VMBeam: Zero-copy Migration of Virtual Machines for Virtual IaaS Clouds

Kenichi Kourai
Kyushu Institute of Technology
Fukuoka, Japan
kourai@ci.kyutech.ac.jp

Hiroki Ooba
Kyushu Institute of Technology
Fukuoka, Japan
hiroki@ksl.ci.kyutech.ac.jp

***Abstract*—Virtual Infrastructure-as-a-Service (IaaS) clouds are emerging for secondary cloud service providers to manage their own IaaS clouds on top of existing IaaS clouds. In virtual IaaS clouds, guest virtual machines (VMs) run inside cloud VMs provided by existing IaaS clouds. Unlike traditional IaaS clouds, they can be migrated between cloud VMs co-located at the same host. However, the performance of such VM migration is low due to slow virtual networks and doubled system loads. To optimize VM migration between co-located cloud VMs, we propose *zero-copy migration* for virtual IaaS clouds. Zero-copy migration just relocates the memory image of a guest VM without any copy. To enable live migration with negligible downtime, it first makes the memory of a guest VM share with the destination cloud VM and thereafter completes memory relocation. We have implemented a system called *VMBeam* for enabling zero-copy migration in Xen. According to our experimental results, zero-copy migration could achieve high migration performance and low system loads.**

***Keywords*-Virtual machines, live migration, virtual IaaS clouds, nested virtualization**

I. Introduction

Infrastructure-as-a-Service (IaaS) clouds are widely used as a basis of various services. They provide users with virtual machines (VMs), in which users can construct their systems from scratch. Recently, *virtual IaaS clouds* are emerging [1]–[4]. A virtual IaaS cloud is a cloud that is constructed on top of an existing IaaS cloud. Like secondary Internet service providers (ISPs), secondary cloud service providers (CSPs) can manage their own IaaS clouds without having data centers, which take high operational cost. For example, they can provide value-added services such as intrusion detection in their clouds. Using a technique called nested virtualization [5], virtual IaaS clouds run *guest VMs* inside *cloud VMs* provided by the underlying IaaS clouds. In virtual IaaS clouds, VM migration is still used, e.g., to keep the availability of guest VMs when cloud VMs are maintained. In particular, VM migration between cloud VMs co-located at the same host is specific to virtual IaaS clouds.

However, the migration performance between co-located cloud VMs is lower than that between physical hosts. First, the virtual network between cloud VMs is often slower than the physical network between hosts. This is due to the overhead of the network virtualization. In virtual IaaS clouds, network interface cards (NICs) have to be emulated even for the communication between co-located cloud VMs. Second, system loads are doubled because one host plays two roles of the source and destination of VM migration. The source cloud VM has to encrypt and transfer the large memory image of a guest VM, whereas the destination cloud VM has to receive and decrypt it. During this migration process, many memory copies are executed. High loads are imposed on CPUs, memory, and networks even for one role, not to mention two roles.

To solve this problem, we propose *zero-copy migration* for virtual IaaS clouds. Zero-copy migration is an optimization for VM migration between cloud VMs co-located at the same host. It just relocates the memory image of a guest VM without any copy, instead of transferring the encrypted image via the virtual network. To enable live migration, zero-copy migration first makes the guest VM share its memory with the destination cloud VM, using *inter-guest memory sharing*. Then it completes memory relocation by releasing the memory at the source cloud VM. Thus it is not necessary to re-transfer the memory modified by the guest VM during VM migration. Consequently, zero-copy migration can reduce both the migration time and downtime.

We have implemented a system called *VMBeam* for enabling zero-copy migration in Xen 4.2.2 [6]. According to our experimental results, zero-copy migration was up to 7.5 times faster than the state-of-the-art VM migration for virtual IaaS clouds. The downtime became 17% shorter at least. Under memory-intensive workloads, the increase in migration performance was much more significant. In addition, zero-copy migration could suppress the total loads of CPUs, memory, and networks to 12.5%, nearly 0%, and nearly 0% of the traditional VM migration, respectively.

This paper is organized as follows. Section II describes issues of VM migration between co-located cloud VMs. Section III proposes zero-copy migration for virtual IaaS clouds and Section IV shows the experimental results. Section V describes the related work and Section VI concludes this paper.

II. VM Migration in Virtual IaaS Clouds

VM migration is still important in virtual IaaS clouds. Guest VMs can be migrated to enable uninterrupted maintenance of cloud VMs, e.g., software upgrade. Some virtual-

ization software such as Xen 4.7 [6] provide live patching, which enables upgrading even the hypervisor in cloud VMs without reboots. However, not all patches can be applied using this technique. In addition, VM migration is used to consolidate guest VMs into a smaller number of cloud VMs and stop idle cloud VMs. This saves the cost of secondary CSPs like in the traditional IaaS clouds, which use VM migration to save power by shutting down idle hosts. Also, load balancing among cloud VMs can be achieved by VM migration.

In virtual IaaS clouds, a guest VM can be migrated between cloud VMs co-located at the same host. This is specific to virtual IaaS clouds, where guest VMs run in cloud VMs, instead of running directly on physical hosts. The co-location of cloud VMs occurs when the underlying IaaS cloud initially creates cloud VMs for a virtual IaaS cloud at the same host. Even if not, they can migrate cloud VMs to the host running other cloud VMs included in the same virtual IaaS cloud. In fact, it is report that the probability of the co-location was more than 8.4% in Amazon EC2 [7]. The co-location is beneficial to both primary and secondary CSPs. Primary CSPs can save the cost by reducing the number of active hosts. Secondary CSPs can provide customers with faster network between guest VMs.

VM migration between co-located cloud VMs is expected to be faster than that between physical hosts. Such VM migration can just move the memory image of a guest VM from the source to the destination cloud VM inside one host. However, this is often not the case. According to our experiments in Section IV, it took nearly 6 minutes to migrate a guest VM with 4 GB of memory between cloud VMs at the same host. This migration time was 3.7 times longer than that between physical hosts in the traditional IaaS cloud.

There are two reasons for such a slowdown of VM migration. First, the virtual network between cloud VMs is often slower than the physical network due to the overhead of network virtualization. In virtual IaaS clouds, NICs have to be emulated for cloud VMs to communicate with each other even if cloud VMs are co-located at the same host. This issue has been partly solved by using a fast virtual network provided by Xen-Blanket [2]. Xen-Blanket para-virtualizes the network of cloud VMs and avoids the full emulation of NICs. However, the overhead of the virtual network still remains.

Second, system loads are doubled because one host plays two roles of the source and destination of VM migration. VM migration stresses the involving hosts and networks largely. The memory image of a guest VM is encrypted to prevent eavesdropping and tampering during the transfer via the public virtual network. Such attacks can be mounted by administrators and the other users in the underlying IaaS clouds. In addition, the copies of a large memory image are executed many times. Therefore VM migration can occupy CPUs and memory bandwidths at both the source and destination cloud VMs.

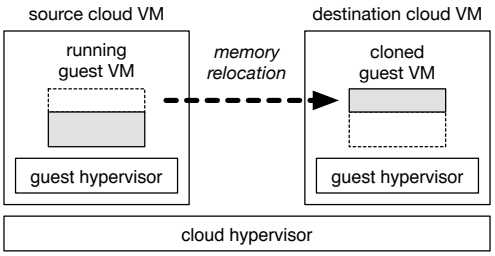


Figure 1. Zero-copy migration for virtual IaaS clouds.

In addition, the downtime during VM migration between co-located cloud VMs is also longer. Live migration [8] achieves negligible downtime as follows. First, the source cloud VM transfers the memory image of a guest VM to the destination cloud VM with the guest VM running. After transferring the entire memory image, it repeatedly re-transfers only modified memory regions. At the final stage, the source cloud VM stops the guest VM and transfers the remaining memory regions modified. Only the time needed for the final stage becomes the downtime, but it is longer in virtual IaaS clouds due to the same reasons above. In particular, the impact is larger under memory-intensive workloads due to much more memory to be transferred at the final stage.

III. VMBeam

A. Zero-copy Migration

To improve the migration performance of guest VMs, we propose *zero-copy migration* for virtual IaaS clouds. Zero-copy migration optimizes VM migration between cloud VMs co-located at the same host. It achieves zero copy by leveraging the fact that VM migration is performed inside one host. It just *relocates* the memory of a guest VM in the source cloud VM to the destination cloud VM without any copy, as illustrated in Fig. 1. The memory image is not transferred via the virtual network.

Zero-copy migration can reduce both the migration time and downtime. Since it does not use the virtual network for data transfers, it does not suffer from the overhead of network virtualization and enables fast VM migration. Consequently, it can decrease the CPU load as well as the network load. In addition, zero-copy migration can reduce the copy overhead of the large memory image of a guest VM by simply relocating the memory. Since no data is exposed to the outside of the source and destination guest VMs thanks to no copy, the encryption of the memory image is not necessary. Furthermore, zero-copy migration can also achieve lower total system loads during VM migration by reducing the migration time itself.

However, naively relocating the memory of a guest VM cannot support live migration. If the memory transfer is simply replaced with memory relocation, a guest VM cannot continue to run during VM migration. This is because a part of the memory does not exist at the source cloud VM after it has been relocated. Each memory page can exist only at either the source or the destination cloud VM. Worse, it takes a substantial time to complete memory relocation of the guest VM. In terms of the downtime, it is not realistic to stop a guest VM during memory relocation.

To enable live migration, zero-copy migration consists of two steps for relocating the memory of a guest VM. First, it makes a guest VM in the source cloud VM *share* the memory with the destination cloud VM, which is called *inter-guest memory sharing*. Since the shared memory exists in both the source and destination cloud VMs, the guest VM can continue to run with its memory even while VM migration is in progress. As the second step, zero-copy migration releases the memory of the source guest VM and completes relocating it to the destination cloud VM at the final stage of VM migration.

Although the traditional live migration needs multiple iterations for re-transferring modified memory regions, zero-copy migration is completed in only one iteration. Thanks to the memory sharing, modifications to the memory of the source guest VM are directly reflected to the destination guest VM. Zero-copy migration does not need to detect memory pages modified by guest VMs during VM migration for re-transfers. It is not necessary to transfer modified memory regions again. This can largely reduce the migration time especially for guest VMs that execute memory-intensive workloads. It also reduces the time for transferring the remaining memory at the final stage of migration. Since a guest VM is suspended at the final stage, this leads to the further downtime reduction.

Note that inter-guest memory sharing used in zero-copy migration does not increase the attack surface. First, only system administrators in virtual IaaS clouds have the capability of using this mechanism. The user of a guest VM cannot eavesdrop on or tamper with memory regions outside the VM illegally. Second, even the system administrators cannot share the memory between cloud VMs that belong to different virtual IaaS clouds. To share the memory, the agreement between source and destination cloud VMs is necessary.

B. Implementation

We have implemented a system called *VMBeam* for enabling zero-copy migration in Xen 4.2.2 [6]. The cloud hypervisor runs cloud VMs as fully virtualized (HVM) guests. In a cloud VM, the guest hypervisor runs guest VMs as HVM guests. In Xen, the management VM called Dom0 is used to manage the other VMs and emulate I/O devices. We refer to Dom0s running on top of the cloud and guest hypervisors as the *cloud* and *guest Dom0s*, respectively. Due to space limitations, we omit the implementation details of VMBeam in this paper. For the details, see our previous workshop paper [9].

IV. Experiments

We have conducted several experiments to show the effectiveness of zero-copy migration. For this purpose, we compared three systems: (1) VMBeam, (2) Xen using nested virtualization (Xen-Nest), and (3) Xen-Blanket [2]. Xen-Nest uses the virtual network between cloud VMs for transferring the memory image of a guest VM. Xen-Blanket enables direct communication between cloud VMs via the cloud Dom0 to improve the performance of the virtual network.

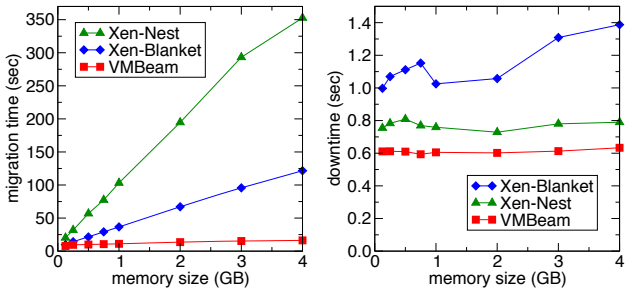
In our experiments, we used Xen 4.2.2 as the cloud hypervisor and ran the cloud Dom0 and two co-located cloud VMs on top of it. In each cloud VM, we used Xen 4.2.2 or Xen-Blanket 4.1.1 as the guest hypervisor and ran the guest Dom0. We ran Linux 3.2.0 in the cloud Dom0 and Linux 3.5.0 in the guest Dom0. In addition, we ran one guest VM on the source cloud VM.

For a physical machine in an existing IaaS cloud, we used a PC with an Intel Xeon E5-2665 processor (8 cores), 32 GB of memory, and a gigabit Ethernet NIC. We assigned three CPU cores and 10 GB of memory to each cloud VM. The cloud Dom0 was assigned the remaining two CPU cores and 9.1 GB of memory. Among the resources assigned to each cloud VM, we assigned one CPU and memory between 128 MB and 4 GB to a guest VM. The guest Dom0 was assigned one CPU and 9.5 GB of memory.

A. Migration Time

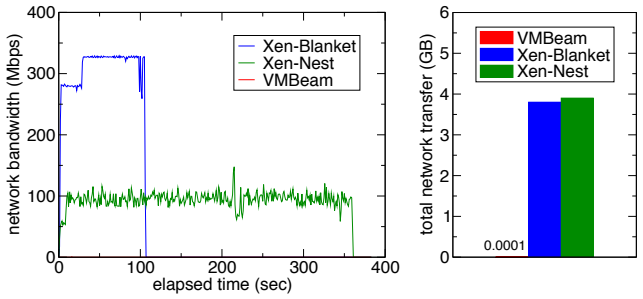
We changed the size of the memory assigned to a guest VM between 128 MB and 4 GB and measured the time needed for VM migration. We used the `xl migrate` command, which migrates a VM using an SSH tunnel. Fig. 2(a) shows the average migration time. The results show that the migration time was proportional to the memory size of a guest VM in all the systems. VMBeam achieved the shortest migration time and could complete the migration of a guest VM with 4 GB of memory only in 16.3 seconds.

VM migration in Xen-Blanket was 1.2 to 7.5 times slower than that of VMBeam although it was 2.1 to 2.9 times faster than that of Xen-Nest. The main reason why Xen-Nest was so slow is the large difference in throughput of the virtual network. In contrast, the root cause of the low performance in Xen-Blanket is encryption and inter-process communication. To examine the impact of encryption of the memory image, we disabled encryption in SSH and measured the migration time. We specified the none cipher in SSH instead of the default cipher. As a result, the migration time became shorter in both Xen-Blanket and Xen-Nest.



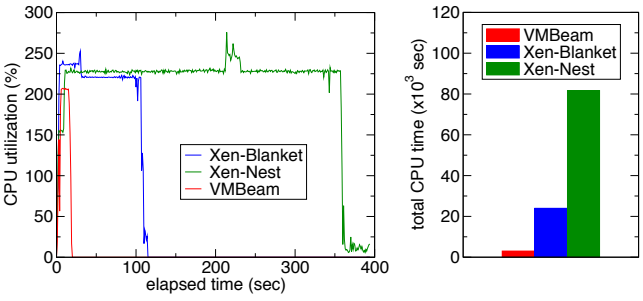
(a) Migration time (b) Downtime

Figure 2. Migration performance.



(a) Bandwidth (b) Total transfer

Figure 3. The network load during VM migration.



(a) Utilization (b) Total time

Figure 4. The CPU load during VM migration.

However, VM migration in VMBeam was still up to 3.5 times faster than that of Xen-Blanket.

B. Downtime

We measured the downtime during VM migration, changing the size of the memory assigned to a guest VM. The downtime is from when a guest VM is stopped at the final stage of VM migration until it is resumed at the destination cloud VM. The average downtime is shown in Fig. 2(b) and did almost not depend on the memory size of a guest VM. The downtime in VMBeam was the shortest and 0.6 second.

Xen-Nest achieved the second shortest downtime, but the downtime was 27% longer on average than that of VMBeam. It took a longer time to transfer modified memory via the slow virtual network at the final stage of VM migration, where the guest VM was stopped. Surprisingly, the downtime in Xen-Blanket was 32% to 76% longer than that of Xen-Nest. Since the network throughput in Xen-Blanket is much higher than that of Xen-Nest, it should take a shorter time to transfer the remaining information at the final stage. The root cause is currently unclear. Even when we disabled encryption of SSH, the improvement was not large. The downtime in Xen-Nest became shorter by up to 0.14 second, but that of Xen-Blanket became rather longer by 0.2 second at worst.

C. Network Load

We examined the network load while we migrated a guest VM with 4 GB of memory. We measured the size of network data transferred in the guest Dom0. Fig. 3(a) shows the changes in consumed network bandwidth and Fig. 3(b) shows the total network transfer during VM migration. Since VMBeam did not use the virtual network for transferring the memory image, the total network transfer was almost zero. For the other systems, approximately 4 GB of data were transferred at almost constant rates. The transfer lasted for 105 seconds in 300 Mbps in Xen-Blanket, whereas it lasted for 360 seconds in 100 Mbps in Xen-Nest.

D. CPU Load

To examine how much zero-copy migration reduced the CPU load, we measured the CPU load of the entire host during the migration of a guest VM with 4 GB of memory. In the cloud Dom0, we obtained the sum of the CPU utilizations of two cloud VMs and the cloud Dom0. Fig. 4(a) shows the changes in CPU utilization and Fig. 4(b) shows the total CPU time used during VM migration. The CPU utilization in VMBeam was 20% less than those in Xen-Blanket and Xen-Nest on average. This is because VMBeam did almost not use the virtual network, which is processed in not only two cloud VMs but also the cloud Dom0. The total CPU time in VMBeam was only 12.5% of Xen-Blanket and 3.7% of Xen-Nest.

E. Memory Load

We examined the amount of memory accessed while we migrated a guest VM with 4 GB of memory. Since we could not obtain the statistics of memory accesses directly from hardware, we estimated them from the number of transferred memory pages and the memory access pattern in each system. Fig. 5 shows the estimated total memory access. In VMBeam, the cloud hypervisor relocates all of the memory pages of the source guest VM to the destination cloud VM. Therefore VMBeam needs no memory access.

For Xen-Nest, the migration client in the source guest Dom0 reads the memory of a guest VM via a memory-mapped region and sends it to the SSH client. The SSH client encrypts the received data and sends the encrypted data to the SSH server in the destination cloud VM via the virtual NIC of the source cloud VM. In cloud Dom0, the QEMU emulating the virtual NIC reads the data and

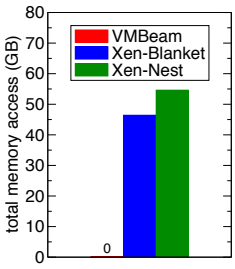
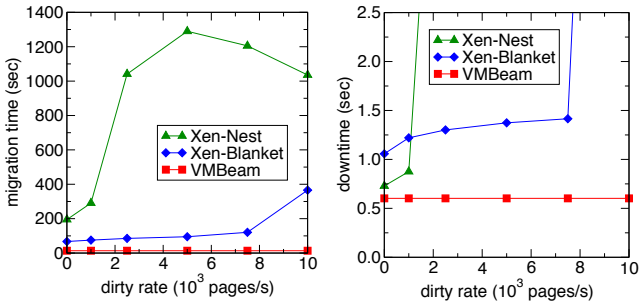


Figure 5. The estimation of the total memory access during VM migration.



(a) Migration time (b) Downtime

Figure 6. The migration performance under a memory-intensive workload.

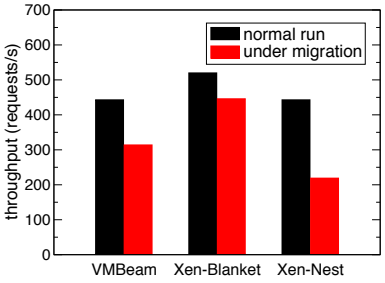


Figure 7. The degradation of web performance under VM migration.

sends it to the QEMU for the destination cloud VM. In the destination guest Dom0, the SSH server receives the data via the virtual NIC of the destination cloud VM. Then it decrypts the received data and sends the decrypted data to the migration server. Finally, the migration server writes it to the memory of a cloned guest VM. Therefore it is estimated that Xen-Nest needs 14 times as many memory accesses as the size of transferred memory in total.

Xen-Blanket slightly reduces the memory access by using a fast virtual network. When the SSH client sends data to the server as in Xen-Nest, the Blanket driver communicates with the network backend driver in the cloud Dom0. The data is passed using the grant table, which is a mechanism for memory sharing provided by the Xen hypervisor. Then the backend driver communicates with the Blanket driver in the destination guest Dom0. The guest Dom0 writes the data to a cloned guest VM as in Xen-Nest. Therefore it is estimated that the total memory access is 12 times as much as the size of transferred memory.

F. Memory-intensive Workload

To examine how memory writes in a guest VM affect the migration performance, we executed VM migration while running a memory-intensive workload in a guest VM. The workload performed writes to the memory of 1 GB at the rate between 1,000 and 10,000 pages per second. To keep the memory dirty rate constant, we emulated memory writes by directly rewriting the dirty bitmap, whose bit is set when the corresponding page is modified. Fig. 6 shows the migration performance of the guest VM when we allocated 2 GB of memory to it.

These results indicate that the migration time and the downtime in VMBeam were almost not affected by memory writes in the guest VM. This comes from no memory re-transfers in VMBeam even under frequent memory modifications. In Xen-Blanket, as expected, the migration time and downtime were proportional to the dirty rate. However, they became too large suddenly when the dirty rate was 10,000 pages per second. The migration time was 366 seconds, while the downtime was 15 seconds.

For Xen-Nest, the migration performance was worst when the dirty rate was 5,000 pages per second. The migration time was 1,290 seconds and the downtime was 102 seconds. The number of modified pages was kept too large during VM migration because the memory image was transferred via the slow virtual network. As a result, Xen-Nest had to re-transfer the large number of pages at the final stage after the iteration of memory re-transfers reached the pre-defined maximum.

G. Impact on Guest VMs

To examine how high system loads due to VM migration affects a guest VM, we ran the Apache web server in a guest VM and migrated the VM. We measured the throughput using the httperf benchmark under normal run and VM migration. We assigned 1 GB of memory to the guest VM. In this experiment, we assigned two CPUs to guest Dom0 to avoid the conflict between network processing for the web server and migration processing in it.

Fig. 7 shows that the throughput under VM migration degraded in all systems. The degradation was 29% in VMBeam and 51% in Xen-Nest. This is probably because the slow virtual network used by the web server became a bottleneck. On the other hand, the performance degradation was only 14% in Xen-Blanket. This is thanks to the fast virtual network provided by Xen-Blanket. Using the fast virtual network in VMBeam would suppress the performance degradation of a guest VM under VM migration.

V. Related Work

Zero-copy migration in this paper was initially proposed in our previous workshop paper [9]. In that paper, we used

zero-copy migration for lightweight software rejuvenation of the guest hypervisor and showed only preliminary results. When the guest hypervisor is aged, the proposed system starts a new virtualized system at the same host using nested virtualization and migrates all the guest VMs onto the new clean guest hypervisor using zero-copy migration. In this paper, we applied zero-copy migration to a different context, which is an optimization of VM migration in virtual IaaS clouds. In addition, we present comprehensive experiments on zero-copy migration.

RDMA-based migration over InfiniBand [10] needs only one copy by hardware, i.e., zero copy in software. The memory image of a VM is directly copied to the memory of a newly created VM at the destination host using Remote Direct Memory Access (RDMA). This migration method can avoid the processing overhead of the TCP/IP stack. However, it still needs to re-transfer memory modified during VM migration. In addition, it cannot encrypt the memory image although the data is transferred via the physical network. For encrypting the memory image, RDMA-based migration needs three copies.

To reduce system loads during VM migration, a technique for limiting a migration speed has been proposed [8]. This technique simply limits the network bandwidth used by VM migration and decreases the network load, resulting in decreasing the CPU and memory loads. However, this leads to longer migration time and downtime. In addition, the total system loads can increase in live migration. Since live migration has to re-transfer memory regions modified during VM migration, the total size of such memory regions can increase as the migration time increases. In contrast, zero-copy migration can achieve both low system loads and the high migration performance.

Many techniques for page sharing among VMs have been developed. VMware ESX Server scans the memory of VMs periodically and makes VMs share identical pages [11]. Satori [12] can detect short-lived sharing opportunities by using sharing-aware block devices. Difference Engine [13] shares not only identical but also similar pages between VMs. Flash cloning in Potemkin [14] creates a new VM from a reference VM image and enables page sharing between them. Since all of these techniques use copy-on-write, page sharing is ceased when shared pages are modified. In contrast, inter-guest memory sharing in VMBeam continues page sharing even for modified pages so that modifications are also shared between VMs.

VI. Conclusion

In this paper, we proposed zero-copy migration, which is an optimization of VM migration between co-located cloud VMs in virtual IaaS clouds. We have implemented VMBeam for enabling zero-copy migration in Xen and showed that zero-copy migration could achieve high migration performance and low system loads. Our future work is to transparently switch the traditional VM migration and zero-copy migration according to the destination. To enable this, we need a mechanism to obtain information on VM placement from the cloud hypervisor.

Acknowledgment

This research was supported in part by JSPS KAKENHI Grant Number JP16K00101.

References

[1] D. Williams, E. Elnikety, M. Eldehiry, H. Jamjoom, H. Huang, and H. Weatherspoon, "Unshackle the Cloud!" in *Proc. Workshop on Hot Topics in Cloud Computing*, 2011.

[2] D. Williams, H. Jamjoom, and H. Weatherspoon, "The Xen-Blanket: Virtualize Once, Run Everywhere," in *Proc. European Conf. Computer Systems*, 2012, pp. 113–126.

[3] C. Liu and Y. Mao, "Inception: Towards a Nested Cloud Architecture," in *Proc. Workshop on Hot Topics in Cloud Computing*, 2013.

[4] A. Fishman, M. Rapoport, E. Budilovsky, and I. Eidus, "HVX: Virtualizing the Cloud," in *Proc. Workshop on Hot Topics in Cloud Computing*, 2013.

[5] M. Ben-Yehuda, M. D. Day, Z. Dubitzky, M. Factor, N. Har'El, A. Gordon, A. Liguori, O. Wasserman, and B.-A. Yassour, "The Turtles Project: Design and Implementation of Nested Virtualization," in *Proc. Conf. Operating Systems Design and Implementation*, 2010.

[6] 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. Symp. Operating Systems Principles*, 2003, pp. 164–177.

[7] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, "Hey, You, Get off of My Cloud: Exploring Information Leakage in Third-party Compute Clouds," in *Proc. Conf. Computer and Communications Security*, 2009, pp. 199–212.

[8] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live Migration of Virtual Machines," in *Proc. Symp. Networked Systems Design and Implementation*, 2005, pp. 273–286.

[9] K. Kourai and H. Ooba, "Zero-copy Migration for Lightweight Software Rejuvenation of Virtualized Systems," in *Proc. Asia-Pacific Workshop on Systems*, 2015.

[10] W. Huang, Q. Gao, J. Liu, and D. K. Panda, "High Performance Virtual Machine Migration with RDMA over Modern Interconnects," in *Proc. Int. Conf. Cluster Computing*, 2007, pp. 11–20.

[11] C. A. Waldspurger, "Memory Resource Management in VMware ESX Server," in *Proc. Symp. Operating Systems Design and Implementation*, 2002, pp. 181–194.

[12] G. Miłós, D. G. Murray, S. Hand, and M. A. Fetterman, "Satori: Enlightened Page Sharing," in *Proc. USENIX Annual Technical Conf.*, 2009.

[13] D. Gupta, S. Lee, M. Vrable, S. Savage, A. C. Snoeren, G. Varghese, G. M. Voelkerrey, and A. Vahdat, "Difference Engine: Harnessing Memory Redundancy in Virtual Machines," in *Proc. Symp. Operating Systems Design and Implementation*, 2008, pp. 309–322.

[14] M. Vrable, J. Ma, J. Chen, D. Moore, E. Vandekieft, A. C. Snoeren, G. M. Voelker, and S. Savage, "Scalability, Fidelity, and Containment in the Potemkin Virtual Honeyfarm," in *Proc. Symp. Operating Systems Principles*, 2005, pp. 148–162.