Virtual Machines Placement for Network Isolation in Clouds

Gyuyeong Kim, Hoorin Park, Jieun Yu, and Wonjun Lee
Department of Computer Science and Engineering Korea University
WCU Future Network Optimization Technology Center (FNOT)
Seoul, Korea
wlee@korea.ac.kr

ABSTRACT

Traffic-aware Virtual Machines (VMs) placement is a key feature for cloud performance isolation. However, variability hinders a data center to provide network isolation. The existing proposals do not give an answer to a question that how we can obtain resource demand of tenants. To address it, we propose a VMs placement mechanism, which measures resource demand in a server cluster, *Incubator* before place VMs into a data center. By the stable matching algorithm, we consider resource availability of a data center. We show that our approach decreases cloud oversubscription, and increases link utilization through an intensive simulation.

Categories and Subject Descriptors

C.2.3 [**Computer-Communication Networks**]: Network Operations

General Terms

Algorithms, Performance

Keywords

Data Center Networks, VM Placement, Network Isolation

1. INTRODUCTION

Infrastructure as a Service (IaaS) provides a virtualized work space to tenants. Tenants can own a number of VMs, and shares resources on a server with the other tenants. The most significant motivation for using IaaS is a cloud data center catches both cost saving and performance guarantee.

As tenants desire to maintain their services without performance interference, providing performance isolation is significant [8]. In contrast with system resources like CPU and memory, a cloud provider manages network resources by traditional best-effort manner due to the variability. However, providing network isolation is desirable to both providers and tenants. Because lasting oversubscription hinders meeting performance guarantee. Tenants may hesitate to use a cloud data center in spite of cost saving in a such case. For providers, a deserted cloud data center cannot earn high revenue.

Recently, efforts to provide network isolation by VMs placement has emerged. Wang *et al.* [11] address the issue by the bin packing algorithm. Breitgand *et al.* [3] extend the work. [3] not only reduce oversubscription probability but also minimize the number of bins, physical servers where VMs are consolidated. However, Prior work assume that overflow probability p, which is determined randomly, nevertheless a method to obtain resource demand is a key consideration for initial traffic-aware VMs placement. Furthermore, their simulation results show that the algorithm only reduces the number of bins, not oversubscription.

In this paper, we design a traffic-aware VMs placement mechanism based on practical traffic measurement. We measure traffic distribution in a server cluster named *Incubator* and place VMs into servers by the stable matching algorithm. We named a series of process as the *Incubating*. Our key contributions are as follows.

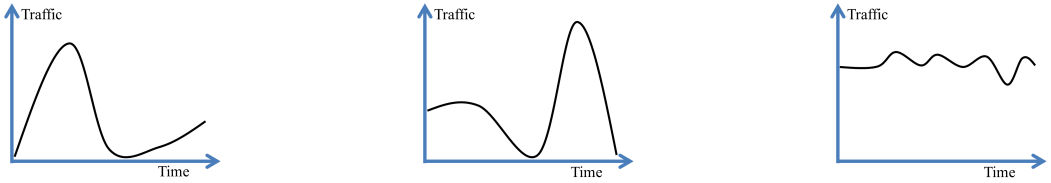
- Prior work assume that traffic demand is exposed or oversubscription occurs randomly, but we measure real traffic demands of tenants.
- By designing a time-variant mechanism, we catch both oversubscription reduction and high utilization.

2. RELATED WORK

We categorize prior work into two types. One is bandwidth guarantee approach [6, 8, 9, 10] that meet network isolation by a bandwidth allocation policy. The other is VMs placement approach [3, 11] that mitigate oversubscription by traffic-aware VMs placement.

Bandwidth guarantee: Seawall [10] is a fair bandwidth sharing policy among VM-pairs. It enforces fairness by ECN feedback mechanism. However, Seawall has a limitation that a tenant can obtain more bandwidth by increasing the number of destinations. Secondnet [6] is a bandwidth provisioning policy. In contrast with Seawall, Secondnet guarantees bandwidth for not each VM but tenant. However, Secondnet only provides network isolation among VMs under a single tenant. Gatekeeper [9] is an ECN feedback based mechanism, which is similar to Seawall. However, Gatekeeper provides per-VM isolation. It does not distinguish the source from the destination. The aforementioned approaches fall short when average aggregate bandwidth demand exceeds a link capacity. Our VMs placement mechanism avoids over-

RACS'12 October 23-26, 2012, San Antonio, TX, USA.



(a) Daily traffic distribution of VM 1 (b) Daily traffic distribution of VM 2 (c) Daily traffic distribution of VM 1 + VM 2

Figure 1: Abstracted motivation illustration

subscription by initial placement phases considering bandwidth demand.

VMs placement: Wang *et al.* [11] model the issue as a stochastic bin packing (SBP) problem. Breitgand *et al.* [3] improve the performance of Wang's work. However, [11] does not explain how they can obtain bandwidth demand. [3] abstracts oversubscription factors as an overflow probability p. We doubt that whether a single value p can truly represents daily traffic distribution. Secondly, the result of the aforementioned work shows that their algorithm is only efficient to minimize the number of servers. Although [3] argue that oversubscription reduction performance is still enough in present public clouds, an evidence is anecdotal. Thus, we design a mechanism that surely reduces oversubscription based on practical traffic measurement.

3. OUR APPROACH AND CHALLENGES

[2, 4] show that the traffic distribution in data centers follows Wei-bull and Poisson probability distribution over time. The wider the range of observations is (in terms of weeks, months, and years), the clearer the result is. Particularly, although demands of tenants increase as services grow, the peak time and the off-peak time are invariant. Therefore, our basic idea is to place VMs having opposite traffic distributions in the same server, which leads to reduce oversubscription of network resources and increase link utilization.

The motivation is illustrated in Figure 1. A service related to banking and stock trading shows the traffic distribution of Figure 1 (a) where there is heavy traffic usually in the morning and afternoon. On the other hand, the traffic of online games can have the distribution of Figure 1 (b). The two kinds of services have different times of peak demand and therefore, if VMs of the two services are placed together in the same server, we can obtain the results of Figure 1 (c). The mixed placement alleviates interference among the different kinds of services and helps the data center to have network performance isolation.

For this approach, we must solve the following questions.

- How can we get the traffic distribution and demand of each tenant?
- Given the traffic distribution, how can we allocate VMs to servers?

For the first question, existing work [3, 11] is based on statistical modeling to expect the demand for network resources. However, instead of expectation, we decide to measure the demand over a period of time because measurement is more accurate.

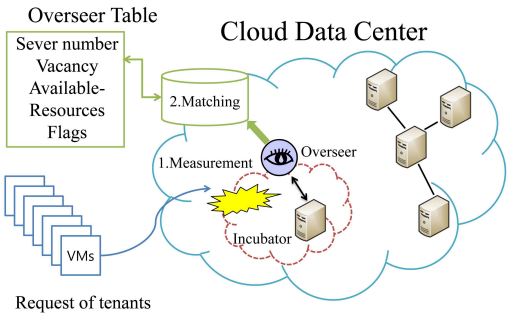


Figure 2: Overview of *Incubating*

To solve the second question, we adopt the variation of the Stable Marriage Problem mentioned in game theory. In Stable Marriage Problem [7], given n men and n women, who have different preference values for the member of the opposite sex, according to their preference value. The job is to determine an assignment where each man is married to one woman. Our matching algorithm assigns VMs to proper servers. Detailed methods are described in section 4 and 5.

4. ARCHITECTURE

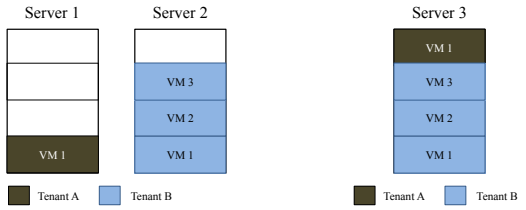
Two parts comprise the architecture of *Incubating*. *Incubator* is the place where VMs' traffic information is measured. Traffic measurement is performed by a tenant unit to avoid traffic blending. Overseer obtains traffic distribution from *Incubator*, and performs the stable matching algorithm. For reliability, Overseer has reserves that replace his job during failure.

4.1 Incubator

Incubator is a server cluster to measure traffic distribution of tenants. As illustrated in Figure 2, *Incubator* is not an independent server cluster, but a subset of a data center. In detail, we bind vacant servers as a cluster. Hence, it is cost-efficient because a provider does not have to establish extra servers for *Incubator*.

Incubator measures traffic as a tenant unit to avoid traffic blending with the other tenants. We define 'one server, one tenant' principle that only VMs of a single tenant can be consolidated in a server of *Incubator*. For example, in Figure 3 (a), a VM of tenant A is placed in server 1. In server 2, three VMs of tenant B are consolidated. However, in Figure 3 (b), as all VMs are placed after *Incubating*, four VMs are consolidated in server 3.

The size of *Incubator* depends on input size change. If



(a) 'one server, one tenant' principle in *Incubator* (b) Consolidation in data center

Figure 3: Difference of consolidation method between *Incubator* and data center

we set the size extremely large, some servers of *Incubator* become 'no man's land' that means it will not be utilized either for measurement or running services. Furthermore, it causes deficiency of servers where VMs will be placed. Reversely, if the size is immoderately small, requested VMs cannot be placed into *Incubator* timely.

To avoid the issue, we design a skinny and input adaptive size adjustment manner as follows. Let R be a set of VMs that a tenant requests. Let n be the maximum number of VMs which can be consolidated in a server. n is determined by system resource status. However, we assume that n is the number of CPU cores in a server for simplicity. Note that our focus is network resources, not system resources. Hence, maximum four VMs can be consolidated in a server by the assumption. The initial size of *Incubator* starts from zero. If the number of requested VMs is 6, *Incubator* size increases to two by (1). Therefore, a server accommodates four VMs and the other one takes remained two VMs. Thus, *Incubator* checks 'no man's land' and always maintains his skinny size.

$$|u| = \begin{cases} \left\lfloor \frac{|R|}{n} \right\rfloor + 1 & (|R| \bmod n \neq 0) \\ \frac{|R|}{n} & (|R| \bmod n = 0) \end{cases} \quad (1)$$

4.2 Overseer

Overseer is an independent server that manages *Incubating* as illustrated in Figure 2. It performs matching algorithm to place VMs into a data center after gathering traffic information. Overseer table stores the information, and Overseer updates his table every iteration of the matching algorithm to maintain the latest information. In Overseer table, server number indicates unique identification number of a server. Vacancy row stores the information of available system resources, and Overseer refers this row when he performs matching algorithm to place VMs. Available resources row stores remained network resources as an array. Lastly, Flag row is expressed in a boolean. The status of a server is determined by the value 'True/False'.

To provide reliability, we design Overseer as a master-reserves architecture as illustrated in Figure 4. Master server performs Overseer's job, and reserves are always on standby. Master delivers an alive packet per common data centers packet interval 1,000 μs [2] to reserves. After that, reserves update their time stamp. If the timestamp is not updated over 3,000 μs after the last update, a reserve checks the status of master and if the master failed, the reserve that detected master's fail first succeeds master's job. After that,

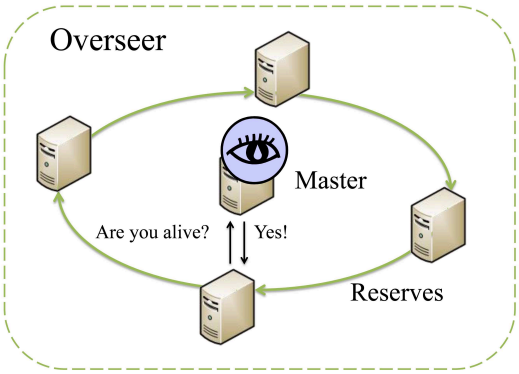


Figure 4: Master-reserves architecture of Overseer

the successor inform reserves that succession is done.

	a_{s_1}	a_{s_2}	...	a_{s_m}
d_{v_1}	21	11	⋯	31
d_{v_2}	55	24	⋯	62
⋮	12	17	⋯	21
d_{v_n}	16	52	⋯	55

(a) Unranked Proposer Table

	d_{v_1}	d_{v_2}	...	d_{v_n}
a_{s_1}	56	72	⋯	34
a_{s_2}	23	32	⋯	48
⋮	73	62	⋯	61
a_{s_m}	16	24	⋯	11

(b) Unranked Accepter Table

V_1	S_{21}	S_{59}	⋯	S_{183}
V_2	S_{330}	S_2	⋯	S_{21}
⋮			⋯	
V_n			⋯	

(c) Ranked Proposer Table

S_1	V_6	V_2	⋯	V_1
S_2	V_1	V_4	⋯	V_3
⋮			⋯	
S_m			⋯	

(d) Ranked Accepter Table

Figure 5: Preference Table

5. ALGORITHM

In this section, we present how *Incubating* works. Algorithm can be divided into two parts. One is measurement process of VMs' traffic distribution and demands. The other part is performing matching algorithm to place VMs into a data center.

5.1 Traffic Measurement

As illustrated in Figure 2, when a tenant requests VMs, Overseer checks whether *Incubator* can accommodate requested VMs, and adjusts the size of *Incubator* as we described in section 4. Lastly, Overseer place VMs into *Incubator* based on 'one server, one tenant' principle.

Traffic measurement is performed for 30 days which is an enough period to obtain the average traffic distribution of VMs. Let U be a set of servers that comprise *Incubator*. Each $b \in B$ is a traffic vector of a server u. Each $u \in U$ updates its accumulated traffic information Ab^t_{n-2} with new information b^t_n for every single day. Thus, traffic vector b, can

Algorithm 1 Server selection algorithm

while there is an unpaired VM **do**
  select an unpaired VM v and the first server s on hit ranked preference table
  remove s from his table so it will not be selected again
  if s is already matched **then**
    if s prefers v than her current partner v' **then**
      set $\mu(v, s)$ as matched
      set $\mu(v', s)$ as unmatched so now v' is unpaired
    else
      v is still unpaired because s is happier with v'
    end if
  else
    the server was not paired yet so accept immediately, $\mu(v, s)$, as married
  end if
end while

maintains the latest information $Ab_n^t = \{Ab_n^1, Ab_n^2, ..., Ab_n^t\}$ as formulated in (2). Each $t \in T$ indicates time of day. avg () in (2) is a function that calculates the arithmetic mean of accumulated traffic and new one.

$$Ab_n^t = avg(Ab_{n-2}^t, b_n^t) \quad (2)$$

After measurement, *Incubator* servers obtain their own average daily traffic distribution information as illustrated in Figure 2. As VMs of a tenant which are consolidated in a server of *Incubator* run similar services, they are managed as a chunk. Therefore, although the number of VMs consolidated in a server is at least larger than one, only one traffic information can be obtained. We can think the case that the lifetime of VMs is under 30 days such as Amazon EC2 Spot Instance [1]. In that case, as measurement result is meaningless, the data are discarded and matching algorithm will never be performed. Overseer now gathers information from *Incubator*, and he performs matching algorithm to place VMs in data center.

5.2 Matching Algorithm

Our algorithm for placement into data center is a variation of stable matching algorithm [7] that solves stable marriage problem. In our algorithm, the details are almost same except assumption that n proposers and m accepters exist in contrast to the original assumption that the number of proposers and accepters is same. We assume VMs, will be placed into data center, as a proposer that a man who proposes a woman. Servers in data center are assumed for a woman accepter who determines whether accepts a proposal or not. All VMs and servers have preference for each. VM prefers a server who has higher value when his demands subtract from available resources of a server. A server prefers a VM who demands less resource. This setting provides flexibility when excepted network trouble which is caused by a burst error.

Matching has two steps. Step 1 is the process of preference calculation for each other. Step 2 is to place VMs into the most appropriate server where can reduce the occurrence probability of network oversubscription. We now present these steps as follows.

Step 1: *Calculation of preferences* Let $V = \{v_1, ..., v_n\}$ be a set of VMs, and $S = \{s_1, ..., s_m\}$ be a set of servers. Each $v \in V$ and $s \in S$ has an ordered list of his (VMs) or her

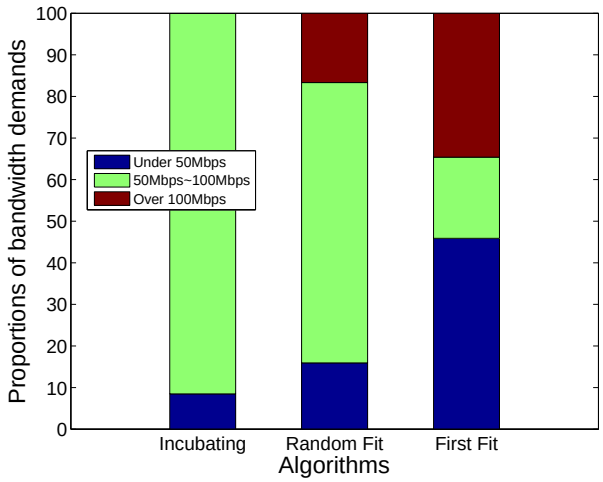


Figure 7: Stacked column chart of servers' traffic demands

(servers) preference list. The list is a temporary list which is perished after matching. A set $A = \{a_{s_1}, ..., a_{s_m}\}$ indicates free resources of each server, and $D = \{d_{v_1}, ..., d_{v_n}\}$ is a set of demands of each VM. Elements of set A and D are not a single value but an array since it stores resources information by discrete time unit. A ith preference of each v to s, P_{v_i,s_i} is calculated as we described in equation (3). We used maximum value of demands since it provides a simplification of problem solving. A ith preference of each s to v, P_{s_i,v_i} is simply determined by following equation (4). Function Δ means the difference between two parameters. The result constructs an unranked table as in Figure 5 (a) and (b). It is sorted in descending order after calculation and converted to complete ranked preference table of VMs/Servers to opposite ones. We showed them in Figure 5 (c) and (d).

$$P_{v_i,s_i} = max(\Delta_{a \in A, d \in D}(a_{s_i}, d_{v_i})) \quad (3)$$

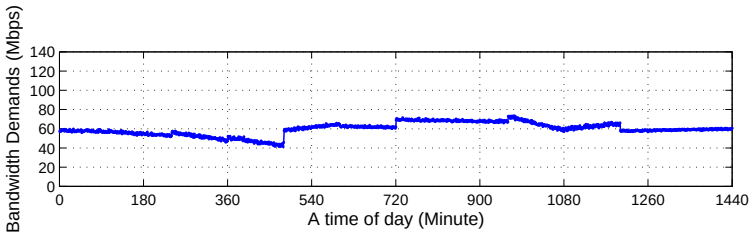
$$P_{s_i,v_i} = min_{d \in D}(d_{v_i}) \quad (4)$$

Step 2: *Selection the most appropriate server* Let μ be a matching between a VM and a server. A pair (v, s) is a matching pair if both v and s prefer being under μ. VMs are proposer and servers are accepter. Therefore, in each iteration, an unmatched VM proposes to the first server on his preference list whom he has not proposed to yet. A server who received a proposal accepts a new proposal $\mu(v', s)$ and rejects her current assignment $\mu(v, s)$ if she prefers a new proposal. Matching is repeated until all VMs are matched. As a result, VMs optimal matching is performed, and VMs are migrated to servers outside *Incubator*. We describe the step in algorithm 1.

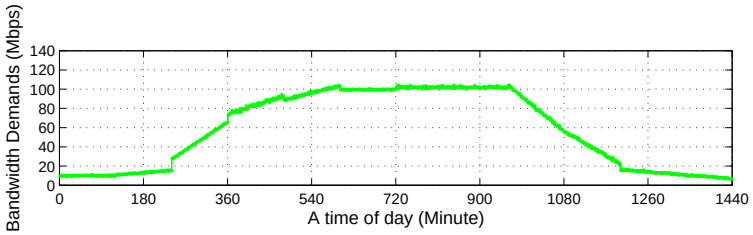
6. EVALUATION

We evaluate performance of *Incubating* comparing with the other two algorithms, Random Fit (RF) and First Fit (FF) [3] and present that our approach has succeeded to reduce oversubscriptions through traffic distribution of an arbitrary server in data center.

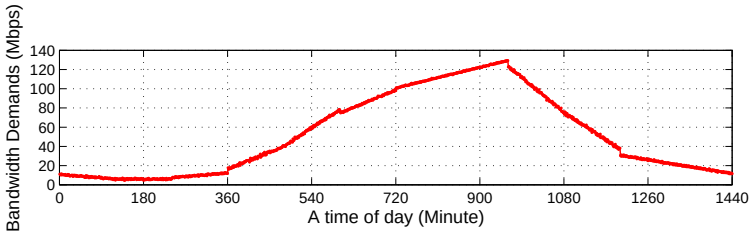
6.1 Simulation Setup



(a) Incubating



(b) Random Fit



(c) First Fit

Figure 6: Traffic distribution of an arbitrary server

We compare our approach with FF, the simplest bin packing algorithm, and RF which is the naïve placement algorithm in present data center [5]. We cannot show directly compared the results between ours and [3]'s. Because they abstracts traffic as a single value overflow probability p. On the other hand, we consider traffic distribution with the passage of time. We generate traffic following Poisson distribution, by a minute. They also never give how we can obtain an overflow probability p. Lastly, their main focus is to minimize the number of servers where VMs are consolidated, not reducing oversubscription. Thus, they focus on showing minimization, not reducing oversubscription.

We set the number of servers as 10,000, and a request of tenants occurs per 20 minutes. Each request can contain maximum 20 VMs, and it follows cumulative probability distribution having 70% of requests demand below five VMs. Lastly, we categorize services into three types, 'Day, Night, Random' which are motivated from kinds of stock market and online games server.

6.2 Result Analysis

We select an arbitrary server which runs same services in a data center to present the result. Figure 6 shows the simulation result for each algorithm. We observe that traffic demands of both FF and RF exceeded over 100Mbps, a link capacity. FF causes relatively short and intensive oversubscription, and RF records long and moderate oversubscription. Especially, as FF records oversubscription exceeded over 120Mbps, the services undergo performance interference.

Our approach shows an extremely smooth traffic distribution since VMs having contrasted distribution are consolidated as we intend in Figure 1. Each server may have dissimilar distribution, however, in general, a server which VMs are placed by *Incubating* shows that the highest demand is always under 100 Mbps. Thus, we present that our approach is efficient to reduce oversubscription by detail example of an arbitrary server in data center.

We now demonstrate whether our approach is valid in all servers used for simulation. Figure 7 is the comparison result of each algorithm's demands proportion of the moment when traffic demand is the highest in a day among servers. 20% and 35% of servers show oversubscriptions for RF and FF, respectively. Adversely, the case does not occur in *Incubating*. Thus, we can know that our proposal is valid in all servers in data center. Lastly, in Figure 6 and 7, most demands for *Incubating* has a range that larger than 50Mbps and less than 100Mbps. From the result, we can obtain one more significant observation that ours not only reduces oversubscription but also increases link utilization.

7. CONCLUSION AND FUTURE WORK

In this paper, we address network isolation issue in cloud data centers. We proposed a traffic-aware VMs placement mechanism, not only reduces oversubscriptions but also increases link utilization. Our approach obtain initial traffic

demand in a server cluster, and place VMs by the stable matching algorithm. For the future work, we will focus on joint VMs placement considering both system and network resources.

8. ACKNOWLEDGMENTS

This research was jointly sponsored by MEST, Korea under WCU (R33-2008-000-10044-0), MEST, Korea under Basic Science Research Program (2011-0012216), MKE/KEIT, Korea under the IT R&D program [KI001810041244, SmartTV 2.0 Software Platform], and MKE, Korea under ITRC NIPA-2012-(H0301-12-3001) and NRF of Korea under the project of Global Ph.D. Fellowship conducted from 2012. Wonjun Lee is the corresponding author.

9. REFERENCES

[1] Amazon ec2 spot instances. http://aws.amazon.com/ko/ec2/spot-instances/.

[2] T. Benson, A. Akella, and D. A. Maltz. Network traffic characteristics of data centers in the wild. In *Proc. of the 10th annual conference on Internet measurement*, (IMC '10), 2010.

[3] D. Breitgand and A. Epstein. Improving consolidation of virtual machines with risk-aware bandwidth oversubscription in compute clouds. In *Proc. INFOCOM*, 2012.

[4] B. Chandrasekaran. Survey of network traffic models. http://www.cse.wustl.edu/ jain/cse567-06/ftp/trafficmodels3.pdf/.

[5] D. Erickson, B. Heller, S. Yang, J. Chu, J. Ellithorpe, S. Whyte, S. Stuart, N. McKeown, G. Parulkar, and M. Rosenblum. Optimizing a virtualized data center. *SIGCOMM Comput. Commun. Rev.*, 41(4):478–479, Aug. 2011.

[6] C. Guo, G. Lu, H. J. 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 the 6th International COnference*, (Co-NEXT '10), 2010.

[7] W. Hunt. The stable marriage problem. http://www.csee.wvu.edu/ ks-mani/courses/fa01/random/lecnotes/lecture5.pdf/.

[8] L. Popa, A. Krishnamurthy, S. Ratnasamy, and I. Stoica. Faircloud: sharing the network in cloud computing. In *Proc. of the 10th ACM Workshop on Hot Topics in Networks*, (HotNets '11), 2011.

[9] H. Rodrigues, J. Renato Santos, Y. Turner, P. Soares, and D. Guedes. Gatekeeper: Supporting bandwidth guarantees for multi-tenant datacenter networks. In *Proc. of the 3rd Workshop on I/O Virtualization*, (WIOV'11), 2011.

[10] A. Shieh, S. Kandula, A. Greenberg, C. Kim, and B. Saha. Sharing the data center network. In *Proc. of the 8th USENIX conference on Networked systems design and implementation*, (NSDI'11), 2011.

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