A Fast Virtual Machine Storage Migration Technique Using Data Deduplication

Kazushi Takahashi and Koichi Sasada
Graduate School of Information Science and Technology
The University of Tokyo
Dai-building 1301, Sotokanda 1-18-13,
chiyoda-ku, Tokyo, Japan 101-0021
Email: {kazushi, ko1}@rvm.jp

Takahiro Hirofuchi
National Institute of Advanced Industrial
Science and Technology (AIST)
Central 2, Umezono 1-1-1, Tsukuba, Japan 305-8568
Email: t.hirofuchi@aist.go.jp

***Abstract*—In this paper, we proposed a fast virtual machine (VM) storage migration technique. Virtual machine storage migration can migrate entire VM states to other hosts, including VM disk images. It is widely used for cross-data-center load management. However, VM disk images generally have large file sizes (typically 1-30GB). Thus, storage migration was time-consuming. To address this problem, we introduce the deduplication technique into traditional VM storage migration. We focused on the fact that it is possible to return VMs from the new VM hosts. For example, a VM first migrates from host A to host B. Next, the VM returns from host B to host A. There will then be additional reusable disk pages in host A. Consequently, to expedite the operation, we only return to host A the disk pages that have been updated while in host B. We implement this idea to a QEMU/KVM (kernel virtual machine). To track the reusable disk pages, we developed a DBT (Dirty Block Tracking) mechanism and a new diff format to store the tracking result. In this paper, we discuss the design and implementation of our prototype. Our technique successfully reduced the transfer time for storage migration from 10 minutes to about 10 seconds in some practical workloads.**

Keywords**-** *Virtual Machine Monitor; VMM; VM Live Migration; VM Storage Migration*

I. Introduction

Live migration involves the migration of virtual machines (VMs) from one physical host to another even while the VM continues to execute. Live migration has become a key ingredient for data center management activities such as load balancing, failure recovery, and system maintenance.

Recently, a new live VM migration mechanism has been developed. Live VM (virtual machine) storage migration allows us to move entire VM states, including VM disk images to another physical machine without stopping the VM. Traditional VM live migration mechanisms transport only machine states such as CPU registers and NIC registers, and require file sharing systems such as NFS [1] and iSCSI, to share VM disk images between the source and destination hosts. However, VM storage migration achieves VM live migration without the file sharing systems.

VM storage migration enables flexible VM deployment across physical computers. First, VM storage migration is widely used in many data centers because it does not require file sharing systems. For example, cross-data-center load management, and VMs can evacuate quickly to other data centers in other regions when the data center is unavailable due to maintenance. Second, we believe that storage migration will be used for personal computer environments in future. Some researchers [2] have studied virtual machine migration for personal computing. However, since they used traditional virtual machine migration, as opposed to storage migration, the system forces users to use file sharing systems that have network connections. However, by introducing VM storage migration, VMs become portable without the need for file sharing systems. VM storage migration has significant potential for future computing.

However, VM disk images are large (typically 1-30GB in size) and VM storage migration is time-consuming. Even when using fast gigabit network environments, the transfer time was significant. For example, a VM which has a disk size of 20 GB requires about 10 minutes in a 1Gbps LAN environment. This is unacceptable by users because it is inconvenient.

We proposed a fast VM storage migration mechanism using *data deduplication* to reduce the transfer time and to reduce the volume of transferred data. Our fast VM storage migration works as follows: assume that two physical hosts, A and B, are located in different regions. Initially, a VM is migrated from source host A to destination host B. Thereafter, the VM returns to A from B. We can then leverage disk pages which were not updated while in host B to reduce the transfer time and the volume of transferred data. Although the initial transfer cost from A to B is large, VM migration will be faster in the subsequent round. Consequently, we can achieve faster VM storage migration than traditional VM storage migration techniques.

We implemented this idea to a QEMU/KVM (kernel virtual machine) to develop a prototype. QEMU/KVM already has a storage migration mechanism. Thus, we improved the storage migration mechanism to support data deduplication by tracking all disk-writing operations on the disk by the VM, and to identify the disk pages that are reusable. To track the disk pages that are reusable, we developed a *DBT (Dirty Block Tracking)* mechanism and new diff format to store the tracking result.

We examined our prototype on several machines which have difference workloads. After developing our prototype, we compared our storage migration with traditional storage migration to determine the difference in the volume of data transferred and time taken are reduced. Our result shows the effectiveness of our deduplication mechanism for fast storage migration.

This paper is based on our previous work which has been presented at a local symposium in Japan (written in Japanese) [3] In this paper, we have substantially improved our previous work by conducting further experiments and making more detailed discussions which is mentioned in Section V.

II. Related Work

There have been related work that enables VM live-migration with VM disk images. Studies by Luo et al. [4] and Bradford et al. [5] enable VM storage migration in Xen [6] using their special back-end drivers, and they achieve VM live-migration without a file sharing system. However, they did not discuss re-using disk pages, and if there are reusable disk blocks in the destination host, the hypervisor can perform data deduplication to reduce the volume of data and time. Therefore, their research is somewhat different from ours. Hirofuchi et al. [7] implemented a NBD (Network Block Device) protocol server. In this system, a user use `/dev/nbd0` to enable VM storage migration without the file sharing system. When the user wants to boot a VM, he executes the VM with `/dev/nbd0` instead of a normal VM storage file. When VM storage migration occurs, `/dev/nbd0` copies the disk image to the destination host. This is also different from our approach since we focus on the application of data deduplication to reduce transfer data and time while Hirofuchi did not discuss data deduplication for VM storage migration, as is the case with our approach.

VMFlocks Project [8] proposed a data deduplication mechanism for VM storage migration between data centers. This project was implemented as a user-level file system on a host OS. This user-level file system inspects VM disk images without hypervisors, and it deduplicates data disk blocks of VM disk images using fingerprint algorithms such as SHA-1 [9] Our research is different in that we implement our deduplication mechanism by modifying a hypervisor. On the other hand, VMFlocks is implemented as a user-level file system on a host OS. Our approach using the DBT within a hypervisor is beneficial since it can leverage raw level hardware commands such as the ATA TRIM command to optimize disk pages, For example, garbage collection for disk pages.

Sapuntzakis et al. [10] proposed several techniques for speeding up virtual machine migration in various user scenarios. More specifically, they proposed a tree structured based VM disk management system as follows: First, a user creates a root VM storage image in a destination host. Second, the user checks out a VM image from the root VM storage image on a source host. Finally, the approach reduces the amount of data transferred by exploiting similarities between the transferred image on the source host and the root image on the destination host. Our proposal is different since we use a simpler approach in which the DBT records dirty block information into a simple dirty map in order to reduce the amount of transferred data.

Intel Research proposed ISR (Internet Suspend/Resume) techniques [11] [12] ISR is a cold VM migration technique and is explained as follows: First, before a VM transfer takes place, a user suspends the VM. Next, the suspended VM image that includes a VM disk image migrates to a destination host. Finally, after migrating the VM image, the VM resumes in the destination host. Kozuch et al. [11] and Tolia et al. [12] proposed a VM storage migration technique using Coda [13], which is a traditional distribution file system. To enable VM storage migration, they store the entire VM status (including VM disk images) to Coda, and frequently download the VM status on the destination host on demand. In fact, this approach supports VM cold migration. Coda has an excellent caching and reading prediction mechanism. Thus, a user can eventually obtain the entire VM status without having a network connection to a Coda server. However, a constant network connection is required to access the Coda file system until file caching is filled. On the other hand, our approach only requires a temporary network connection while transferring the VM. Also, our approach supports VM live-migration.

VMware's VMware Storage VMotion [14][15] supports VM migration including VM file images. Unlike our approach, they do not leverage re-usable disk pages on a destination host to reduce transferred data or time.

The Shrinker project [16] focuses on reducing the transferred data to minimize transfer time. To reduce the transfer time, they share VM memory pages between a destination host and a source host using a distributed hash table (DHT). They focus on using memory pages to reduce migration time while we focus on disk pages.

III. Motivation

In this paper, we proposed fast VM storage migration using data deduplication. As mentioned above, our idea would be effective if VMs are transferred between specific physical hosts.

In this section, we discuss the kinds of scenarios in which VMs can be transferred between specific physical hosts, and the kinds of situations for which our deduplication mechanism would be suitable.

In today's cloud computing environments, we believe that the transfer of VMs between specific physical hosts is very likely. Two scenarios are as follows:

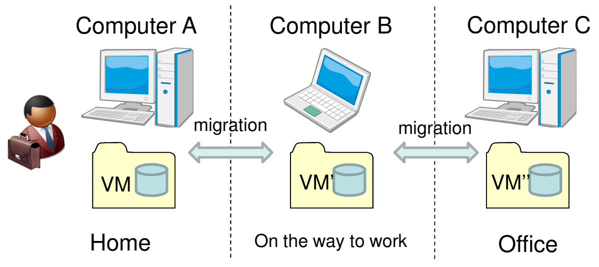

Figure 1. Live-migration for personal use.

A. VM storage migration for personal computer environment

Recently, the use of computing systems that provide VMs instead of physical computers has increased. In this case, users cannot use physical computers directly, but can use virtual machines that are isolated from each other. We refer to such a system as a *personal virtual machine system*. Personal VM systems are more convenient than traditional physical computers and have the following benefits: First, we can easily store entire VMs on portable storage devices such as USB memory, and it is therefore portable. Personal computer environments can therefore be used at any location. Secondly, Personal VM systems can provide highly secure computing, because backups of personal computing environments can easily be made as virtual machine images. If a VM is contaminated by a malicious program, the computing environment can be quickly recovered using a backup VM image. As previously mentioned, personal VM systems have many benefits. Moka Five [17] has developed a personal VM system which is widely used, and it is believed that personal VM system usage will rapidly spread.

We believe that by introducing storage migration technologies to personal VM systems, we can realize more convenient computing environments. Figure 1 shows an image of VM storage migration in a personal VM system. This image was inspired by Shivani Sud et al. [2], whose work we summarized as follows:

> Jane uses her home computer to check her email and reviews a presentation she needs to deliver later that morning. As the day progresses, she seamlessly migrates her work environment from her home PC to her mobile device before leaving home. While traveling she continues reviewing the presentation, adding notes as she rides the subway to work. Soon it is time for her to dial into a teleconference. On reaching her desk, her work environment seamlessly migrates from her mobile device to the office PC, and she can now use the office PC to continue reviewing the presentation, while she continues her teleconference from her mobile device.

However, as mentioned above, VM disk images are large (typically 1-30GB in size). Consequently, VM storage migration is time-consuming. For example, a 20GB disk image, requires about 10 minutes in a gigabit LAN environment. Traditional VM live-migration including storage migration focuses mainly on reducing the down-time in extreme cases. In fact, pre-copy and post-copy live-migration algorithms are designed in an effort to reduce the down time. However, in the assumed personal VM, it is important to minimize the entire migration time as opposed to the down-time. This is because for personal VM live-migration, the most important thing is that when a user wants to migrate a VM to another device, the VM moves immediately to the another device without subsequent network communication. Our fast storage migration using data deduplication can reduce the entire transfer time, and is therefore considered to be effective.

B. Follow the "moon" data center access model

To reduce the electricity bill and cooling cost in data centers, some companies have proposed a strategy to deploy server computers to regions in the world which have nightfall. Using this approach, it is possible to maximize the use of inexpensive off-peak electricity and lower temperatures. By taking this approach, VMs are frequently deployed in the data centers which are located in nighttime regions. In fact, a VM may be migrated to a host to which it has previously been deployed. By introducing our fast storage migration, we can reduce transfer time. Additionally, in today 's multi-tenant cloud computing environments, many customers' VMs are consolidated into a single data centers. If many VMs concurrently migrate to a night-time data center with their large disk images, the bandwidth of the data center may become saturated. However, our fast storage migration approach can reduce the volume of transferred data for VM migrations.

As previously mentioned, in today's cloud computing environments, VM would alternate between particular physical hosts. Thus, our fast storage migration using data deduplication is an effective way of achieving this goal.

IV. System Overview

We implement a simple prototype for VM storage migration mechanisms using data deduplication. More precisely, we implement DBT (Dirty Block Tracking) on a hypervisor. DBT is a tracking module for the writing of a guest OS on a VM. In order to achieve fast storage migration, DBT works as follows : First, a VM disk image is divided into fixed chunks using DBT. Next, DBT tracks all written disk page blocks. DBT leverages a bitmap to manage the disk pages that have been updated by the guest OS. Then, when VM live-migration from a host A to a host B occurs, the hypervisor executes normal slow VM-live migration if there are no available reusable disk page blocks in the destination

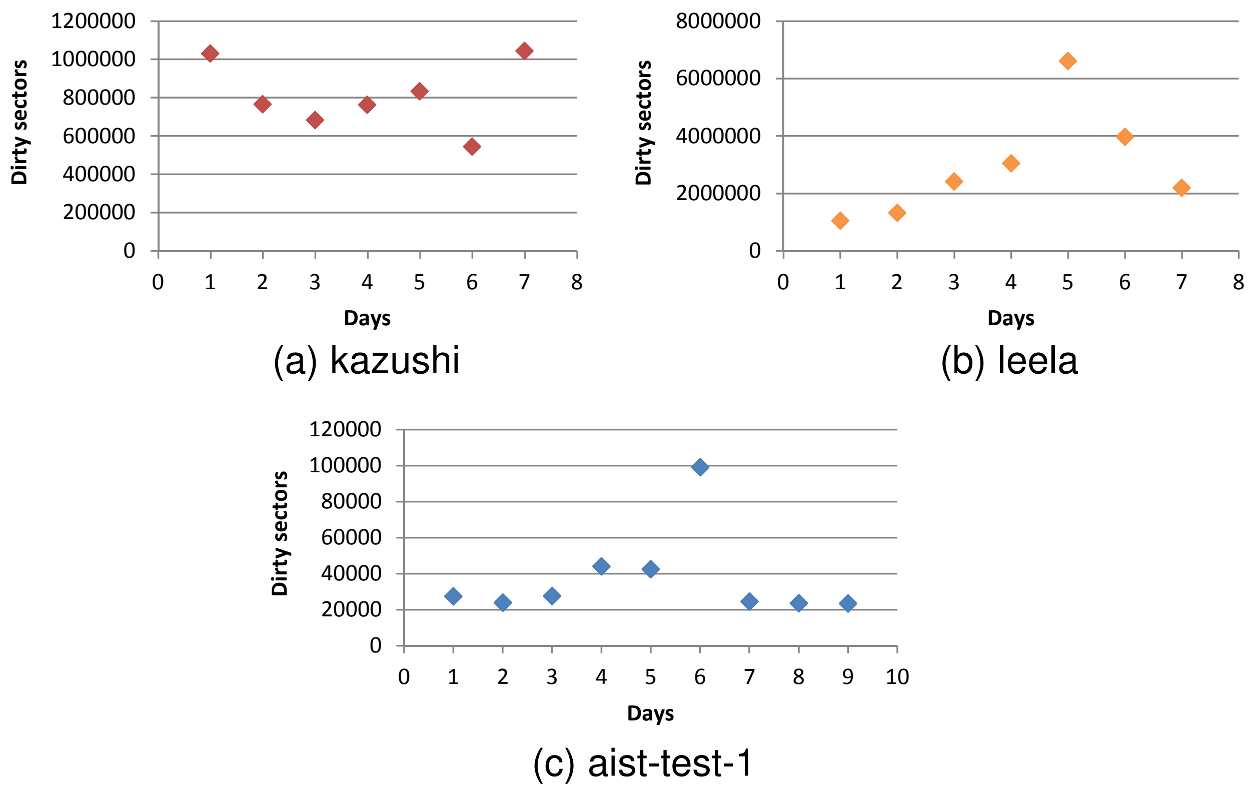

Figure 2. Three machines and three different workloads.

host B. Entire VM disk images are translated. On the other hand, if there are reusable disk pages in the destination host B, only dirty pages which have been updated on the source host A are transferred. With this mechanism, we can achieve our fast VM storage migration using data deduplication.

V. Pre Examination

To examine the efficiency of deduplication transfer for storage migration, we investigate for several days the number of disk pages that are updated in the average daily operation of a computing environment. We examined three computers which have different workloads. Three computer setups are shown in Table I. With the exception of aist-test-1, all of the machines are physical machines. Kazushi consisted of a hard disk drive that is 300GB in size, and which executes Ubuntu 11.04 (32bit) with the ext4 file system format. Leela consisted of a solid state drive that was 160GB in size, and which executes Windows 7 (32bit) with the NTFS format. Aist-test-1 consisted of a hard disk drive that was 60GB in size, and which executes CentOS 6.2 (64 bit) with the ext4 file system format.

Aist-test-1 is a server which works as a web-based groupware. Kazushi is a laptop computer which executed web-browsing operations including the playback of YouTube videos and text editing. Leela is also a laptop computer which executed only web-browsing operations, including the playback of YouTube videos.

Table I. Pre examination environments

Name	Type	Size	OS	FS
(a) kazushi	HDD	300GB	Ubuntu 11.04 (32bit)	ext4
(b) leela	SSD	160GB	Windows 7 (32bit)	NTFS
(c) aist-test-1	HDD	60GB	CentOS 6.2 (64bit)	ext4

Our results are shown in Figure 2. For all of the test machines, we find that changes were made to only a few disk pages in the entire physical disks. First, in the case of kazushi, a maximum of only 0.49 GB disk space was updated, out of 300 GB. This accounts for only 0.16% of the 300 GB disk. A minimum of, only 0.25 GB of the 300 GB disk space was updated. This accounts for only 0.08% of the disk. Secondly, in the case of leela, a minimum of only 0.49 GB of the 160 GB disk space was updated, accounting for only 0.33% of the disk, while a maximum of only 3.14 GB of the 160 GB disk space was updated, accounting for only 2.11% of the disk. Thirdly, for aist-test-1, a minimum of only 0.01 GB of the 60 GB disk space was updated, accounting for only 0.07% of the disk, while a maximum of 0.04 GB of the 60 GB disk space was updated, accounting for only 0.31% of the virtual disk.

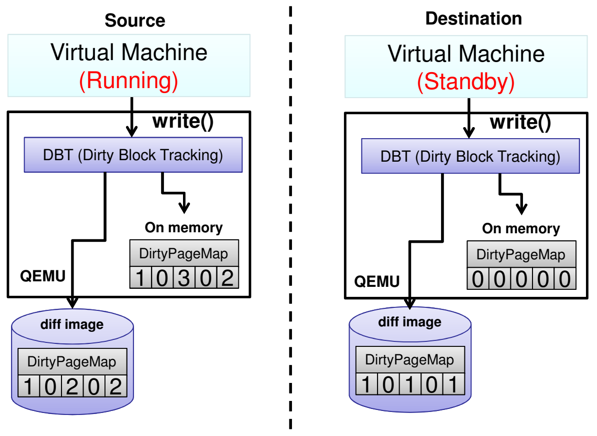

Figure 3. Our prototype system overview.

This examination shows that our deduplication transfer mechanism for virtual machine storage migration is efficient.

VI. Design

In the previous section, we showed the deduplication system for storage migration is efficient. Thus, we designed a system for the deduplication of storage migration. An overview of our system is shown in Figure 3.

1) Dirty Block Tracking: DBT is a mechanism that traces entire disk pages written by a guest OS, and records the tracking result into the dirty map structure. DBT within a hypervisor hooks the writing by a guest OS. DBT simply divides entire disk images at block boundaries, and allocates 8 bits of space for each block. This 8 bit space is updated by DBT with a generation number, which is an identifier for VMs. When a VM moves to another host by storage migration, this value is increased. Consequently, we can identify the disk pages that should be transferred when storage migration takes place.

2) Diff Image Structure: Diff image is a new VM image format that supports deduplication for VM storage migration by managing the structured dirty map with DBT. The diff image structure is shown in Figure 4. The diff image consisted of G, which is the generation number, S, which is the seed number, and the freeze flag, which indicates whether or not the VM image is able to boot, and the dirty page map, which indicates which blocks have been updated by the guest OS. The generation number is increased when the VM migrates to another host, and this number is initially one. The seed number is a unique number which is allocated when the disk image is created.

3) Migration Algorithm: The migration procedures are as follows:

(a) When a diff disk image is created, the diff structure is initialized. The seed number is a unique number, the generation number is 1, the freeze flag is 0, and the dirty page map is all zeros.

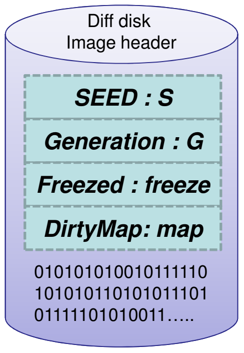

Figure 4. The structure of diff disk image.

(b) A guest OS writes to the disk. Using DBT, all writings by the guest are tracked and recorded. The blocks updated by the guest are tracked and recorded. The recording is conducted by writing the generation number into the dirty page map.

(c) The guest OS migrates to a host A. This is a slow storage migration process because it is an initial transfer. The generation number G is incremented when this migration is completed.

(d) A host B, which is the destination, acquires ownership of the guest OS. Thus, the VM image in the host A is frozen, and it is temporarily not bootable.

(e) The guest OS writes to the disk in the host B. DBT records the disk pages that are updated in the generation number G.

(f) The guest OS migrates the host B to host A. Now, fast deduplication storage migration is possible. DBT compares the generation number in the host A with generation numbers in the dirty map in the host B. If a generation number in the dirty map in host B is greater than the generation number G in host A, the disk block which corresponds to the generation number in the dirty map is transferred.

A. Discussion

We also considered another approach to achieve data deduplication for storage migration. In our prototype, we proposed a simple method to track disk block writing with dirty maps using DBT. On the other hand, other methods using fingerprints such as SHA-1 [9] and Rabin-fingerprint [18] are available. In fact, as in rsync [19], we can use the *Compare-by-hash* [20], [21] method to achieve data deduplication.

According to Jacob et al. [22], hand-optimized SHA-1 implementation, running on a single Intel Core-2 CPU core is able to hash more than 270 MB of data per second, which is almost enough to saturate the full write bandwidth of three SATA disk drives. Thus, although DBT calculates SHA-1 hash when disks are written to by a guest OS, the guest OS does not incur loss of speed.

However, our goal in this paper is to show that our deduplication VM storage migration method is practical. Thus, in this paper, we adopt a more simple method which uses the dirty page map with generation numbers. As described later, we believe that the use of SHA-1 hash has some side effects. We now plan to develop a deduplication VM transfer mechanism using SHA-1

VII. Implementation

We implement the previously mentioned DBT and the diff format to Linux/KVM (QEMU) [23], [24] We divide entire VM image files into chunks. DBT constructs a dirty page map using an 8-bit space for each of the chunks. Currently, for the bitmap, one chunk is 2,048 sectors, where one sector is 512 bytes. This is a constant value for QEMU. A bitmap that is in 4 Kbytes is generated for 4 GB. VM disk images. Although a VM image that is 20 GB is generated, a bitmap that is about 20 KB is generated. Thus, the bitmaps do not place additional stress on the physical hard disk drive. This bitmap is deployed in memory when the guest OS executes on the VM. When the guest OS exits, the bitmap is updated on the diff image.

Additionally, we implement two APIs to communicate the dirty block information between the disk driver layer and the live migration mechanism layer in QEMU, and to increment the generation number when storage migration is completed. QEMU implementation is a structured design, For example, vmdk, qcow, and qcow2, which are formats of the QEMU's disk images, are updated as device drivers in QEMU. In order to develop new QEMU disk formats, developers implement only the specific handlers in QEMU. Developers, who are desirous of adding new QEMU disk image formats, can implement a new QEMU format by implementing only the specific callback handlers. BlockDriver structure in block_int.h source header file of QEMU defines the callback handlers to implement QEMU disk image in QEMU. We add two callback handlers to QEMU because there are no APIs to increment the generation number and, to communicate the dirty map between the disk driver layer and live-migration implementation layer.

VIII. Evaluation

We evaluate and analyze the impact of our deduplication migration mechanism. First, we examine whether or not writing to the disk has slowed. Secondly, we conduct measurements for the speed and efficiency of the data transfer using several machines which have different workloads.

A. Evaluation for the DBT cost

In this subsection, we show that DBT does not incur a loss of speed by tracking the disk writing operations. Our setup consisted of a ThinkPad X220 laptop computer that was booted up with an Intel Core i5-2430M @ 2.40GHz, and which has 4 GB of memory. This machine is attached to a Seagate 7200rpm HDD with a cache memory of 16 MB.

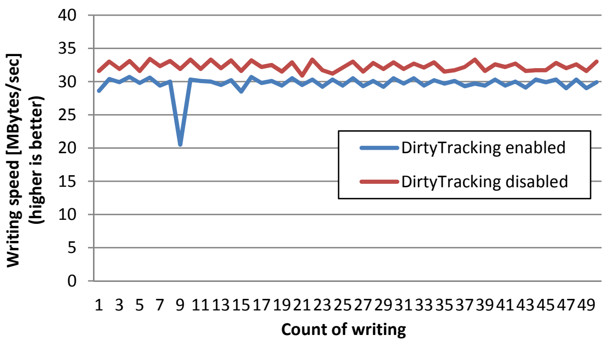

Figure 5. The result of writing benchmark.

To realize the impact of DBT on the performance, we compare two benchmark results: First, the write speed of the diff format with DBT. Secondly, the write speed for the raw-format, which is the primitive VM format for QEMU. We measure only the write cost and not the read cost because DBT only works with guest disk writing. We use UNIX dd commands to measure the disk write cost. Using dd, we write 1 MB blocks 100 and 50 times.

The evaluation result is shown in Figure 5. The x-axis indicates the write speed in Mbytes/sec, and The y-axis indicates the writing count. We find that the hypervisor without DBT achieved 32.304 Mbytes/sec, and the hypervisor with DBT achieved 29.656 Mbytes/sec. Because DBT leads to a decrease in the write speed of only 8%, it is not thought to be significant.

B. Migration speed with different workloads

In this subsection, using a series of benchmarks, we show the speed and the efficiency of data transfer for migration.

We assume practical workloads as follows: First, a user downloads an MS Office power point file, views, edits, and saves the file. Secondly, the user downloads from a Japanese literary website a literary creation "Kokoro" by Souseki Natume, who is a famous Japanese scholar in the field of Literature. The user then views, edits, and saves the information. Next, the user views the video on YouTube with a Firefox web-browser. Fourthly, the user downloads a 3 MB PDF file and views it. We run these practical workloads on both Windows 7 (32bit) and Ubuntu 11.04 (32bit) operating systems. After each user performs these actions for five minutes on both virtual machines, we conduct our deduplication procedure for both VM storage migration and normal VM storage migration. Finally, we compare the times taken for our deduplication storage migration and normal VM storage migration.

The source host consists of a ThinkPad X60 laptop computer booted with an Intel Core Duo CPU T2400 @ 1.83 G Hz with 2 GB of memory. The destination host consists

Table II. Storage migration measurement result on Ubuntu 11.04 desktop (32bit)

	No deduplication	PDF	Presentation	YouTube	Kokoro
Whole Migration Time (sec)	919.358	29.139	30.141	28.720	25.900
Transferred size (MBytes)	—	101	104	111	90

Table III. Storage migration measurement result on Windows 7 Professional (32bit)

	No deduplication	PDF	Presentation	YouTube	Kokoro
Whole Migration Time (sec)	991.044	89.028	68.892	78.450	86.703
Transferred size (MBytes)	—	933	613	927	907

of a DELL LATITUDE D630 laptop computer booted with an Intel Core 2 Duo T7300 @ 2.0 GHz with 2 GB of memory. Both computers are connected in a 1 Gbps LAN environment.

The result for Ubuntu 11.04 (32 bit version) is shown in Table II. The longest migration time was 30.141 seconds for the presentation benchmark. On the other hand, the best migration time was 25.900 seconds for the Kokoro benchmark. We found that our approach was able to reduce the migration time. All of the benchmark results lasted about 10 minutes. However, they lasted only about 10 seconds after introducing our approach. We also found that using our method, the volume of transferred data had been reduced from 20 GB to several hundreds of megabytes.

Next, the result for Windows 7 Professional (32 bit version) is shown in Table III. When compared with the result for Ubuntu, in the case of Windows, the number of dirty disk blocks was larger. and the volume of data that was transferred was also larger. The longest migration time was 89.028 seconds for the PDF benchmark. On the other hand, the best migration time was 68.892 seconds for the presentation benchmark. Windows was shown to generate a greater number of dirty disk blocks than Ubuntu. Additionally, we found that the Presentation benchmark consumed the most migration time in the case of Ubuntu while the PDF benchmark consumed the most migration time in the case of Windows. The best time in the case of Ubuntu was 28.720 seconds for the YouTube benchmarks, while for Windows, the best time was 68.892 seconds for the presentation benchmark. All of the benchmarks results in Windows achieved a migration time of about 1 minute.

It was found that the introduction of the deduplication for storage migration led to greater efficiency. For all of the benchmarks, we were able to achieve faster storage migration than traditional storage migration techniques.

IX. Future Work

As mentioned above, we can use fingerprint algorithms such as SHA-1 and Rabin fingerprint to deduplicate VM disk pages. In fact, we divide the VM image files into chunks, and calculate fingerprints for all chunks. VM images are transferred, we can compare fingerprints in the source VM image with those in the destination VM image to exploit deduplicated VM disk blocks. This approach also provides deduplication on local VM disk storage to reduce the volume of local storage data. Although this approach somewhat complicated, it is better than our bitmap approach. Therefore, we will implement the deduplication system using a fingerprint algorithm.

X. Conclusion

In this paper, we proposed a fast VM storage migration technique using data deduplication. In the pre-examination results, we show that data deduplication for fast VM is an effective approach because only a few disk blocks are usually updated in daily computing operations. Thus, we implement a prototype that realizes fast VM storage migration using data deduplication. For all of the benchmarks, we achieve storage migration that is faster than traditional storage migration.

References

[1] S. Shepler, B. Callaghan, D. Robinson, R. Thurlow, C. Beame, M. Eisler, and D. Noveck, "Network file system (nfs) version 4 protocol," 2003.

[2] S. Sud, R. Want, T. Pering, K. Lyons, B. Rosario, and M. X. Gong, "Dynamic migration of computation through virtualization of the mobile platform," in *MobiCASE*, 2009, pp. 59–71.

[3] K. Takahashi and K. Sasada, "A fast vm transport mechanism that consider generations with disk dirty page tracking," in *53th Programming Symposium*. IPSJ, January 2011, pp. 37–45.

[4] Y. Luo, B. Zhang, X. Wang, Z. Wang, Y. Sun, and H. Chen, "Live and incremental whole-system migration of virtual machines using block-bitmap." in *CLUSTER'08*, 2008, pp. 99–106.

[5] R. Bradford, E. Kotsovinos, A. Feldmann, and H. Schiöberg, "Live wide-area migration of virtual machines including local persistent state," in *Proceedings of the 3rd international conference on Virtual execution environments*, ser. VEE '07. New York, NY, USA: ACM, 2007, pp. 169–179. [Online]. Available: http://doi.acm.org/10.1145/1254810.1254834

[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," *SIGOPS Oper. Syst. Rev.*, vol. 37, pp. 164–177, Oct. 2003. [Online]. Available: http://doi.acm.org/10.1145/1165389.945462

[7] T. Hirofuchi, H. Ogawa, H. Nakada, S. Itoh, and S. Sekiguchi, "A live storage migration mechanism over wan for relocatable virtual machine services on clouds," *Cluster Computing and the Grid, IEEE International Symposium on*, vol. 0, pp. 460–465, 2009.

[8] S. Al-Kiswany, D. Subhraveti, P. Sarkar, and M. Ripeanu, "Vmflock: virtual machine co-migration for the cloud," in *Proceedings of the 20th international symposium on High performance distributed computing*, ser. HPDC '11. New York, NY, USA: ACM, 2011, pp. 159–170. [Online]. Available: http://doi.acm.org/10.1145/1996130.1996153

[9] National Institute of Standards and Technology, *FIPS PUB 180-1: Secure Hash Standard*, Apr. 1995, supersedes FIPS PUB 180 1993 May 11. [Online]. Available: http://www.itl.nist.gov/fipspubs/fip180-1.htm

[10] C. P. Sapuntzakis, R. Chandra, B. Pfaff, J. Chow, M. S. Lam, and M. Rosenblum, "Optimizing the migration of virtual computers," *SIGOPS Oper. Syst. Rev.*, vol. 36, pp. 377–390, December 2002. [Online]. Available: http://doi.acm.org/10.1145/844128.844163

[11] M. Kozuch, M. Satyanarayanan, T. Bressoud, and Y. Ke, "Efficient state transfer for internet suspend/resume," *Intellectual Property*, no. May, 2002.

[12] N. Tolia, N. Tolia, T. Bressoud, T. Bressoud, M. Kozuch, M. Kozuch, and M. Satyanarayanan, "Using content addressing to transfer virtual machine state," Tech. Rep., 2002.

[13] M. Satyanarayanan, J. J. Kistler, P. Kumar, M. E. Okasaki, E. H. Siegel, David, and C. Steere, "Coda: A highly available file system for a distributed workstation environment," *IEEE Transactions on Computers*, vol. 39, pp. 447–459, 1990.

[14] VMware, Inc., "VMware Storage VMotion: Non-disruptive, live migration of virtual machine storage," http://www.vmware.com/products/storage-vmotion/

[15] A. Mashtizadeh, E. Celebi, T. Garfinkel, and M. Cai, "The design and evolution of live storage migration in vmware esx," in *Proceedings of the 2011 USENIX conference on USENIX annual technical conference*, ser. USENIXATC'11. Berkeley, CA, USA: USENIX Association, 2011, pp. 14–14. [Online]. Available: http://dl.acm.org/citation.cfm?id=2002181.2002195

[16] P. Riteau, C. Morin, and T. Priol, "Shrinker: Efficient Wide-Area Live Virtual Machine Migration using Distributed Content-Based Addressing," INRIA, Research Report RR-7198, Feb. 2010. [Online]. Available: http://hal.inria.fr/inria-00454727/en/

[17] MokaFive, "MokaFive Player," http://www.moka5.com/

[18] M. O. Rabin, "Fingerprinting by random polynomials." TR-CSE-03-01, Center for Research in Computing Technology, Harvard University, Tech. Rep., 1981.

[19] A. Tridgell and P. Mackerras, "The rsync algorithm," Australian National University, Department of Computer Science, Technical Report TR-CS-96-05, Jun. 1996, http://rsync.samba.org.

[20] J. Black, "Compare-by-hash: a reasoned analysis," *Proc 2006 USENIX Annual Technical Conference*, pp. 85–90, 2006. [Online]. Available: http://www.usenix.org/event/usenix06/tech/full_papers/black/black.pdf

[21] V. Henson and R. Henderson, "Guidelines for using compare-by-hash," 2005.

[22] J. G. Hansen and E. Jul, "Lithium: virtual machine storage for the cloud," in *Proceedings of the 1st ACM symposium on Cloud computing*, ser. SoCC '10. New York, NY, USA: ACM, 2010, pp. 15–26. [Online]. Available: http://doi.acm.org/10.1145/1807128.1807134

[23] F. Bellard, "Qemu, a fast and portable dynamic translator," in *ATEC'05: Proceedings of the USENIX Annual Technical Conference 2005 on USENIX Annual Technical Conference*. Berkeley, CA, USA: USENIX Association, 2005, p. 41. [Online]. Available: http://portal.acm.org/citation.cfm?id=1247401

[24] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori, "KVM: the Linux Virtual Machine Monitor," in *Proceedings of the Linux Symposium*, 2007, pp. 225–230.