A CPU Overhead-aware VM Placement Algorithm for Network Bandwidth Guarantee in Virtualized Data Centers

Kwonyong Lee and Sungyong Park
Department of Computer Science and Engineering
Sogang University
Seoul, Republic of Korea
Email: kwonyong82@gmail.com, parksy@sogang.ac.kr

***Abstract*—As server consolidations based on the virtualization techniques become popular and cloud services continue to grow rapidly, more and more data centers are being built to accommodate a number of virtual clusters running various workloads. Since these virtual clusters often share the resources provided by physical machines (PMs), it is more likely that the interferences between virtual machines (VMs) affect the performance of applications running on top of the virtual clusters. While a lot of studies have proposed different virtual machine placement algorithms to investigate this issue, the problem caused by network performance variability still remains as a challenging issue. Since they usually ignore the CPU overhead to process the communications between VMs, the network bandwidth allocated to a VM cannot be fully utilized when a PM has not enough CPU resources to cover the CPU overhead for VM networking functions. This results in unpredictable application performance running on the virtual clusters. This paper proposes a virtual machine placement algorithm that considers the CPU overhead incurred to reserve network bandwidth in a virtualized data center environment. In order to decide the CPU overhead necessary to guarantee the network bandwidth allocated to a VM, a performance model based on standard linear regression using the data collected from a real environment is used. By comparing the amount of CPU resource available in the driver domain with the CPU overhead obtained from the performance model, the proposed algorithm decides whether the network bandwidth for the VM can be provided or not and selects an appropriate location for VM placement. The benchmarking results show that the proposed algorithm guarantees the network bandwidth allocated to each VM without violations when the CPU resources are shared by multiple VMs.**

***Keywords*—virtual machine placement; virtual machine networking; network bandwidth reservation; virtualized data center**

I. Introduction

With the proliferation of server virtualization technologies, today's data centers are increasingly adopting these techniques to build virtualized data centers. In a virtualized data center, service providers create or destroy virtual clusters on demand, and run various multi-tenant applications with different workloads. The virtual cluster consists of multiple virtual machines (VMs) that can be hosted on multiple physical machines (PMs), and shares the virtualized resources provided by the PMs. Although the virtual clusters can be easily created or destroyed by the provider, they may suffer from the interferences between VMs due to the fact that virtualized data centers generally over-subscribe a variety of resources to increase resource utilizations and reduce costs. For example, the OpenStack, which is one of the most famous open source platforms to build virtualized data centers, allows sixteen virtual CPUs (VCPUs) to run on one physical CPU (PCPU) [1]. The network topologies are also typically over-subscribed by a factor of 5:1 to 20:1 at Top-of-Rack (ToR) switches and up to 240:1 when the paths cross through the highest layer of network topology tree [2]. It is also reported that the network architecture in Facebook's data center is over-subscribed by 40:1 in practice [3].

Therefore, the performance of applications running on top of the virtual clusters is often unstable at runtime compared to the case where they are run on physical clusters that utilize dedicated resources provided by the PMs. Considering that the cost of renting VMs from the virtualized data centers is usually determined by the total execution times of applications, the number of VMs used, and the amount of resources allocated per VM, each tenant is likely to expect satisfiable performance in return for the expenditure. However, the unstable (or unpredictable) performance incurred by using shared resources make it difficult to predict the execution times of applications, and sometimes causes time-critical applications not to finish their execution on time.

In practice, the virtualized data centers have utilized various capacity planning tools to decide an appropriate location for VM placement such as VMware Capacity Planner [4] and IBM CloudBurst [5]. Since these tools and their relevant researches [6], [7] only consider the local resource constraints including CPU, memory, and storage to place VMs on PMs, the networked applications running on the virtual clusters still experience performance problems [8], [9], which becomes a bottleneck on the virtualized data centers as identified by [2], [10]. There are more recent studies that propose network or traffic-aware VM placement algorithms [11]–[14]. Other research efforts [15]–[19] suggest VM placement algorithms that guarantee tenants' network bandwidth requirements in order to provide predictable VM performance.

Despite these efforts, the problem caused by network performance variability still remains a challenging issue. The reason is that previous VM placement algorithms ignore the CPU overhead for VM networking functions. For example, in a

/15 $31.00 © 2015 IEEE
DOI 10.1109/ICCAC.2015.40

split driver model, which is used by Xen [20], a driver domain intermediates the communications between VMs and consumes a considerable amount of CPU resources [21]. Although the mechanism to transfer data between VMs is changed from page-flipping to copying, the driver domain still consumes CPU resources to move data from VM to its memory space. As a result, if the driver domain has not enough CPU resources to cover the CPU overhead for VM networking functions, the network bandwidth allocated to each VM cannot be fully utilized during the executions of networked applications. This results in unpredictable application performance running on the virtual clusters.

This paper proposes a VM placement algorithm that considers the CPU overhead incurred to guarantee network bandwidth in a virtualized data center environment. To place a VM on PMs providing stable network bandwidth allocated to the VM, the proposed algorithm finds an appropriate PM whose driver domain can retain a certain amount of CPU resource required to cover the CPU overhead necessary to guarantee the network bandwidth allocated to the VM. In order to decide the CPU overhead, a performance model based on standard linear regression using the data collected from a real environment is used. By comparing the amount of CPU resource available in the driver domain which can be obtained from the VCPU to PCPU ratio with the CPU overhead necessary to guarantee the network bandwidth, the proposed algorithm decides whether the network bandwidth for the VM can be provided or not and selects an appropriate location for VM placement. We first conduct a simple experiment to estimate the accuracy of performance model regarding the CPU overhead and then compare the proposed algorithm with Oktopus [16]. The experimental results show that the proposed algorithm guarantees the network bandwidth allocated to each VM without violations. Since the proposed algorithm considers the CPU overhead to determine the location of VM placement, the network bandwidth allocated to each VM is ensured without violations even if the CPU resources are shared by multiple VMs and CPU utilization available to driver domain is constantly changing.

The rest of this paper is organized as follows. In Section II, previous VM placement algorithms sharing the network bandwidth in virtualized data centers are briefly discussed. Section III presents a performance model to determine the CPU overhead necessary to guarantee the network bandwidth. Section IV describes the proposed VM placement algorithm to guarantee the network bandwidth required by tenants. We evaluate the performance model regarding the CPU overhead and the proposed algorithm in Section V and conclude this paper in Section VI.

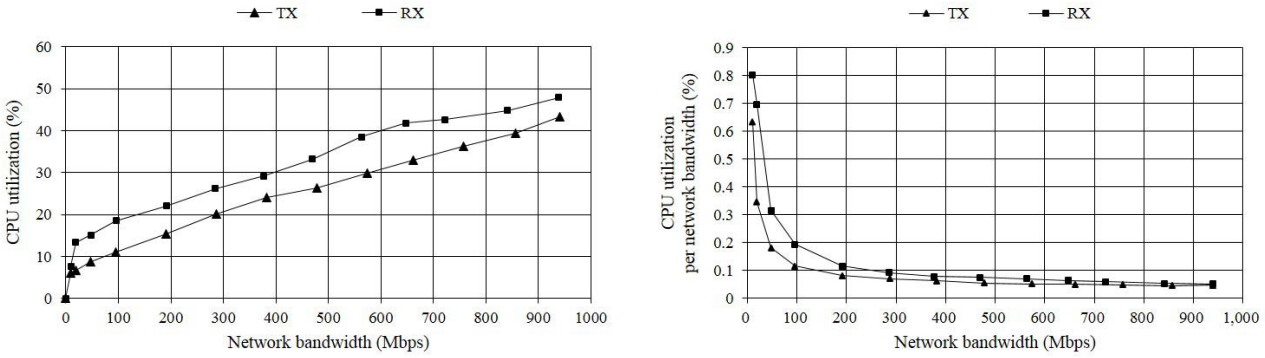
II. Related Works

There have been a lot of research efforts proposing various VM placement algorithms to solve the problem related to the network performance variability. Meng et al. [11] propose a traffic-aware VM placement algorithm considering network resource consumption. While the goal of the algorithm is to reduce the aggregate traffic in the data center, they assume that the traffic between VMs is known and do not consider the link capacity constraints. Brian et al. [12] consider the network constraints with CPU and the memory constraints to solve the VM placement problem. They propose a network-aware VM placement algorithm by satisfying the predicted traffic demand that is resilient to the time-variations. Wang et al. [13] consider the non-deterministic traffic demands between VMs in the VM placement algorithm. By modeling traffic demands as stochastic variables, they solve the VM placement problem as a stochastic bin-packing. Li et al. [14] study the VM placement problem for cost minimization. They define cost functions for the network traffics and the utilization of PMs, and formulate the problem to minimize the costs using a binary search based heuristic algorithm. These studies include the network constraints within the algorithm and reduce the amount of network traffic between VMs to cover the traffic patterns by placing the VMs on PMs appropriately.

There are several different research activities that introduce abstractions to define the amount of network bandwidth required for a virtual cluster and propose the VM placement algorithms to deploy the abstractions in the virtualized data centers. SecondNet [15] and Oktopus [16] introduce systems for providing network bandwidth using deterministic reservation models. SecondNet defines the bandwidth requirements between all VM pairs based on a pipe model and proposes an allocation algorithm to deploy the defined model. Oktopus proposes two abstractions based on a hose model for network bandwidth reservations; virtual cluster (VC) and virtual oversubscribed cluster (VOC). The authors of Oktopus also propose a heuristic algorithm to allocate VC and VOC. Proteus [17] extends the VC abstraction of Oktopus to a fine-grained virtual network abstraction, called time-interleaved virtual clusters (TIVC). CloudMirror [22] defines the network bandwidth requirements as a tenant application graph (TAG) model, which presents the communication patterns of applications consisting of multiple tiers. The authors of CloudMirror provide a deploying algorithm for TAG in [18]. Yu et al. [19] propose a virtual cluster abstraction with stochastic bandwidth requirements between VMs, called stochastic virtual cluster (SVC), and a VM allocation algorithm for their model. Although these studies try to define the abstractions for providing predictable network performance by reserving network bandwidth, they do not consider the CPU overhead in driver domains to intermediate the communications between VMs in the virtualized data centers.

III. CPU Overhead

To fully provide the network bandwidth allocated to each VM, it is important to estimate the CPU overhead necessary to guarantee the network bandwidth in the driver domain. In this section, we first briefly present a split driver model which is used by Xen [20]. The split device driver model separates real I/O devices from VMs and provides virtual I/O device interfaces to VMs by introducing driver domains. The driver domain utilizes the real I/O device drivers to handle I/O activities of VMs. The virtual device driver is split into two important logical components, which are *netback* and *netfront* drivers. The *netback* and *netfront* drivers reside in the driver domain and each VM, respectively. When a packet arrives at a physical network device, an interrupt for the arrival of the packet is delivered to the driver domain. The driver domain then transfers the packet into a software Ethernet bridge. The *netback* driver notifies the arrival of the packet to a corresponding *netfront* driver through an event channel and



(a) (b)

Fig. 1. CPU utilizations with network bandwidth; (a) CPU utilization with varying network bandwidth limitations (b) CPU utilization per network bandwidth

copies the packet into a shared memory in the hypervisor. After the *netfront* driver receives the notification of packet arrival, it looks for the packet in the shared memory and copies it to the VM memory space. Sending a packet from a VM is handled by a reverse procedure.

To transfer the data between VMs, the old version of Xen used a page flipping technique with no data copy involved. The page flipping technique swaps the memory page containing the received packet with a free memory page of the corresponding VM. However, it requires three address remaps and two memory allocation/deallocations for each packet receive operation and two address remaps for each packet transmit operation. The cost of mapping and unmapping pages are equal to the copy cost for large 1500-byte packets, which means that the page flipping is less efficient than copying for small packet sizes [23]. As a result, the page flipping technique was replaced by the data copy mechanism since Xen 3.1 version.

While the data copy mechanism is more efficient than the page flipping technique, the memory-to-memory copy causes a considerable amount of CPU consumption that make a negative effect on the performance of applications running on virtual clusters. Although there exist several studies to improve the performance of network function by optimizing the protocol processing overheads or using network-friendly CPUs, the CPU overheads caused by memory-to-memory copy between *netback* and *netfront* drivers is unavoidable. Since the CPU resources are often over-subscribed and shared by multiple VMs in the virtualized data centers as mentioned in Section I, the performance would be degraded when the CPU consumption for VM networking is not considered.

To estimate the amount of CPU consumption for memory-to-memory copy in VM networking functions, we performed several experiments with various data sizes on a real testbed. For the experiments, we used two PMs (8 PCPUs, 16GB of memory, 1Gbps network, Ubuntu Server 15.04 64bit, kernel 3.19.0-18), where one of them runs Xen hypervisor 4.5 and hosts one VM (4 VCPUs, 2GB of memory, Ubuntu 15.04 64bit, kernel 3.19.0-18). We assumed that all VCPUs are not pinned to PCPUs and have the same scheduling weight values for simplicity. We used a network benchmark tool *iperf* to change the size of memory-to-memory copy and for measuring the network bandwidth, and a traffic control tool *tc* for controlling the network bandwidth requested by the VM.

In Fig. 1, the results for transmitting and receiving a packet are represented by *TX* and *RX*, respectively. As shown in Fig. 1(a), as the amount of VM networking traffic increases, more CPU resources are needed as we have expected. We can also see that receiving a packet requires more CPU resources than transmitting a packet. Fig. 1(b) presents the CPU overhead per bandwidth which is calculated from the values of Fig. 1(a). As shown in Fig. 1(b), the CPU overhead can be represented by exponential (or power) functions with a negative value for an exponent. From the results shown in Fig. 1, we define two functions, CPU_{tx} and CPU_{rx} for the CPU overhead necessary to guarantee the network bandwidth. For each function, we estimate the coefficients of an exponential function $y = ax^b$ from the values shown in Fig. 1(b), where x denotes the amount of network bandwidth currently used and y is the CPU utilization per network bandwidth at the corresponding network bandwidth. Then, we use a log transformation of the original data and apply a standard linear regression [24] with $ln(y)$ as a dependent variable and $ln(x)$ as an explanatory variable.

$$CPU_{tx}(x) = 1.7352 \cdot x^{-0.553} \quad (1)$$

$$CPU_{rx}(x) = 3.6126 \cdot x^{-0.631} \quad (2)$$

, where x is the amount of network bandwidth used.

Using (1) and (2), we obtain the amount of CPU resources (CPU utilization) needed by a driver domain to intermediate a specific amount of network bandwidth for VM networking.

IV. CPU Overhead-aware VM Placement Algorithm

To provide the network bandwidth requested by tenants, a virtualized data center needs to determine which PM has enough network bandwidth for VMs in the virtual clusters. It is also necessary to ensure that the amount of network bandwidth requested by tenant is available for all virtual links between VMs on a physical network topology. For simplicity, we assume that the virtualized data center uses a tree-based

network topology. The main goal of the proposed algorithm is to estimate the CPU overhead for VM networking functions discussed in Section III and use the value for deciding an appropriate VM location in a virtual cluster. While it is generally true that a PM has enough network bandwidth to deploy the VMs and the amount of residual network bandwidth on physical links connecting to the other PMs is larger than the amount of network bandwidth required, we need to consider that the driver domain intermediates the communications between VMs and thereby consumes a considerable amount of CPU resources to guarantee the network bandwidth.

Since the VM placement problem with bandwidth constraints is known as NP-hard [25], we use a heuristic algorithm which is similar to that of Oktopus. First of all, the algorithm selects a set of candidate PMs having enough network bandwidth to execute VM networking functions belonging to a virtual cluster. To decide the candidate PMs, the amount of network bandwidth allocated to VMs running on each PM pm should be calculated by (3) since the PM may already run one or more VMs for other virtual clusters.

$$NET^{pm}_{alloc} = \sum_{vm \in VM} NET^{vm}_{alloc} \tag{3}$$

, where VM is a set of VMs running on the PM pm.

The algorithm then finds a set of candidate PMs satisfying (4).

$$PM_{cand} = \{pm \mid \exists pm \in PM \ that \\ NET^{pm}_{cap} - NET^{pm}_{alloc} \geq NET_{req}\} \tag{4}$$

, where PM is a set of PMs, NET^{pm}_{cap} is a network bandwidth capacity of PM pm, and NET_{req} is the amount of network bandwidth requested for the virtual cluster.

The next is to apply the CPU overhead to the VM placement problem. To do this, we need to estimate the CPU overhead for network bandwidth guarantee in the driver domain. While we have defined the CPU overhead both for *TX* and *RX*, the proposed algorithm simplifies the CPU overhead as CPU_{rx}. The reason is that the algorithm intends to guarantee the same amount of network bandwidth for both directions (bidirectional virtual links) and CPU_{rx} covers CPU_{tx} at all times. However, it is straightforward to use both functions if we assume that the virtual links are unidirectional with different bandwidth amount (directed virtual links). Then, the CPU overhead can be calculated by (5) using (2).

$$CPU_{overhead} = net \cdot CPU_{rx}(net) \tag{5}$$

, where net is a specific amount of network bandwidth.

Using (5), the algorithm calculates the CPU overhead needed both for the network bandwidth allocated to the VMs currently running on the PM and for a new VM that can be added. If the driver domain has more CPU resources than the CPU overhead calculated by (5), the PM is removed from the set of candidate PMs. To provide the deterministic network bandwidth reservation, the CPU resources available in the driver domain is estimated when all the VCPUs are busy. The amount of CPU utilization available in the driver domain depends on a VCPU to PCPU ratio, which represents how many VCPUs share PCPUs that can be calculated by $VCPU_{total}/PCPU_{total}$, where $VCPU_{total}$ is the number of VCPUs including driver domain and $PCPU_{total}$ is the number of PCPUs in a PM. For example, if there are five VCPUs running on two PCPUs, the ratio is 2.5. Using the ratio, the CPU utilization available in the driver domain dd is calculated by (6).

$$CPU^{dd}_{avail} = VCPU^{dd} \cdot \frac{100.0}{Ratio} \tag{6}$$

, where $VCPU^{dd}$ is the number of VCPUs allocated to dd. For example, if $PCPU_{total} = 4$, $VCPU_{total} = 8$ and $VCPU^{dd} = 2$, then $CPU^{dd}_{avail} = 100$.

Therefore, all the PMs belonging to the set of candidate PMs satisfy $CPU^{dd}_{avail} \geq CPU_{overhead}$ after considering the CPU overhead. However, It should be noted that the number of VCPUs increases when deploying a new VM on the PM and thereby the VCPU to PCPU ratio is changed. The algorithm thus considers the incremental number of VMs when calculating the VCPU to PCPU ratio by (7).

$$Ratio = \frac{VCPU_{total} + \Delta VCPU}{PCPU_{total}} \tag{7}$$

, where $\Delta VCPU$ is the incremental number of VCPUs for a new VM.

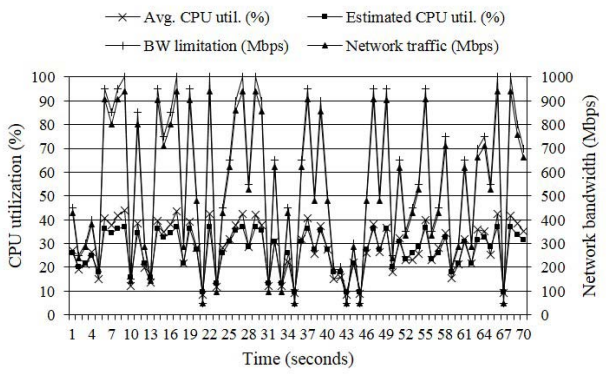
At last, the proposed algorithm selects the smallest subtree that can accommodate the virtual cluster including all VMs and the virtual links between them. For this selection, the algorithm calculates the number of VMs available in each subtree of tree-based network topology. When calculating the number of VMs for multiple subtrees of the subtree, a physical link between the sub-subtree and the rest of the subtree needs to satisfy $NET^{l}_{res} \geq min(n, |VM_{req}| - n) \cdot NET_{req}$, where VM_{req} is the number of VMs requested, NET^{l}_{res} is the amount of residual network bandwidth on the physical link l and n is the number of VMs for allocating to the sub-subtree. If there exist multiple subtrees available at the same level, the algorithm selects a subtree with the least amount of residual network bandwidth on a physical link connecting to the rest of the network topology.

V. Performance Evaluation

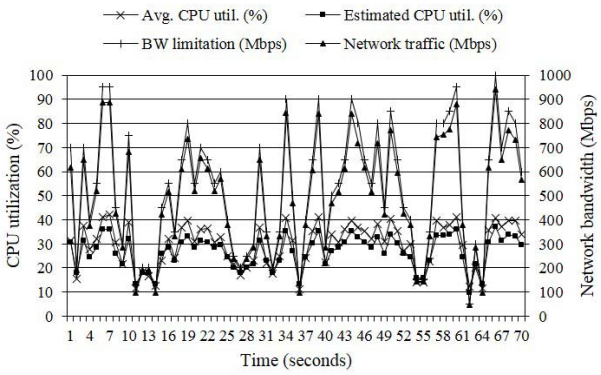
In this section, we first show the accuracy of our CPU overhead model discussed in Section III on our testbed and evaluate our algorithm through a simulation.

A. Accuracy of CPU Overhead Model

Our testbed consists of two PMs with two Intel Xeon 2.40GHz quad-core processors (8 PCPUs), 16GB memory, and a 1Gbps network interface card. The PMs run Ubuntu 15.04 server 64bit and kernel 3.19.0-18. The Xen hypervisor 4.5 is installed in a PM and the driver domain in the PM has 4 VCPUs. One VM with 4 VCPUs and 2GB memory is used for this experiment. The VM runs the same OS and kernel with PMs. All the VCPUs in the PM have the same scheduling weight values. We generate network traffic between



(a)



(b)

Fig. 2. CPU utilizations and network bandwidth; (a) TX (b) RX

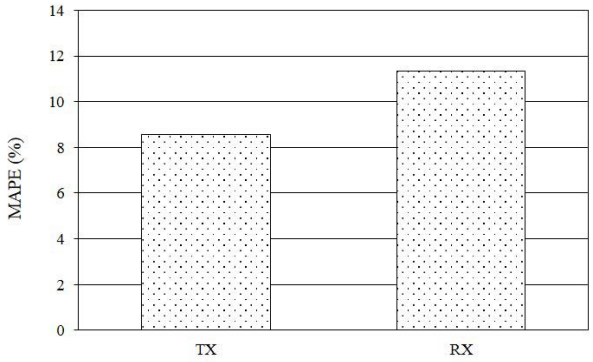


Fig. 3. Accuracy of CPU overhead model

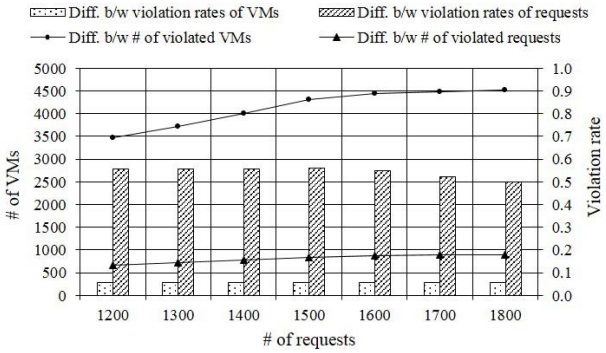


Fig. 4. Differences between violation rates and between the number of violations

the VMs running on one PM and the other PM using a network benchmark tool *iperf* and restrict the network bandwidth using a traffic control tool *tc*.

We conduct two experiments to evaluate the accuracy of our CPU overhead model by transmitting and receiving network traffic, respectively. Fig. 2 shows the CPU utilizations and network bandwidth for both cases, *TX* and *RX*. As shown in Fig. 2, our CPU overhead model relatively well estimates the CPU utilization for the corresponding network bandwidth. And we can see that the CPU utilizations follow the network bandwidth actually used.

Fig. 3 presents the accuracies of *TX* and *RX* using a mean absolute percentage error (MAPE) [26], which is widely used for measuring forecast errors in statistics and calculated by (8). Using MAPE, the average error rate of CPU overhead estimation is calculated to 8.5% for TX and 11.3% for RX. Contrary to our expectation, the error rates are relatively high and the value of RX is higher than that of TX. This is due to the fact that there exist some CPU consumptions required by our experiment tools including a monitoring tool executing on the driver domain. We think that these CPU consumptions make it difficult to estimate the CPU overhead accurately and thereby the CPU utilization of the driver domain is affected by a big magnitude.

$$\left(\sum_{i=1}^{n}\left|\frac{x_i-\hat{x}_i}{x_i}\right|\right)\cdot\frac{100.0}{n} \tag{8}$$

,where n is the number of data, x_i is i-th actual CPU utilization, and $\hat{x}_i$ is i-th CPU utilization estimated by the CPU overhead model.

B. VM Placement Algorithm

We implement a simulator to evaluate the proposed VM placement algorithm. A three-level tree-based topology is used for data center network. With no path diversity, the network topology is over-subscribed by 4:1 ratio. We implement two VM placement algorithms in the simulator and compare them. One is an algorithm without considering CPU overhead, which is similar to that of Oktopus, and the other is the proposed algorithm. The data center network consists of 16,000 PMs. Forty PMs form a rack and are connected to a top-of-rack (ToR) switch with 1Gbps network bandwidth. Every 20 ToR switches are linked to an aggregate switch with 10Gbps and 20 aggregate switches are connected to a root switch with 50Gbps. To generate tenants' requests, we use Gaussian distribution. For each request, the number of VMs is randomly selected by the distribution of mean 50 and standard deviation 5. The

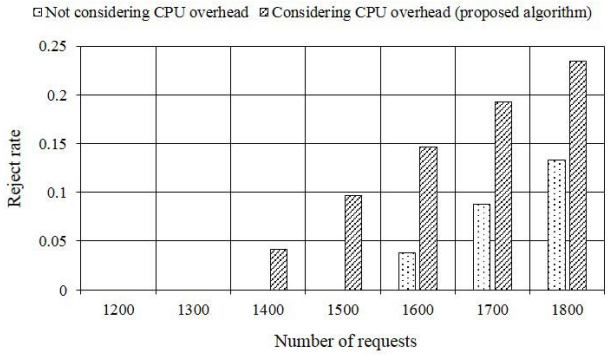


Fig. 5. Reject rate for tenants' requests

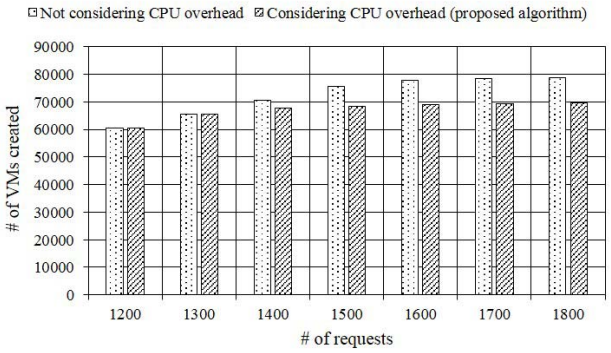


Fig. 6. The number of VMs created

network bandwidth of the request is also obtained by the distribution of mean 200 and standard deviation 50. Each PM has 8 PCPUs and hosts a driver domain with 4 VCPUs. The VMs of requests are generated with 2 VCPUs.

Fig. 4 shows the differences between the number of violated VMs placed by the algorithm with and without considering CPU overhead. We measure the number of violations in the VMs and depict the average values of twenty iterations in Fig. 4. In Fig. 4, we can also see the differences between the violation rates in the VMs. Since the proposed algorithm considers the CPU overhead to allocate the VMs without violations, it outperforms the algorithm without considering CPU overhead. One thing we need to consider is that the violation rate of requests is larger than that of VMs. The reason is that a violated VM belonging to a request makes the request to violate the tenant's bandwidth requirement and thereby increases the violation rate of requests.

However, since the proposed algorithm rejects the tenants' requests if the driver domains have not enough CPU resources to cover the CPU overhead, the reject rate may become larger. Fig. 5 presents the reject rate for tenants' requests when considering the CPU overhead to place VMs or not. As shown in Fig. 5, the reject rate of the proposed algorithm is larger than that of the other. Likewise, the number of created VMs placed by our algorithm is smaller than the other as shown in Fig. 6. The reason why the reject rate of the algorithm without considering CPU overhead rises from 1,600 requests is that, since each PM has 1Gbps (1024Mbps) bandwidth capacity, the average network bandwidth requirement of requests is 200Mbps, and the average number of VMs per request is 50, the number of requests that can be accepted on average is closed to $16,000 * 1024Mbps/200Mbps/50 \approx 1638$. Therefore, we can conclude that the algorithm without considering the CPU overhead utilizes entire network bandwidth of all the PMs while it makes a lot of violations occurred by the CPU overhead. It is also shown in Fig. 6 that the number of VMs created increases up to a value closed to $16,000 * 1024Mbps/200Mbps = 81,920$ on average.

VI. Conclusions

The unpredictable application performance occurred by network performance variability becomes a critical problem in the virtualized data centers. Although there have been a lot of research efforts to address this problem by designing various VM placement algorithms, they miss the fact that the CPU overhead for VM networking functions affects the network bandwidth allocated to each VM. This paper proposes a CPU overhead-aware VM placement algorithm that considers the CPU overhead needed for the VM networking functions in the driver domain. If the VMs belonging to a virtual cluster requested by a tenant can't utilize sufficient network bandwidth, the virtual cluster suffers from the violations. However, the proposed algorithm estimates the CPU overhead using our performance model and places the VMs by considering the CPU utilization available in the driver domain. It is shown in the experiments that the proposed algorithm doesn't experience any violations, although it may reject more requests from the tenants. Since the proposed VM placement algorithm is reactive and static in a sense that we do not consider the temporal aspects of network usages, we are currently trying to extend the algorithm to proactive and dynamic fashion and to include more complex network topologies with path diversity such as Fat-tree. We are also planning to improve the accuracy of our model to estimate the CPU overhead.

Acknowledgment

This work was supported by Next-Generation Information Computing Development Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT & Future Plannig (No. 2012M3C4A7033348) and Institute for Information & communications Technology Promotion (IITP) grant funded by the Korea Government (MSIP) (No. B0101-15-0644, Research on High Performance and Scalable Manycore Operating System).

References

[1] Overcommitting, Compute Nodes, OpenStack documents, http://docs.openstack.org/openstack-ops/content/compute_nodes.html

[2] 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 Computer Communication Review, vol. 39, no. 4, pp. 5162, 2009.

[3] N. Farrington and A. Andreyev, Facebook's Data Center Network Architecture, in Proc. of IEEE Optical Interconnects Conference, pp. 49-50, Santa Fe, NM, USA, 2013.

[4] VMware Capacity Planner, http://www.vmware.com/products/capacity-planner

[5] IBM CloudBurst, https://www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/IBM CloudBurst/page/IBM CloudBurst Product Overview

[6] M. Sindelar, R. K. Sitaraman, and P. Shenoy, Sharing-Aware Algorithms for Virtual Machine Colocation, in Proc. of the 23rd annual ACM symposium on Parallelism in algorithms and architectures (SPAA), pp. 367-378, San Jose, CA, USA, 2011.

[7] S. Lee, R. Panigrahy, V. Prabhakaran, V. Ramasubrahmanian, K. Talwar, L. Uyeda, and U. Wieder, Validating Heuristics for Virtual Machine Consolidation, Microsoft Research, Technical Report, 2011.

[8] G. Wang and T. S. E. Ng., The Impact of Virtualization on Network Performance of Amazon EC2 Data Center, In Proc. of the 29th conference on IEEE Information communications (INFOCOM), pp. 1-9, San Diego, CA, USA, 2010.

[9] D. Mangot, Measuring EC2 system performance, 2009, http://tech.mangot.com/roller/dave/entry/ec2_variability_the_numbers_revealed

[10] J. C. Mogul and L. Popa, What We Talk About When We Talk About Cloud Network Performance, in ACM SIGCOMM Computer Communication Review, vol. 42, no. 5, pp. 44-48, 2012.

[11] X. Meng, V. Pappas, and L. Zhang, Improving the Scalability of Data Center Networks with Traffic-aware Virtual Machine Placement, in Proc. of the 29th conference on IEEE Information communications (INFOCOM), pp. 1154-1162, San Diego, CA, USA, 2010.

[12] O. Biran, A. Corradi, M. FAnelli, L. Foschini, A. Nus, D. Raz, and E. Silvera, A Stable Network-Aware VM Placement for Cloud Systems, in Proc. of the 12th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing (CCGrid), pp. 498-506, Ottawa, Canada, 2012.

[13] M. Wang, X. Meng, and L. Zhang, Consolidating Virtual Machines with Dynamic Bandwidth Demand in Data Centers, in Proc. of the 28th conference on IEEE Information communications (INFOCOM), pp. 71-75, Shanghai, China, 2011.

[14] X. Li, J. Wu, S. Tang, and S. Lu, Let's Stay Together: Towards Traffic Aware Virtual Machine Placement in Data Centers, in Proc. of the 31th conference on IEEE Information communications (INFOCOM), pp. 1842-1850, Toronto, ON, Canada, 2014.

[15] 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 ACM International Conference on emerging Networking EXperiments and Technologies (Co-NEXT), no. 15, pp. 1-12, New York, NY, USA, 2010.

[16] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron, Towards Predictable Datacenter Networks, In Proc. of the ACM SIGCOMM conference, pp. 242-253, Toronto, ON, Canada, 2011.

[17] D. Xie, N. Ding, Y. C. Hu, R. Kompella, The Only Constant is Change: Incorporating Time-Varying Network Reservations in Data Centers, in ACM SIGCOMM Computer Communication Review, vol. 42, no. 4, pp. 199-210, 2012.

[18] J. Lee, Y. Turner, M. Lee, L. Popa, S. Banerjee, J.-M. Kang, and P. Sharma, Application-Driven Bandwidth Guarantees in Datacenters, in Proc. of the 2014 ACM conference on SIGCOMM, pp. 467-478, Chicago, IL, USA, 2014.

[19] L. Yu and H. Shen, Bandwidth Guarantee under Demand Uncertainty in Multi-tenant Clouds, in Proc. of the 34th IEEE International Conference on Distributed Computing Systems (ICDCS), pp. 258-267, Madrid, Spain, 2014.

[20] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, Xen and the Art of Virtualization, in Proc. of the 19th ACM Symposium on Operating Systems Principles (SOSP), pp. 164-177, Bolton Landing, NY, USA, 2003.

[21] L. Cherkasova and R. Gardner, Measuring CPU Overhead for I/O Processing in the Xen Virtual Machine Monitor, in Proc. of Annual Conference on USENIX Annual Technical Confernece (ATEC), pp. 387, 390, Anaheim, CA, USA, 2005.

[22] J. Lee, M. Lee, L. Popa, Y. Turner, and S. Banerjee, CloudMirror: Application-Aware Bandwidth Reservations in the Cloud, in Proc. of USENIX Workshop Hot Topics Cloud Computing (HotCloud), 2013.

[23] J. R. Santos, G. Janakiraman, and Y. Turner, Network optimizations for PV guests, 3rd Xen Summit, 2006.

[24] Simple linear regression, http://en.wikipedia.org/wiki/Simple_linear_regression

[25] A flexible model for resource management in virtual private networks N. G. Duffield, P. Goyal, A. Greenberg, P. Mishra, K. K. Ramakrishnan, and J. E. van der Merive, A Flexible Model for Resource Management in Virtual Private Networks, in ACM SIGCOMM Computer Communication Review, vol. 29, no. 4, pp. 95-108, 1999.

[26] Mean Absolute Percentage Error, http://en.wikipedia.org/wiki/Mean_absolute_percentage_error