

Towards Unobtrusive VM Live Migration for Cloud Computing Platforms

Akane Koto¹, Hiroshi Yamada¹, Kei Ohmura², and Kenji Kono¹

¹Keio University, ²NTT Software Innovation Center

{koto, yamada}@sslab.ics.keio.ac.jp, ohmura.kei@lab.ntt.co.jp, kono@ics.keio.ac.jp

Abstract

Live migration of virtual machines (VMs) is an attractive technique for managing cloud computing platforms such as those for load balancing and maintenance of physical machines. However, the execution of live migration significantly consumes computational resources on the source and destination, thus causing *migration noise* that degrades the performance of the VMs collocated on these hosts during live migration. Migration noise lowers the total throughput of data centers and make it difficult to carry out live migration to handle unpredictable increases in workload such as sudden request bursts and flash crowds. This paper introduces *SonicMigration*, which is an approach to shortening total migration time by pruning the VM memory to be transferred. Preliminary experiments with our prototype implemented on Xen 4.1.0 and Linux 2.6.38 show the migration time with our prototype is up to 68.3% shorter than that for Xen-based live migration and it successfully reduces network traffic by up to 83.9%.

1. Introduction

Cloud computing platforms allow users to host their applications on a huge number of globally shared resources. Cloud service providers manage computational resources at data centers and offer users re-

sources in a pay-as-you-go manner. To manage the resources efficiently, cloud service providers commonly employ system virtualization technology where several virtual machines (VMs) coexist on the same physical host. Using the virtualization, they easily adjust resource allocations by changing the number of running VMs and balance the loads by migrating VMs across physical machines. For example, Amazon EC2 [1] manages more than ten virtualized data centers and rents various types of VM instances.

The live migration of VMs is a promising technique for managing cloud computing platforms. Live migration moves a running VM between different physical machines without losing any states such as network connections. The use of live migration makes it more effective to manage resources. For example, the availability of services can be improved by migrating less loaded VMs to another host to assign resources to more loaded VMs or by migrating VMs from a host to be maintained. VM replacement policies using live migration, including load balancing [5, 14] and power saving [4, 13], have been widely studied in research communities.

However, live migration causes *migration noise*, which interferes with the performance of VMs running on the source and destination when a VM is migrated live. Since live migration transfers the whole memory pages of a VM from the source to the destination to maintain its running states, the execution of live migration consumes CPU time on the source and network bandwidth, both of which are shared among the collocated VMs. In addition, live migration consumes CPU time on the destination since the virtual machine monitor (VMM) running on it receives the transferred pages and sets up the VM to run it. As a result, migration noise happens due to resource contention between live

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

APSys '12, July 23-24, 2012, Seoul, S. Korea

Copyright 2012 ACM 978-1-4503-1669-9/12/07... \$15.00

migration and the collocated VMs. For example, when a VM becomes heavily loaded, the live migration of another collocated VM to assign more resources to the heavily loaded VM degrades its performance. Although we can mitigate migration noise by using a function that throttles page transfers [3], this increases migration time, thus failing to readily allocate the resources of the migrating VM to the heavily loaded VM.

The live migration of VMs is ineffective and sometimes unacceptable due to migration noise, resulting in inefficient data center operations. Even if sophisticated VM replacement policies [4, 5, 13, 14] output better VM placements, administrators cannot migrate VMs easily since migration noise interferes with VMs and it can violate their service-level agreements. As a result, administrators cannot fully benefit from the previous work. Although we can avoid migration noise by inferring the coming workloads and migrating VMs in advance, this is difficult in practice because an increase in requests suddenly occurs and is unpredictable, e.g. sudden request bursts and flash crowds. This unpredictability forces us to undertake reactive handling where we need to manage such sudden request increases after they happen.

In this work, we focus on the total execution of live migration to mitigate migration noise, not only on the downtime that has been studied in most prior research. This paper introduces *SonicMigration*, which is an approach to shortening the total execution of live migration by pruning the VM memory to be transferred. The key insight behind *SonicMigration* is to avoid transferring *soft pages* that are unnecessary for the system to work correctly after live migration, such as file cache pages and free pages. *SonicMigration* examines memory pages when live migration is triggered, and marks soft pages to avoid transferring them. Reducing the number of transferred pages leads to shortening the total migration time, which means *SonicMigration* consumes less CPU time on the source and destination. Moreover, the reduction decreases the network usage of live migration.

We implemented a prototype of *SonicMigration* on Xen 4.1.0 and Linux 2.6.38, and conducted a preliminary experiment. The experimental results demonstrate that our prototype shortens the total migration time by up to 68.3 %, compared with Xen-based live migration. The results also reveal that *SonicMigration* successfully reduces network traffic by up to 83.9 %.

2. Effect of Migration Noise

2.1 Experimental Setup

We conducted an experiment to examine how migration noise interferes with collocated VMs. We set up three Dell machines each of which had a Xeon 2.8 GHz processor with 29 GB of memory. They were connected to one another through a Gigabit Ethernet. We ran Xen 4.1.0 and Linux 2.6.38 as domain 0 on two of the machines. We also ran Linux 2.6.35 on the third machine that was used as an NFS server in which domain disk images were stored. The directory containing the disk images was mounted by the other machines.

In this experiment, we ran two domain Us on the one machine running the Xen, each of which executed para-virtualized Linux 2.6.38. They were assigned 2 GB of memory and a 20 GB virtual disk. On the domain Us, we ran a postmark benchmark that was modeled after an e-mail server. We measured the throughput of the postmark. To build a situation where a domain gets heavily-loaded unpredictably, we changed a parameter of the postmark benchmark on a domain U (*domU1*) after the postmark benchmarks had run for about 10 seconds. After changing the parameter, we executed the Xen-based pre-copy live migration of the other domain (*domU2*). To clearly understand the effect of migration noise, we tuned the parameter for the postmark to perform CPU intensive tasks. *DomU1* first handled 5,000 transactions per second and then 11,000 transactions per second after the parameter was changed, while *domU2* constantly handled 7,000 transactions per second. For comparison, we measured the throughput of the postmark under two conditions where *domU1* was run solely, and where the domain Us were run together without live migration.

2.2 Results

The results are plotted in Fig. 1. The x-axis is the elapsed time and the y-axis is the throughput for the postmark. These figures indicate that live migration severely degrades the throughput of the collocated domain. When we execute *domU1* solely, it successfully handles the requested workload even after the parameter is changed (Fig. 1)(a). This is because the Xen hypervisor assigns *domU1* enough CPU time that it can handle the workloads. When we run *domU1* with *domU2*, *domU1* only performs 7,500 transactions per second after the parameter is changed (Fig. 1(b)). This results from the CPU contention between *domU1* and

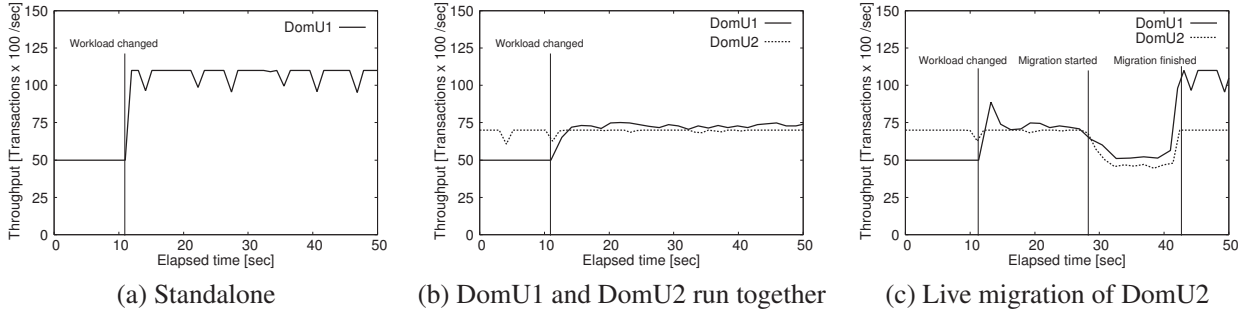


Figure 1. Throughputs of postmark in each situation.

domU2, as will be described later. From Fig. 1(c), the domain Us’ throughputs are significantly lowered during live migration. The throughputs are similar to those in Fig. 1(b) until live migration starts. During live migration, the domain Us’ throughput is 30% lower due to migration noise. After live migration is complete, domU1 has similar throughput to when it is run solely.

Fig. 2 shows CPU usage on the source and destination. The figures reveal that live migration causes CPU contention with domU1 and domU2. When we carried out live migration at 28 second, domain 0 starts consuming CPU time to transfer memory pages and processor states. Each domain U’s CPU usage is naturally lowered during this phase since the CPU is contended by the three domains. As a result, each domain Us’ throughput is severely degraded. Surprisingly, the CPU usage on the destination is high during live migration. This is because the Xen on the destination receives transferred pages and set up data structures for the VM. We can say that migration noise even affects domain Us running on the destination.

3. SonicMigration

3.1 Overview

The key insight behind SonicMigration is to avoid transferring pages that are unnecessary for the system to work correctly after live migration is completed. We refer to these pages as *soft pages*. Conventional live migration transfers all the memory pages of a VM even if the pages are not used for the kernel or user processes. If a VM is assigned 1024 MB of memory, the VMM transfers all the memory to the destination. In contrast, SonicMigration does not copy soft pages such as free pages or file cache pages. For example, if a VM is assigned 1024 MB of memory and the guest kernel uses

928 MB as a file cache, SonicMigration transfers 72 MB since the file cache is soft-state and can be reproduced from the disk.

We found that soft pages include a free page and a page containing soft-state kernel objects. Even if a VMM discards the content of free pages, the guest works correctly because free pages are initialized when the kernel uses them. Soft-state kernel objects include caches for disk blocks and caches for kernel resource managers. A file cache is a typical example of soft-state kernel objects. Since a file cache contains data on a disk, the guest can reproduce it by reading the data from the disk. Likewise, caches for resource managers such as a slab cache can be reproduced from the original data objects. We regard as hard-state kernel objects file caches that are in use. Such caches include caches that are marked as dirty, and caches that are locked since the process is changing its state.

To prevent soft pages from being transferred, we insert an extension into the guest kernel to explicitly notify the VMM about which pages are unnecessary. When live migration is carried out, the guest kernel examines its memory objects and sends the VMM the guest physical addresses of the soft pages. The VMM does not send them to the destination, based on the given addresses. When the transfer of the pages and processor states is complete, the VMM compensates for the lost pages by allocating new pages to the guest. After that, the VM on the destination is resumed.

3.2 Design

There are two main issues in designing SonicMigration. The first is how the guest kernel sends the addresses of soft pages to the VMM. Since the kernel changes its memory objects over time, objects on soft pages can become hard-state. If the guest kernel does

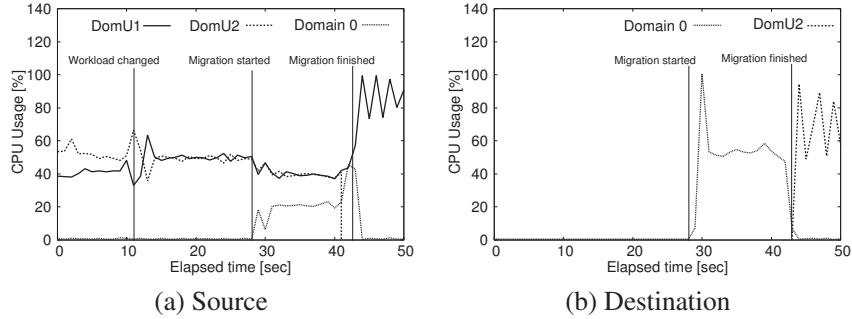


Figure 2. CPU usage of each domain.

not update the information on soft pages in the VMM appropriately, the VMM could fail to transfer pages in which the hard-state objects are stored. Although the guest kernel can issue a hypercall to update the information when the kernel objects on soft pages are modified, this incurs high CPU overhead, which interferes with the VMs on the source. To address this issue, we create a shared memory area between the guest and VMM. The kernel stores and periodically updates the guest physical addresses of the soft pages in the area.

The second issue is how to keep the kernel state consistent with the lost soft-state kernel objects when the VM is resumed. Since SonicMigration does not transfer soft pages, the guest kernel resumed on the destination does not have soft-state kernel objects. If kernel data for managing kernel objects, such as a page cache list, are not updated appropriately and remain inconsistent with the current objects, the guest kernel causes access to the soft-state kernel objects that have not been transferred. The kernel in this case could fetch incorrect objects, resulting in an inconsistent state in the kernel. At worst, the kernel will crash. To address this issue, we update the kernel data for managing soft-state kernel objects in the phase where the VM is suspended and processor states are transferred.

3.2.1 Shared Memory for Address Notification

To efficiently notify a VMM about which pages are soft pages, the guest kernel creates a shared memory area with the VMM. SonicMigration uses this information as a *hint* to select which pages are necessary to transfer. When booting up, a guest kernel sends the VMM a request to create the shared memory area. It periodically stores the guest physical addresses of the soft pages into the shared memory area. The VMM refers

to this area at the beginning of the page copy phase such as an iterative copy phase and marks pages that are not to be transferred. If the notified soft pages are updated and dirtied, the VMM copies the dirty pages to the destination in the current or next page copy phase.

Note that this design releases us from strictly notifying the VMM of the addresses of soft pages. SonicMigration transfers dirty pages even if the pages are soft pages. By doing so, SonicMigration guarantees that pages containing the hard-state kernel objects are copied to the destination. The guest kernel can be resumed with the consistent state by combining this notification and the update of the kernel state, which is described in the next section.

3.2.2 Kernel Data Update

To keep the kernel state consistent with the lost soft-state kernel objects, the kernel data for managing soft-state kernel objects are updated in the VM-suspend phase where the VMM suspends the VM and transfers processor states. By updating in this phase, we guarantee that the guest kernel can run at the destination with a consistent state. For example, the kernel in the pre-copy approach updates the data in the stop & copy phase. Since the kernel starts at the destination with the kernel data that indicate there are no soft-state kernel objects, it consistently restarts on the destination. Since this design incurs some overhead in the VM-suspend phase, we are now analyzing this overhead and exploring a way of minimizing it.

The guest kernel updates the kernel data cooperatively with the VMM. When live migration enters the VM-suspend phase, the VMM sends a virtual interrupt to the VM before suspending it. The guest kernel receiving the interrupt updates the kernel data for man-

aging the page cache and free pages. Then, the guest issues a hypercall for the VMM to start performing subsequent instructions including VM suspension and transfer of processor states.

4. Preliminary Experiments

4.1 Experimental Setup

We implemented a prototype by modifying para-virtualized Linux 2.6.38 and Xen 4.1.0. We conducted preliminary experiments to examine the basic performance of our prototype. In these experiments, we used the same machine configuration as described in Sec. 2. Our prototype ran on two machines running Xen 4.1.0.

To confirm the basic characteristics of SonicMigration, we measured (1) the total migration time and (2) network traffic of SonicMigration and Xen-based live migration (simply called *default migration*). We ran domain U which had 2 GB of memory and a 20 GB disk. We first read 2 GB files to fill the buffer cache and then varied memory usage at the user-land (i.g., hard-state kernel objects) from 256 to 2000 MB. After these preparations, we conducted live migration of the domain U with SonicMigration and default migration.

4.2 Results

Fig. 3 shows the results. The migration times are given in Fig. 3(a). The x-axis is the memory size we allocated at the user-land, and the y-axis is the migration time. Fig. 3 reveals that SonicMigration’s migration time shortens as the usage of soft state kernel objects increases. When we do not allocate any memory at the user-land, SonicMigration’s migration time is 68.3% shorter than that of the default. The migration time of SonicMigration is almost the same as that of the default for 2000 MB.

Fig. 3(b) shows the network traffic of each migration. The x-axis is the memory size we allocated at the user-land, and the y-axis is the number of pages that was transferred during the migration. From the figure, we can see that SonicMigration successfully reduces the network traffic. The reduction in the number of memory pages depends on our allocation. When our allocation is 0, SonicMigration reduces 83.9 % fewer pages than default migration. This is because SonicMigration prunes a large part of memory that is used as the buffer cache. SonicMigration’s network traffic is quite similar to that of the default for 2000 MB.

5. Discussion

Our approach is to mitigate migration noise at the expense of losing soft-state kernel objects on the destination. Since the guest kernel cannot access page caches stored on the source, the performance of resumed VM can be degraded. We are exploring ways of adjusting how many soft pages SonicMigration discards. For example, we transfers some soft pages which are significantly referred to. Considering not only the collocated VMs’ performance but also that of the migrating VM is a considerable challenge to confront.

If the guest kernel is hijacked, SonicMigration can fail to achieve live migration. For example, when the hijacked kernel marks all the pages as soft pages in the shared memory, SonicMigration does not transfers pages that are not dirtied during the page copy phase such as the iterative copy phase. In this case, the VM could crash on the destination since hard-state kernel objects are not transferred. Although the hijacked VM is compromised by our modules being attacked, SonicMigration guarantees isolation between VMs. This means that the other VMs are not compromised even if a VM is hijacked and our modules are attacked. We intend to explore ways of protecting against these types of attacks in future work.

6. Related Work

Some approaches aim at reducing the network traffic of the live migration. MECOM [7] compresses pages using a characteristic-based compression algorithm on the source and decompresses them on the destination. Delta compression live migration [12] caches some frequently accessed pages, creates the delta calculated from the cached pages, compresses the delta using an XOR, and sends the compressed data to the destination. The destination decompresses the sent data using the pages which have already been transferred. These approaches are complementary to SonicMigration to mitigate migration noise. CR/RT-Motion [9] transfers execution trace logs to the destination and creates the same state VM by replaying execution based on the trace logs. This approach causes severe migration noise since logging and replaying the execution consume significant CPU time. SonicMigration focuses on mitigating migration noise.

There are researches that avoid the migration noise using good models of migration execution. Breitgand et al. model the costs of pre-copy live migration to define

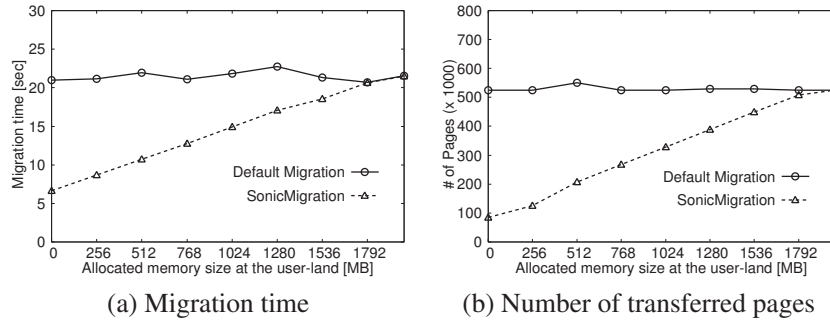


Figure 3. Migration time and network traffic.

the amount of bandwidth to be used for migration in each step of the pre-copy approach [2]. Lim et al. model a migration process as a pair of jobs that run on the source and destination [8]. These models are effective for well-known workloads and thus make it difficult to handle unpredictable workloads such as sudden request bursts and flash crowds. Our approach is to modify the mechanism of live migration itself to make live migration feasible even under such workloads.

To improve the functionality offered by VMMs such as snapshots and VM replication, previous approaches reduce the number of VM pages. Park et al. [11] demonstrate a technique of taking a checkpoint with only a subset of the VM’s memory. The technique tracks the I/O operations of the guest to external storage to avoid dealing with pages whose content is stored in non-volatile storage. RemusDB [10] is a VM replication system tailored for database systems. It employs a similar technique that tracks disk read requests to detect pages whose content is on the disks. However, these techniques cannot detect all types of soft-state kernel objects since some soft-state kernel objects such as slab caches do not perform any I/O when they are created. SonicMigration is a para-virtualized approach that allows us to completely know the kernel running states and manipulate the kernel objects more aggressively.

7. Current status

We are conducting experiments with various workloads to confirm how effectively SonicMigration mitigates migration noise. In addition, we are analyzing SonicMigration behavior in greater detail to know which module consumes more resources. After these experiments, we intend to extend SonicMigration to make

live migration more unobtrusive. We also consider ways of adjusting how many soft pages SonicMigration discards, as described in Sec. 5.

References

- [1] EC2: Amazon Elastic Compute Cloud. <http://aws.amazon.com/ec2/>.
- [2] D. Breitgand, G. Kutieli, and D. Raz. Cost-Aware Live Migration of Services in the Cloud. In *Proc. of Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services*, Mar. 2011.
- [3] 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. of the 2nd USENIX Symp. on Networked Systems Design and Implementation*, pages 273–286, May 2005.
- [4] S. Govindan, D. Wang, A. Sivasubramaniam, and B. Urgaonkar. Leveraging Stored Energy for Handling Power Emergencies in Aggressively Provisioned Datacenters. In *Proc. of the 17th ACM Int’l Conf. on Architectural Support for Programming Languages and Operating Systems*, pages 75–86, Mar. 2012.
- [5] F. Hermenier, X. Lorca, J.-M. Menaud, G. Muller, and J. Lawall. Entropy: a Consolidation Manager for Clusters. In *Proc. of the 2009 ACM Int’l Conf. on Virtual Execution Environments*, pages 41–50, Mar. 2009.
- [6] M. R. Hines and K. Gopalan. Post-Copy Based Live Virtual Machine Migration Using Adaptive Pre-Paging and Dynamic Self-Ballooning. In *Proc. of the 2009 ACM Int’l Conf. on Virtual Execution Environments*, pages 51–60, Mar. 2009.
- [7] H. Jin, L. Deng, S. Wu, X. Shi, and X. Pan. Live Virtual Machine Migration with Adaptive Memory Compression. In *Proc. of 2009 IEEE Int’l Conf. on Cluster Computing*, pages 1–10, Sep. 2009.
- [8] S.-H. Lim, J.-S. Huh, Y. Kim, and C. R. Das. Migration, Assignment, and Scheduling of Jobs in Virtualized Environment. In *Proc. of the 3rd USENIX Workshop on Hot Topics in Cloud Computing*, Jun. 2011.
- [9] H. Liu, H. Jin, X. Liao, L. Hu, and C. Yu. Live Migration of Virtual Machine Based on Full System Trace and Replay. In *Proc. of the 18th ACM Int’l Symp. on High Performance Distributed Computing*, pages 101–110, Jun. 2009.
- [10] U. F. Minhas, S. Rajagopalan, B. Cully, A. Aboulmaga, K. Salem, and A. Warfield. Remusdb: Transparent high availability for database systems. *PVLDB*, 4(11):738–748, 2011.
- [11] E. Park, B. Egger, and J. Lee. Fast and Space-Efficient Virtual Machine Checkpointing. In *Proc. of the 7th ACM Int’l Conf. on Virtual Execution Environments*, pages 75–86, 2011.
- [12] P. Svård, B. Hudzia, J. Tordsson, and E. Elmroth. Evaluation of Delta Compression Techniques for Efficient Live Migration of Large Virtual Machines. In *Proc. of the 7th ACM Int’l Conf. on Virtual Execution Environments*, pages 111–120, Mar. 2011.
- [13] A. Verma, P. Ahuja, and A. Neogi. pMapper: Power and Migration Cost Aware Application Placement in Virtualized Systems. In *Proc. of the 9th ACM/IFIP/USENIX Int’l Conf. on Middleware*, pages 243–264, Dec. 2008.
- [14] T. Wood, A. Venkataramani, and M. Yousif. Black-box and Gray-box Strategies for Virtual Machine Migration. In *Proc. of the 4th USENIX Symp. on Networked Systems Design and Implementation*, pages 229–242, Apr. 2007.