Vol.2012-OS-122 No.13
2012/8/1

# 仮想マシン移送における移送ホストの負荷軽減手法

古藤 明音[1,a)]　山田 浩史[1,b)]　大村 圭[2,c)]　河野 健二[1,d)]

**概要**：クラウド環境でロードバランシングや物理マシンのメンテナンスを行う技術の１つに，仮想マシン (VM) のライブマイグレーションがある．しかし，既存のライブマイグレーション手法は動作の度に VM の移送元と移送先の計算リソースを著しく消費する．その結果，同一ネットワーク内のシステムのスループットを低下させる要因 (マイグレーションノイズ) となっている．こうしたマイグレーションノイズにより，サーバが急激な負荷に追従した負荷分散を行えなくなったり，同一ネットワークを利用している他の VM のネットワークを介した処理のパフォーマンスが低下してしまう．そこで本研究ではライブマイグレーションを高速化するだけでなくマイグレーションノイズにも注目し，移送する VM のサイズを小さくすることで移送にかかる時間を短縮するライブマイグレーション手法を提案する．具体的には移送する VM の転送ページ数を減らすことで，マイグレーション中のネットワーク使用量や CPU 時間などを削減する．本提案手法を Xen 4.1.0 と Linux 2.6.38 上に実装した．実験により，既存のライブマイグレーション手法に比べて，移送にかかる時間を 68.8% ，ネットワーク使用量を 89.3% 削減することができた．

***Abstract:*** Live migration of virtual machines (VMs) is a powerful tool for the management of cloud computing platforms such as load balancing and physical machine maintenance. However, live migration execution significantly consumes computational resources on the source and destination hosts, thus causing a *migration noise* that degrades the performance of the VMs collocated on these hosts. This paper presents *SonicMigration*, an approach to shortening total migration time by pruning the VM memory to be transferred. It avoids transferring pages that are unnecessary for the system to work correctly after the live migration, such as file cache pages and free pages, in order to lower consumption of the shared resources such as CPU time and network traffic. We implemented a prototype of SonicMigration on Xen 4.1.0 and Linux 2.6.38, and conducted a preliminary experiment. The experimental results show that the migration time of our prototype is up to 68.3% shorter than the Xen-based live migration and successfully reduces the network traffics by up to 89.3%.

## 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 resources 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], [12] and power saving [4], [11], 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 col-

[1] 慶應義塾大学
[2] NTT ソフトウェアイノベーションセンタ
[a)] koto@ics.keio.ac.jp
[b)] yamada@ics.keio.ac.jp
[c)] ohmura.kei@lab.ntt.co.jp
[d)] kono@sslab.ics.keio.ac.jp

located 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 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.

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. Migration Noise

Since live migration transfers a large number of memory pages, its execution causes *migration noise* that degrades the performance of the VMs collocated on the source and destination. Live migration transfers at least all pages of the VM in the iterative copy phase. As mentioned in Sec. 1, transferring pages consumes significant network bandwidth and CPU time on the source and destination. This consumption is more severe when the VM has more memory. When live migration causes resource contention with the collocated VMs, migration noise occurs. For example, when a VM becomes heavily loaded and we carry out live migration of another VM to allocate more resources to the heavily-loaded VM, its performance is degraded during live migration. Moreover, resource contention increases the total migration. Long migration time prevents the VMM from readily releasing the resources of the migrating VM and from allocating them to other VMs just after they need more resources.

### 2.1 Effect of Migration Noise

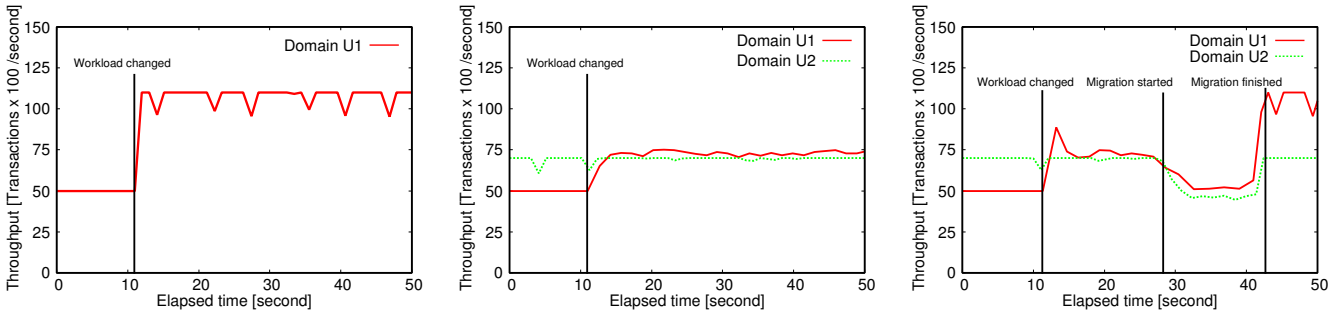
#### 2.1.1 Experimental Setup

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 precopy 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.

#### 2.1.2 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 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




(a) Standalone (b) DomU1 and DomU2 run together (c) Live migration of DomU2

**図 1 Throughputs of postmark in each situation.**

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

We believe that it is important to mitigate migration noise on cloud computing platforms where resources are shared with different VMs. This paper introduces *SonicMigration*, which is an approach to mitigating migration noises. The goal of SonicMigration is twofold, i.e., (1) shortening the total execution of live migration to reduce CPU usage, and (2) cutting network traffic caused by live migration.

### 3.1 Overview

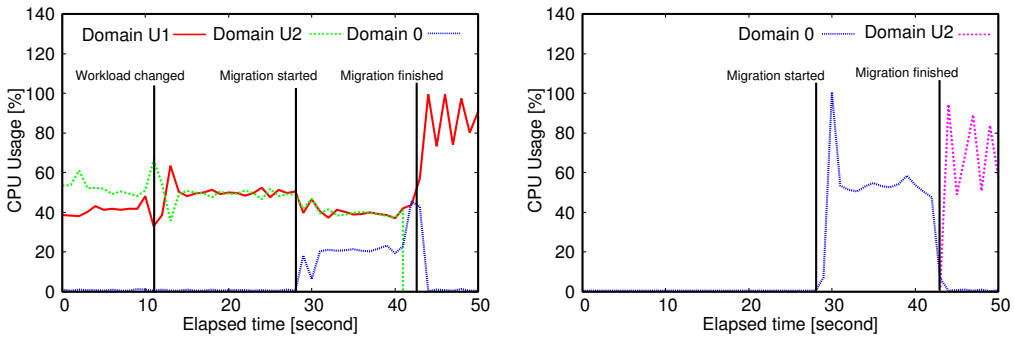
The key insight behind SonicMigration is to avoid transferring pages that are unnecessary for the system to work correctly after live migration is completed. We refer to these pages as *soft pages*. Conventional live migration transfers all the memory pages of a VM even if the pages are not used for 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 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




(a) Source (b) Destination

**図 2 CPU usage of each domain.**

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 Update Kernel Data

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 managing 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.

### 3.3 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.

## 4. Preliminary Experiments

We implemented a prototype by modifying para-virtualized Linux 2.6.38 and Xen 4.1.0. Our prototype consists of three modules; kernel, migration, and VMM modules. The kernel module running on Linux creates a shared memory area with domain 0 which carries out live migration. And, the kernel module writes the information on soft pages in the shared memory area. It also updates the kernel states when receiving an event from the Xen hypervisor. After the update, the kernel module issues a hypercall to transfer control to domain 0. The migration module running on domain 0 obtains the guest physical addresses of soft pages from the shared memory area to prevent soft pages from being transferred. When the migration process enters the stop & copy phase, the migration module issues a hypercall to send an event to migrating domain U. The VMM module running on the Xen hypervisor appropriately handles hypercalls from domain 0 and domain U.

### 4.1 Experimental Setup

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.e., 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

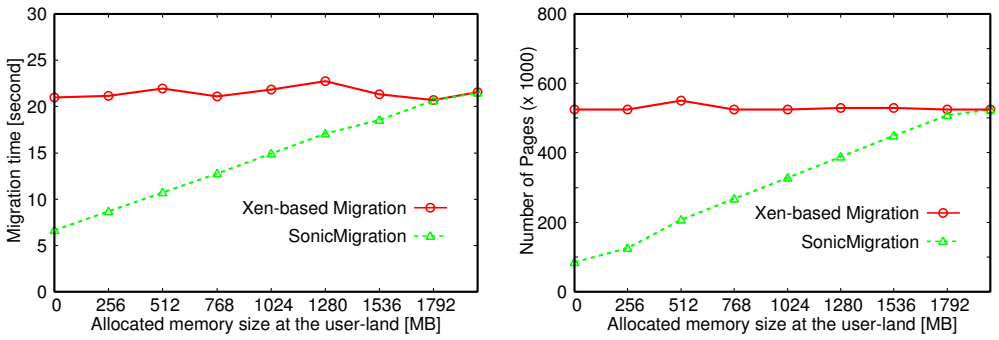
Fig. 3 shows the results. The migration times are given in Fig. 3(a). The x-axis is the memory size we allocated at the user-land, and the y-axis is the migration time. Fig. 3 reveals that SonicMigration's migration time shortens as the usage of soft state kernel objects increases. When we do not allocate 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. Related Work

Approaches to shortening downtime during live migration have been extensively studied. The pre-copy approach [3] transfers memory pages iteratively and subsequently copies the processor states. As described in Sec. 2, the pre-copy approach causes significant migration noise due to the transfer of a large number of memory pages. The post-copy approach [6] reduces the number of pages to be transferred by first transferring the processor states and copying the memory pages when the VM accesses them. However, the post-copy approach has to retain the memory pages of the migrating VM on the source until the transfer of all pages is complete, failing to readily allocate more memory to the heavily loaded VM. The focus of our work is on mitigating migration noise. In addition, our technique can be applied to both techniques and can be complementary to their use.

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 [10] 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.




(a) Migration time (b) Number of transferred pages

**図 3 Migration time and network traffic usage.**

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 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.

## 6. Conclusion

Live migration is an attractive technique for managing cloud computing platforms. However, live migration is unobtrusive since its execution consumes significant computational resources. As a result, it can cause migration noise that degrades the performance of the collocated VMs due to resource contention. This paper introduced SonicMigration, an approach to shortening the total migration time by pruning the VM memory to be transferred. It avoids the transfer of soft pages that are unnecessary for the system to work correctly after live migration, such as file cache pages and free pages. Our preliminary experimental results indicate that the total migration time of the prototype is 68.3 % shorter than that of Xen-based live migration. Moreover, the network usage by SonicMigration is 83.9% lower than that of Xen-based live migration. We are now 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.

## 参考文献

[1] EC2: Amazon Elastic Compute Cloud. http://aws.amazon.com/ec2/.

[2] David Breitgand, Gilad Kutiel, and Danny 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] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew 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] Sriram Govindan, Di Wang, Anand Sivasubramaniam, and Bhuvan 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] Fabien Hermenier, Xavier Lorca, Jean-Marc Menaud, Gilles Muller, and Julia 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] Michael R. Hines and Kartik Gopalan. Post-Copy Based Live Virtual Macine Migration Using Adaptie 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] Hai Jin, Li Deng, Song Wu, Xuanhua Shi, and Xiaodong 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] Seung-Hwan Lim, Jae-Seok Huh, Youngjae Kim, and Chita 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] Haikun Liu, Hai Jin, Xiaofei Liao, Liting Hu, and Chen 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] Petter Svärd, Benoit Hudzia, Johan Tordsson, and Erik 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 Excution Environments*, pages 111–120, Mar. 2011.

[11] Akshat Verma, Puneet Ahuja, and Anindya 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.

[12] Timothy Wood, Arun Venkataramani, and Mazin 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.