011
k-cut	19.8	14.5	11.7	11.7	12.0	11.9	11.7

TABLE IV
COMPUTATION TIME FOR MAPREDUCE TRAFFIC

Algorithm	Calculating Time(s)						
	Reliability Requirement						
	0.5	0.6	0.7	0.8	0.9	0.99	1.0
NETMAP	0.121	0.120	0.167	0.120	0.240	0.246	0.227
Swap	309	426	477	460	418	448	410
Random	0.563	0.062	0.066	0.124	0.122	0.122	0.117
k-cut	23.7	25.8	22.3	35.0	55.5	53.9	48.3

VI. RELATED WORK

The increasing prominence of multi-tenant IaaS clouds has prompted research interests in virtual machine placement. TMVPP [23] is the first work to consider network factors when placing VMs. In the paper, the authors formally defined the Traffic-aware VM Placement Problem (TVMPP) as an optimization problem. In addition to network traffic, [5] takes fault tolerance into account, in which network cost and fault tolerance are jointly optimized; yet, in contrast to our work, the fault tolerance constraint is not strictly enforced. SecondNet [17] reserves link bandwidth when placing VMs to provide pairwise bandwidth guarantee. For all these works, a complete pairwise traffic matrix is required.

Other than traffic matrix, various network abstraction models are proposed for tenants to more efficiently express their network requirements. Duffield et al. introduced the hose model [10] for wide-area VPNs. Virtual-cluster [4] emulates a two-tier physical network, so that tenants can get predictable environments in shared clouds. TIVC [25] further models the time-varying nature of the networking requirement of cloud applications. Lee et al. [19] proposed a similar abstraction to our FAM, but their abstraction ignores the traffic characteristics between functions.

Recently there have been a great number of proposals for data center network topologies, including Fattree [3], VL2 [15] and BCube [16]. Fattree [3] is a hierarchical multi-rooted tree topology can be built with commodity switches to provide multiple equal-cost paths between any pair of servers with its core tier and aggregation tier forming a Clos [8] topology.

VII. CONCLUSION

In this work we analyze the traffic data of a real multi-tenant cloud, based on which we propose a new tenant network abstraction model FAM and a new network distance model for hierarchical data center networks. We design a novel VM placement algorithm, NETMAP to reduce the traffic of the cloud provider's data center network and preserve the reliability requirements from tenants. Trace-driven simulations show that NETMAP can effectively optimize the network traffic under various circumstances for both latency-sensitive and throughput-sensitive traffic data.

REFERENCES

[1] Apache hive. *http://hive.apache.org/*.

[2] Facebook future-proofs data center with revamped network. http://www.wired.com/2012/06/facebook-nc-data-center/all/.

[3] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. In *Proc. of ACM SIGCOMM*, 2008.

[4] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron. Towards predictable datacenter networks. In *Proc. of SIGCOMM*, 2011.

[5] P. Bodík, I. Menache, P. Mani, D. Maltz, M. Chowdhury, and I. Stoica. Surviving failures in bandwidth-constrained datacenters. In *Proc. of ACM SIGCOMM*, 2012.

[6] M. Chowdhury and I. Stoica. Coflow: a networking abstraction for cluster applications. In *Proc. of the 11th ACM Workshop on Hot Topics in Networks*, 2012.

[7] W. Cui and C. Qian. Difs: Distributed flow scheduling for adaptive routing in hierarchical data center networks. In *Proc. of IEEE ANCS*, 2014.

[8] W. Dally and B. Towles. Principles and practices of interconnection networks. 2004.

[9] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 2008.

[10] N. Duffield, P. Goyal, A. Greenberg, P. Mishra, K. Ramakrishnan, and J. van der Merive. A flexible model for resource management in virtual private networks. *ACM SIGCOMM Computer Communication Review*, 1999.

[11] N. Farrington and A. Andreyev. Facebook's data center network architecture. In *Proc. of IEEE Optical Interconnects*, 2013.

[12] J. Fliege and B. Svaiter. Steepest descent methods for multicriteria optimization. *Mathematical Methods of Operations Research*, 2000.

[13] S. Ghemawat, H. Gobioff, and S.-T. Leung. The google file system. In *ACM SIGOPS Operating Systems Review*, 2003.

[14] P. Gill, N. Jain, and N. Nagappan. Understanding network failures in data centers: Measurement, analysis, and implications. In *Proc. of ACM SIGCOMM*, 2011.

[15] 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 *Proceedings of ACM SIGCOMM*, 2009.

[16] C. Guo et al. Bcube: a high performance, server-centric network architecture for modular data centers. In *Proc. of ACM SIGCOMM*, 2009.

[17] C. Guo, G. Lu, H. Wang, S. Yang, C. Kong, P. Sun, W. Wu, and Y. Zhang. Secondnet: a data center network virtualization architecture with bandwidth guarantees. In *Proc. of USENIX CoNext*, 2010.

[18] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. *ACM SIGOPS Operating Systems Review*, 2007.

[19] J. Lee, Y. Turner, M. Lee, L. Popa, and S. Banerjee. Application-driven bandwidth guarantees in datacenters. In *Proc. of ACM SIGCOMM*, 2014.

[20] X. Li and C. Qian. Low-complexity multi-resource packet scheduling for network function virtualization. In *Proc. of IEEE INFOCOM*, 2015.

[21] E. Loiola, N. Abreu, P. Boaventura-Netto, P. Hahn, and T. Querido. A survey for the quadratic assignment problem. *European Journal of Operational Research*, 2007.

[22] G. Malewicz, M. Austern, A. Bik, J. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. Pregel: a system for large-scale graph processing. In *Proc. of the 2010 ACM SIGMOD*.

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

[24] H. Saran and V. Vaziran. Finding k cuts within twice the optimal. *SIAM Journal on Computing*, 1995.

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

[26] Y. Yu and C. Qian. Space shuffle: A scalable, flexible, and high-bandwidth data center network. In *Proc. of IEEE ICNP*, 2014.