PAPER Special Section on Parallel and Distributed Computing and Networking

A WAN-Optimized Live Storage Migration Mechanism toward Virtual Machine Evacuation upon Severe Disasters

Takahiro HIROFUCHI[†a)], **Mauricio TSUGAWA**[††], **Hidemoto NAKADA**[†], ***Nonmembers***,
Tomohiro KUDOH[†], ***Member***, ***and*** **Satoshi ITOH**[†], ***Nonmember***

SUMMARY Wide-area VM migration is a technology with potential to aid IT services recovery since it can be used to evacuate virtualized servers to safe locations upon a critical disaster. However, the amount of data involved in a wide-area VM migration is substantially larger compared to VM migrations within LAN due to the need to transfer virtualized storage in addition to memory and CPU states. This increase of data makes it challenging to relocate VMs under a limited time window with electrical power. In this paper, we propose a mechanism to improve live storage migration across WAN. The key idea is to reduce the amount of data to be transferred by proactively caching virtual disk blocks to a backup site during regular VM operation. As a result of pre-cached disk blocks, the proposed mechanism can dramatically reduce the amount of data and consequently the time required to live migrate the entire VM state. The mechanism was evaluated using a prototype implementation under different workloads and network conditions, and we confirmed that it dramatically reduces the time to complete a VM live migration. By using the proposed mechanism, it is possible to relocate a VM from Japan to the United States in just under 40 seconds. This relocation would otherwise take over 1500 seconds, demonstrating that the proposed mechanism was able to reduce the migration time by 97.5%.

key words: *disaster recovery, live migration, virtual machine, virtual machine monitor*

1. Introduction

IT services are indispensable for our daily life and commercial activity, as such, there is a need for robust and dependable IT Infrastructure that can remain operational even upon severe disasters. On March 11th 2011, the Great East Japan Earthquake hit Japan as the most powerful known earthquake to have ever hit the country. In the east area of Japan, electricity power grid systems were partially damaged. IT services were disrupted due to power blackout, harming business continuity as well as preventing rapid response against the disaster.

In our former study [1], we investigated damages of datacenters operated by universities and research institutions located in the disaster-struck area. Contrary to our expectation, physical damages to server computers and network equipments were minimal, thanks to quake-resistant building constructions and correct installations and operation of hardware components. Uninterruptible power supplies (UPSs) and power generators kept servers and network devices operational for tens of minutes. Several servers were able to maintain Internet connectivity despite the severity of the disaster. In the academic backbone network of Japan, although some optical fiber cables, close to the eastern coast, suffered damages, the routing paths of network traffic were automatically switched to other undamaged routes.

We believe this finding opens alternative opportunities to the design of IT systems disaster recovery (DR). If IT systems are operational during a *time window* of tens of minutes after a disaster, we can possibly *evacuate* services to safe locations. This contrasts with traditional approaches of DR that proactively backs-up data assuming that services are shutdown upon a disaster. In our ongoing project, we consider live migration of virtual machines (VMs) as an enabling technology to evacuate virtualized IT systems to safe locations. Live migration is implemented in the virtual machine layer and is not an application-specific mechanism, transparently supporting the evacuation of unmodified applications. The evacuation of an entire IT system requires the transfer of all VM states including CPU, memory, and storage such that no dependency with the disaster site is left after migration. Classic live migration mechanisms [2]–[4] focuses on CPU and memory state transfer, assuming shared storage among physical servers. Although several studies to include storage migration exist, they require long periods of time to relocate virtual disks. In the context of DR, there is a need for an improved storage migration mechanism that can complete the evacuation of VMs to a remote site within a limited time window.

In this paper, we propose a WAN-optimized live storage migration mechanism that can relocate the whole VM state to a remote site in a short period of time (e.g., tens of seconds). The proposed mechanism proactively transfers virtual disk state to a backup site while in normal operating conditions. When a disaster strikes, our system performs live storage migration of not-yet-cached blocks concurrently with the memory and system states relocation. By transferring the data in advance, the proposed mechanism can dramatically reduce the amount of data transferred and consequently the time required to complete the live migration of the whole VM state. The live storage migration is implemented by using a postcopy migration technique [5]. The amount of data transferred for storage migration is de-

Manuscript received December 25, 2012.
Manuscript revised May 18, 2013.

[†]The authors are with Information Technology Research Institute, National Institute of Science and Technology, Tsukuba-shi, 305–8568 Japan.

[††]The author is with Advanced Computing and Information Systems Laboratory, University of Florida, Gainesville, FL32611, United States of America.

a) E-mail: t.hirofuchi@aist.go.jp

DOI: 10.1587/transinf.E96.D.2663

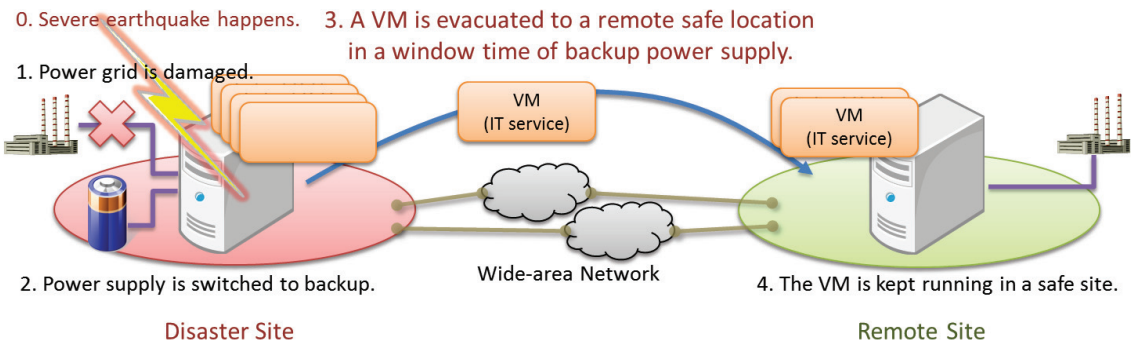

Fig. 1 VM evacuation upon disasters.

terministic and does not change depending on the activity of the VM. Using the proposed mechanism, it is possible to complete a storage migration in a short period of time, even for VMs that intensely update disk blocks.

Section 2 motivates this paper in the context of our IT services recovery project. Section 3 explains the design and implementation of the proposed mechanism. Section 4 evaluates a prototype implementation and presents experimental results. Section 5 discusses related work and Sect. 6 concludes this paper.

2. VM Evacuation Upon Disasters

Our project studies disaster-resilient IT infrastructure, which can tolerate severe earthquakes such as the Great East Japan Earthquake. It is known that severe earthquakes damage power plants and power lines, causing power blackout in datacenters that continues for days to months. However, just after an earthquake, backup power systems can supply electricity to server computers for tens of minutes until running out of battery or fuel. As illustrated in Fig. 1, our system evacuates VMs to a safe site during this time window. The assumption considering the experience of 2011 Earthquake is that thanks to quake-resistant technologies, server computers and network switches can be kept operational. The major concern for IT infrastructure is the lack of electricity.

When a severe disaster strikes, our system starts VM evacuation to a remote site in a safe location. An emergency earthquake alert broadcasted by a governmental agency, or power blackout will be possible triggers to start the evacuation. The earthquake alert system has been successfully used in other real-world systems. For example, when a severe earthquake is detected, train control systems immediately broadcast emergency brake commands to all trains in operation. The speeds of trains are reduced to a safe level before seismic waves reach the trains. We are confident that integrating IT systems with the alert system could make them robust against earthquakes.

In our prior work [6], we have developed a transparent tunneling mechanism exploiting the Mobile IPv6 protocol [7], which allows VMs to preserve network reachability by keeping the same IP address even after migrating to different locations. We have also studied a feedback-based control mechanism to optimize parallel live migration sessions [8]. It enables an evacuation system to migrate a cluster of VMs efficiently to remote sites.

In this paper, we address the performance problem of storage migration mechanisms that require an extraordinarily long time to complete. Storage migration entails the transfer of all virtual disk blocks, with the sizes on the order of Gbytes, and it is challenging to transfer such amount of data within a limited time window.

3. Proposed Mechanism

Our storage migration mechanism is called `xNBD+`. As illustrated in Fig. 2, the proposed mechanism is composed of two daemon programs, both of which run on storage servers at the source and destination sites. The NBD I/O daemon handles I/O requests from a VM, and performs read/write operations on a disk image file. The NBD sync daemon periodically synchronizes the image file disk between two sites.

The proposed mechanism is mostly implemented in the sync daemon, while a VM only interacts with the I/O daemon and sees a storage server of the Network Block Device (NBD) protocol [9] (i.e., a block-level I/O protocol over IP networks). The NBD I/O daemon provides a virtual disk to a VM, which is running on Qemu/KVM [10] or Xen [11]. Qemu/KVM supports the NBD protocol without any additional components, and Xen also supports the protocol through the block device driver of the NBD protocol on a host operating system.

During normal operation, the NBD sync daemon at the source site periodically sends written data to a destination server in a backup site, synchronizing the data of a virtual disk with cache data on the destination. This synchronization is performed only using the excess bandwidth between the two sites, and completely independent of the disk I/O operations performed by VMs. Thus, the disk I/O performance of VMs is not adversely affected even if WAN conditions change. In disaster operation, the proposed mechanism performs a live storage migration, which does not need to transfer all the disk blocks but only the blocks that are not cached. Because pre-caching is performed on a best-effort basis using the available excess bandwidth, the number of

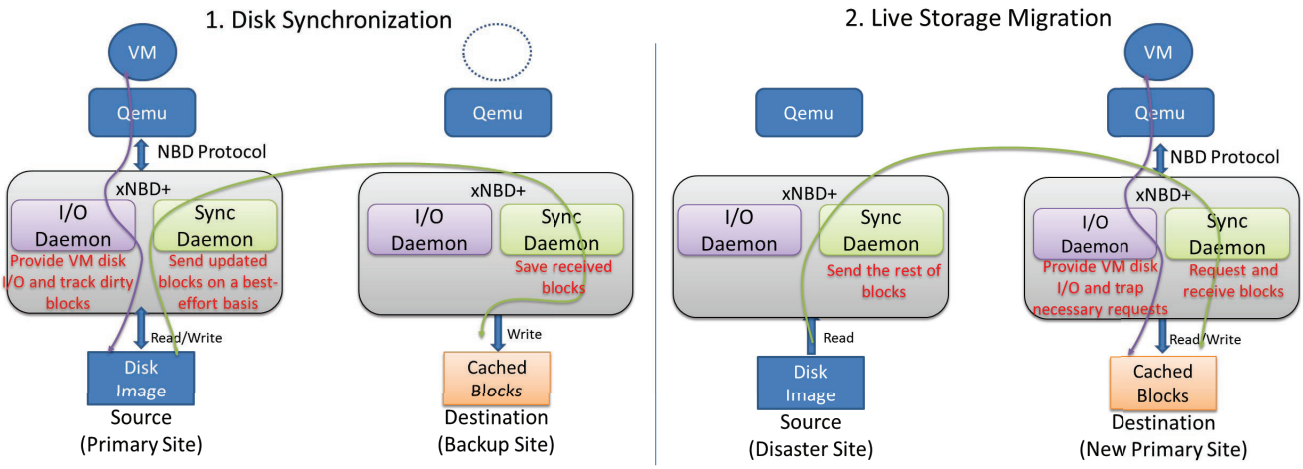

Fig. 2 Overview of our WAN-optimized storage migration mechanism.

successfully cached blocks depends on the VM disk I/O activity and network conditions. However, as shown in later sections, time to migrate an entire VM state can be dramatically reduced in many situations.

The proposed mechanism builds on a postcopy migration technique that we previously developed [5] and works as follows: first, a virtual machine monitor (VMM) performs a regular VM migration, which does not include the relocation of storage. Then, the destination storage server starts processing disk I/O requests from the migrated VM. If a read request to a not-yet-cached disk block is issued, the destination storage server reads the corresponding block from the source server, and returns the data of the block to the VM as well as writes the data to the local storage at the destination. The other types of requests (i.e., read operations to already cached blocks, and write operations) are handled only at the destination, since it is not necessary to read corresponding blocks from the source. In parallel with this on-demand block transfer, the destination server reads the rest of the blocks in the background. Finally, all the blocks are transferred to the destination.

In contrast, precopy storage migration used in other studies works as follows: after a migration is invoked, a VMM copies all memory pages and disk blocks to the destination. Since the VM is still running at the source, memory pages and disk blocks are updated at the source server while they are copied to the destination. These updated memory pages and disk blocks are iteratively copied to the destination, until the data size of remaining pages and blocks is sufficiently small for an acceptable downtime. Then, the VMM stops the VM, and copies the rest of VM states (e.g., remaining memory pages, disk blocks, VCPU registers, and device status) to the destination. Finally, the VMM resumes the VM at the destination. As discussed in Sect. 4.2, the iterative copy phase of the precopy migration is not suitable for WAN environments. In the worst case, the size of remaining VM states never reaches a threshold value, which means a migration does not finish.

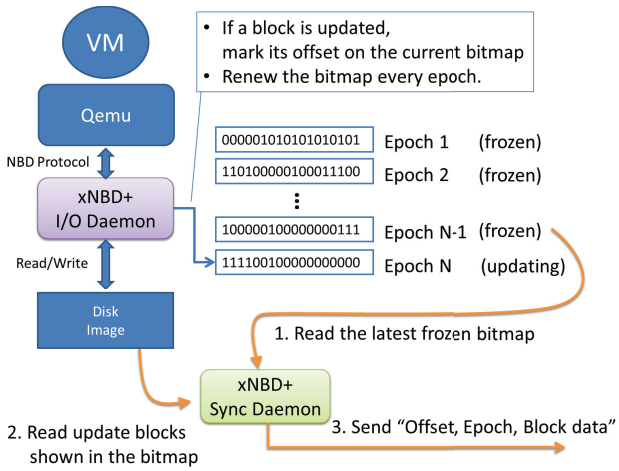

Fig. 3 The sender side mechanism of the disk synchronization in normal operation.

3.1 Disk Synchronization During Normal Operation

The disk synchronization mechanism has been designed to meet the following requirements: (1) the synchronization mechanism must be robust against unstable WAN environments - this means that the system must prevent data inconsistencies in the backup site caused by potential intermittent network failures; (2) the synchronization activities must not adversely affect VM performance - i.e., the synchronization mechanism should not influence the operations performed by the VM. Traditional DR systems over a WAN need to sacrifice system performance for strict data synchronization. Our project, assuming the existence of a time window upon disasters, aims to explore an alternative approach that can preserve system performance during normal operation.

Details of the designed synchronization mechanism are illustrated in Fig. 3 and Fig. 4. The synchronization mechanism periodically searches for updated blocks at the primary storage server and sends them to the backup storage server

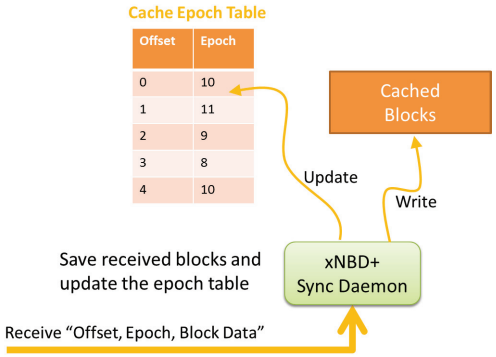

Fig. 4 The receiver side mechanism of the disk synchronization in normal operation.

on a best-effort basis. Virtual disk synchronization is carried out concurrently with handling I/O requests for the VM, using the available bandwidth of a WAN. All synchronization messages include an epoch number, which indicates the freshness of synchronization data. Epoch number is also used to detect potential data loss in the backup site due to network connection failures or synchronization bandwidth shortages. When the connection is recovered, pending synchronization data is transmitted. In addition, our proposed mechanism does not have any exclusive lock between disk I/O handling and synchronizing actions. The performance of disk I/O is completely independent of fluctuating WAN performance even when network becomes unreachable.

On a source storage server, the I/O daemon is responsible for maintaining the epoch counter, which is incremented periodically. A bitmap keeps track of updated blocks in each epoch. When the epoch counter is incremented, the I/O daemon freezes the current bitmap, and creates the next bitmap. The sync daemon detects the creation of a new bitmap, scans the bitmap of the previous epoch[†], finds updated block offsets, and sends synchronization data by reading the offsets of the disk image file. The data of an updated block is encapsulated into a synchronization data frame, where the offset of the block and the epoch counter of the bitmap are added to the data.

On a destination storage server, the sync daemon is responsible for receiving synchronization data frames, and storing synchronization data in a local storage. It also updates an offset-epoch-pair table (Cache Epoch Table, CET) based on received frames. CET records the freshness of each block (represented by an offset number) in the cached image file. For example, if the CET entry for the block of offset 4 indicates epoch 10, the cached data corresponding to the block was transferred reading the bitmap of the epoch 10. Because the sync daemon scans only frozen bitmaps, this happened in the epoch 11 or later. It should be noted that the data of the cached block is not a snapshot at a given point of time. Since there is no exclusive access control between the I/O daemon and the sync daemon, the sync daemon may potentially send data of a block that is being overwritten by the I/O daemon. Data inconsistencies are resolved in the final phase of the storage migration.

[†]Skip the bitmap of the current epoch, which is being updated by the I/O daemon.

3.2 Live Disk Migration in Disaster Operation

Upon a disaster, our system starts the process of relocating the whole VM state to a safe location. The memory state of the VM is migrated by a mechanism implemented by the VMM, while the disk state of the VM is migrated by our mechanism proposed in this paper. For memory migration, we can use a normal precopy-based migration mechanism that is available in most VMMs([2]–[4]), or the precopy/postcopy hybrid migration mechanism that we have developed for the latest Qemu/KVM [12].

If normal precopy-based memory migration is used, the entire migration process works as follows. First, a memory migration mechanism transfers memory pages to the destination for a while. After the rest of memory pages become sufficiently small, the memory migration mechanism temporarily stops the VM and copies the memory and system states to the destination. In the background, the storage migration mechanism performs the following steps.

0. The source sync daemon stops the synchronization operation and prepares for the migration of not-yet-cached blocks.
1. The source sync daemon scans all bitmaps generated, and creates an offset-epoch-pair table, called Final Epoch Table (FET). FET records the epoch number for each block that indicates when the final update happened.
2. The source sync daemon sends FET to the destination.
3. The destination sync daemon compares FET and CET to determine which cached block is valid. For a given block offset, if the epoch number of FET is equal to that of CET, the cached block of the offset is consistent and valid. The destination sync daemon creates Cached Bitmap, the bitmap of successfully-cached blocks.
4. With cached blocks and Cache Bitmap on hand, the I/O and sync daemons in the destination initialize postcopy storage migration.

Then, the memory migration mechanism resumes the VM at the destination. The I/O daemon serves disk I/O for the VM. Concurrently, the sync daemon transfers not-yet-cached blocks from the source. After all blocks are cached in the destination, there will be no dependency with the source, and we can safely shutdown physical servers in the source site.

Step 3 is explained through an example in Fig. 5. The block (offset 1) has the epoch number 11 in FET, which means the block had final updates during the epoch 11 - i.e., it was not updated in the epoch 12 and later. At the destination, the block has the epoch number 11 in CET, which means the block was cached after the bitmap of the epoch 11 had been frozen, which could only have happened in the epoch 12 or later. In this case, the cache of the block con-

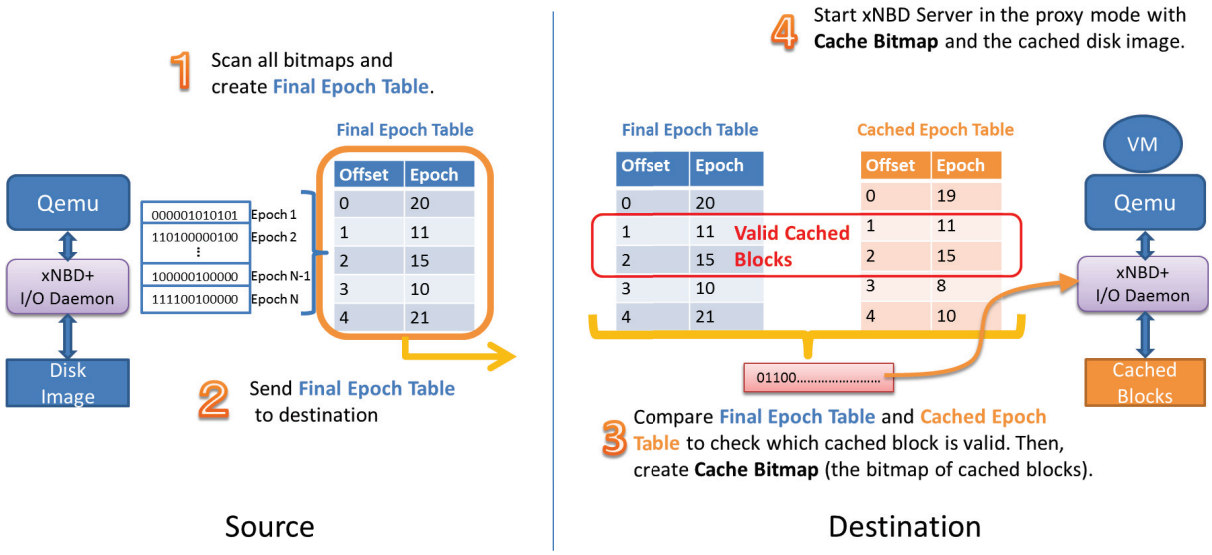

Fig. 5 The way of validating cached blocks upon a live migration.

tains the same data as the block at the source disk image.

3.3 Implementation

We have developed the prototype of the proposed mechanism `xNBD+`, extending our previously developed `xNBD` storage migration system. Since `xNBD` only supports storage migration without pre-caching of disk blocks, we have newly implemented the sync daemon to synchronize disk data in advance, and extended the I/O daemon to only transfer the necessary blocks. We have refactored `xNBD` and added approximately 3000 lines of C code.

4. Evaluation

We conducted experiments to confirm that the prototype works as expected reducing the time required to migrate the whole state of a VM. We compared our prototype with the native live storage migration implementation of Qemu/KVM. We used the latest stable release of Qemu (`qemu-1.2.1`) and the KVM driver of Linux Kernel 2.6.32 on the host operating system. The setup can migrate virtual disks as well as the memory and system states of a VM when the `-b` option is added to the `migrate` command of the Qemu monitor console. It implements a precopy migration algorithm. Unlike our mechanism, the migration mechanism of Qemu does not pre-cache disk blocks in advance before a migration is invoked. We compared two configurations: (1) we used the Qemu's migration mechanism *without* storage support, and also used our storage migration mechanism `xNBD+`; (2) we used the Qemu's migration mechanism with storage support.

Figure 6 shows the network setting used in experiments. A VM is started at a source node, and then migrated to a destination node over an emulated WAN. Between the source and destination nodes, we inserted a network emulator node that adds a network latency and a bandwidth limitation. It uses the traffic control mechanism of Linux (i.e., the `tc` command). In both configurations with or without using the proposed storage migration mechanism, `xNBD+` provides disk I/O for the VM at the source and destination nodes.

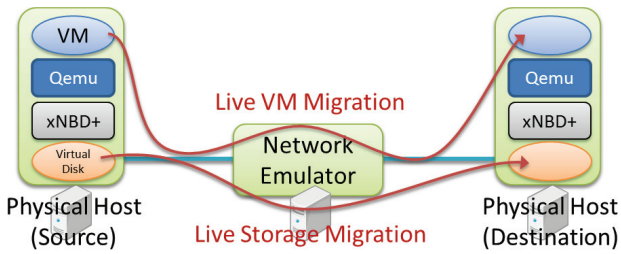

Fig. 6 The network setting of experiments.

The VM is configured to have 1VCPU, 512MB memory, and a 4GB virtual disk. The network between the source and destination nodes is configured to emulate a long-fat network such as a network between Japan and US in mind. The network emulator node sets a 100ms RTT on a 1Gbps network link. The migration code of Qemu allows users to configure the acceptable downtime of a migration. In the experiments, we have set the acceptable downtime to 1 second. We consider that the default value of the acceptable downtime, 30ms, will be unachievable in WAN environments, as long as the precopy-based migration algorithm is used in Qemu/KVM. Note that this limitation is not imposed nor is due to the proposed storage migration mechanism. In this paper, we chose 1 second for the parameter, which will not visibly affect applications in most of situations. We are currently working on this issue by introducing a precopy/postcopy hybrid algorithm to Qemu/KVM. Further details will be reported in our future work.

The system has been configured to increment the epoch value every 10 seconds. This interval is a configurable pa-

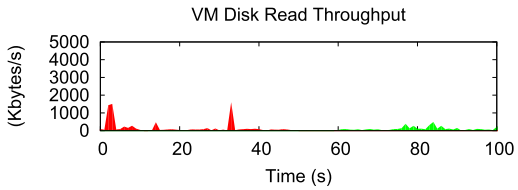

Fig. 7 VM disk read throughput. The red area shows disk I/O performed in the source host and the green area shows disk I/O performed in the destination.

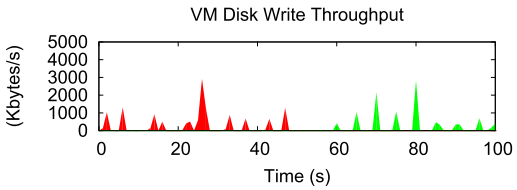

Fig. 8 VM disk write throughput. The red area shows disk I/O performed in the source host, and the green area shows disk I/O performed in the destination.

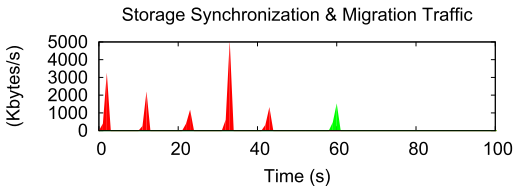

Fig. 9 Storage synchronization traffic and storage migration traffic. The red area shows storage synchronization traffic, and the green area shows storage migration traffic.

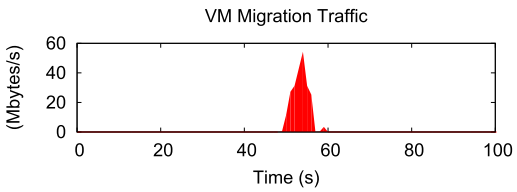

Fig. 10 Migration traffic of VM memory.

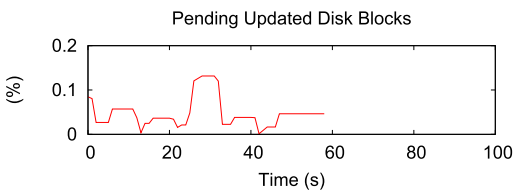

Fig. 11 Pending updated disk blocks.

rameter of the prototype. A smaller value will reduce migration time when evacuating VMs, but will increase synchronization traffic during normal operation. As justified below, 10-seconds interval is an appropriate trade-off point for the situations we tested.

4.1 Basic Tests

First, we conducted experiments to confirm the basic functionality of the prototype. During this experiment, the Linux kernel compilation was executed on the guest operating system. As shown in Fig. 7 and Fig. 8, the VM was sporadically reading source files from the virtual disk and writing back object files to it. Figure 9 shows disk synchronization traffic produced by the proposed mechanism. It periodically transfers updated blocks to the destination. Figure 11 shows the number of the updated blocks that are not yet synchronized with the destination node, which was always lower than 0.2% of the total disk blocks. This means that the pending data size, which must be transferred upon a migration, was lower than 8Mbytes at any given point in time.

At the time of 50 seconds, we started the VM live migration. Figure 10 shows the migration traffic produced by VM memory migration. The first spike was the iterative copy phase of the migration algorithm, and then the second, tiny spike was the final phase where the rest of memory pages and VM states except disk blocks were transferred. As shown in Fig. 9, not-yet-synchronized disk blocks were transferred at the time of 58 seconds, which took only approximately 1 second. In total, this live migration including storage migration was completed in about 10 seconds. In another experiment, we used the Qemu's migration mechanism with storage support instead of our proposed mechanism. In this case, the total migration time of memory and disk blocks took over 200 seconds.

The results confirm that the proposed mechanism can significantly reduce the VM migration time by means of pre-synchronization of virtual disk storage. The migration time was reduced to 5.4% of the case without our mechanism.

It should be noted that in the current implementation of Qemu, only one thread handles both the data transfers of disk I/O and migration. Since migration traffic tends to be much burstier than disk I/O traffic, the thread consumes a substantially large CPU time around the migration code. As shown in the above experiment, while the memory migration was on-going, the read/write performance of the virtual disk was suppressed. This problem is not specific to our migration mechanism, and will be resolved in a future release of Qemu/KVM.

4.2 Application Workloads

After validating the prototype, we performed experiments with other types of applications running on the guest operating system. Table 1 summarizes the results. *idle* represents the case without active applications. *kernel* means that the Linux kernel compilation process was running (i.e., the same as Sect. 4.1). *pgbench* is the case running the `pgbench` database benchmark program.

In all cases, our mechanism drastically reduced the migration time. For the idle workload, our mechanism was able to migrate the entire VM in 7.4 seconds, which was a dramatic reduction from 197.8 seconds required by the Qemu's migration mechanism. Similarly, for the kernel compile workload, it achieved 11.3 seconds in contrast to 206.4 seconds by the Qemu's mechanism. Remarkably,

Table 1 Total migration time under idle, kernel compile, and pgbench workloads. The proposed mechanism (`migrate & xNBD+`), the Qemu's migration mechanism with storage support (`migrate -b`), and the mechanism without pre-caching (`migrate & xNBD`) are compared. (Seconds)

workload	`migrate & xNBD+` (proposed mechanism)			`migrate -b`	`migrate & xNBD` (no pre-caching)		
	total	memory	disk		total	memory	disk
idle	7.4	7.3	0.1	197.8	185.0	5.0	180.0
kernel	11.3	9.8	1.5	206.4	184.8	7.3	177.5
pgbench	16.5	14.9	1.6	not finished	201.0	13.6	187.4

our mechanism successfully finished the live migration with the pgbench workload, while the Qemu's migration mechanism has failed. The pgbench program continuously updated disk blocks; the disk write throughput was always over 2Mbytes/s and sporadically went up beyond 10Mbytes/s. The Qemu's storage migration mechanism, which is based on a precopy algorithm, was unable to reduce the pending blocks below the threshold value. In contrast, our proposed storage migration mechanism, which is based on a postcopy algorithm, can always finish a migration in a finite period of time.

Table 1 also shows the elapsed times of the memory part and the storage part of a migration. Since our proposed mechanism periodically synchronized disk blocks in advance, the elapsed time of the storage migration was only 1.6 seconds for the pgbench workload. In this case, the data size of pending disk blocks at the beginning of the storage migration was only 2.9Mbytes, which allowed the prompt completion of the storage migration.

Without pre-caching, approximately 180 seconds was needed to complete the disk migration in all tested workloads. The time required to transfer a disk using postcopy storage migration can be roughly calculated by dividing the size of the disk with the available TCP bandwidth - the transferred data size is deterministic, and most disk blocks are transmitted through data bursting. In theory, the type of workload running on a VM has no influence in this storage migration. Without pre-caching, migration times were basically smaller compared to those of the Qemu's mechanism, but much larger than those with pre-caching, clearly showing that pre-caching mechanism greatly contributed in the reduction of migration times.

4.3 I/O-intensive Workloads

In order to properly benchmark the storage migration process without having the influence of memory updates, we developed a micro benchmark program that intensively updates disk blocks. We carefully designed the program to only touch disk blocks with minimum updates to memory pages. The program creates a 1Gbytes scratch file with the `DIRECT_IO` option, which forces the kernel to bypass the page cache operation. It repeatedly updates disk blocks from the first to the last offset of the scratch file. The updates frequency is controllable through parameter settings.

While running the micro benchmark program in the guest operating system, we performed a VM migration by using our proposed mechanism. Figure 12 shows the total migration time and the memory migration part of it. As expected, the time required for memory migration was approximately constant at 8 seconds in all experiments. The time taken by the storage migration increased as the disk write speed went up. However, even in the case of the highest write speed (69Mbytes/s), the total migration time was only 40 seconds, showing that the synchronization mechanism greatly contributed to the reduction of the total migration time.

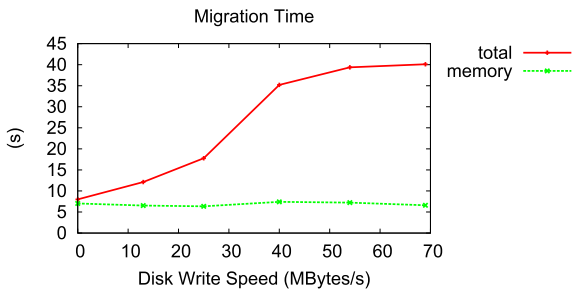

Fig. 12 Migration time under different disk write speeds in the case of our proposed mechanism. `total` means the total migration time including both the memory and storage parts. `memory` means the migration time of only the memory part. Note that the Qemu's migration mechanism with storage support could not finish any migrations tested with the disk write speed of 13Mbytes/s and more. Without pre-caching, approximately 180 seconds was needed to complete the disk migration part in all tested workloads.

Thanks to the postcopy approach of our storage migration mechanism, the maximum size of unsynchronized blocks will be capped by that of the disk blocks of the current working set independently of the disk write speed. In this experiment, the size of the scratch file was 1Gbytes and the average throughput of storage migration traffic was around 200Mbps. The ceiling of the storage migration time will be around 40-50 seconds; this is a rough estimate considering 40 seconds (1Gbytes / 200Mbps) of the storage migration part, plus 8 seconds of the memory migration part. It should be noted that the Qemu's migration mechanism with storage support could not finish any migrations with the tested disk write speeds over 13Mbytes/s.

Through the above experiments, we confirmed that 1) the pre-caching mechanism greatly contributes to the reduction of migration time and 2) the postcopy mechanism keeps migration time at manageable values even under I/O intensive workloads. We also observed that the CPU utilization of the NBD sync daemon was approximately 5 even in the case of the highest write speed (69Mbytes/s). The CPU of the physical machine is an Intel Core2 Quad Q9400

2.66GHz and the memory size is 16Gbytes. The overhead of synchronization is relatively small considering the modern hardware, and the impact on VM performance will be negligible in most cases. Storage read operations periodically performed by the NBD sync daemon can potentially incur resource contention of disk I/O. However, through the experiments, we observed that the disk contents requested by the sync daemon tended to exist in the page cache of the host operating system, because these blocks were recently written by the VM (through the host operating system/hypervisor). Thus, these read operations would hit the page cache avoiding the risk of disk I/O contention. Although performance impacts are not zero, we believe they will be minimal in most practical use-cases.

4.4 Different Network Conditions

4.4.1 Emulated Network Conditions

In order to evaluate the proposed mechanism under different network conditions, experiments using the kernel compile workload were conducted using a variety of network latencies. Figure 13 shows total migration times with different network latencies. Our proposed mechanism successfully reduced the total migration times under any network latencies. Even in the case that the round-trip time was 150ms, the total migration time was only 16 seconds, which was far less than 283 seconds in the case of the Qemu's migration.

The Qemu's migration mechanism required at least 150 seconds even for the cases of low network latencies. Because the kernel compile workload continuously updates disk blocks, the precopy algorithm repeatedly performed the iterative copy phase. Even without pre-caching, the post-copy algorithm achieved shorter migration times compared to the Qemu's migration. However, these migration times were much larger than those when pre-caching was enabled, highlighting the importance of the pre-caching mechanism to reduce the migration times in all the tested conditions.

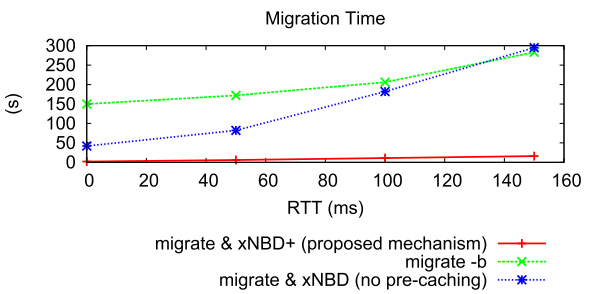

Fig. 13 Total migration time under different RTTs. The kernel compile workload was used. Our proposed mechanism (`migrate & xNBD+`), the Qemu's migration mechanism with storage support (`migrate -b`), and the proposed mechanism without pre-caching (`migrate & xNBD`) are compared.

We also compared the different link speeds of the emulated network by adding the experiments with the network link speed of 100Mbps. The kernel compile workload was used, and the RTT of the network was set to 100ms. As shown in Table 2, our mechanism dramatically reduced the total migration time in the 100Mbps experiments, from 407.8 seconds to 21.5 seconds. Since the size of not pre-cached blocks was quite small, our storage migration was able to promptly complete in both link speeds.

The memory migration times of the proposed mechanism were slightly longer than the cases without pre-caching, e.g., 2.5 seconds in the 1Gbps condition. After stopping the synchronization operation, our proposed mechanism generates Cache Bitmap as explained in Sect. 3.2. As for Table 2, the elapsed time of this procedure is included in the memory migration time field.

4.4.2 The Real Internet between Japan and US

Finally, we conducted experiments using the real Internet between Japan and United States. We migrated the same VM with the kernel compile workload from the AIST Tsukuba site to University of Florida. The round-trip latency was approximately 180ms, and the TCP network throughput measured by `iperf` was fluctuating from 2Mbps to 32Mbps.

Our proposed mechanism completed a live migration of the whole VM state in just 39.7 seconds, while the Qemu's migration with storage support required 1590.5 seconds (i.e., 26 minutes and 30.5 seconds). Without pre-caching, a live migration took 394.0 seconds. While the absolute values depends on network conditions and are likely to change, our proposed mechanism achieved a dramatic reduction of migration time in a real experimental setup with a very high-latency network. The migration time was only 2.5% of the Qemu's migration case. The pre-caching mechanism contributed the further 90% reduction (i.e., 394.0 to 39.7 seconds) of the migration time.

Conducting the same experiments with the pgbench workload, we observed that the Qemu's migration could not successfully live-migrate a VM. Because the disk write

Table 2 Total migration time under different link speeds. The kernel compile workload was used. RTT was 100ms. The proposed mechanism (`migrate & xNBD+`), the Qemu's migration mechanism with storage support (`migrate -b`), and the mechanism without pre-caching (`migrate & xNBD`) are compared. (Seconds)

link speed	`migrate & xNBD+` (proposed mechanism)			`migrate -b`	`migrate & xNBD` (no pre-caching)		
	total	memory	disk		total	memory	disk
1Gbps	11.3	9.8	1.5	206.4	184.8	7.3	177.5
100Mbps	21.5	19.3	2.2	407.8	386.8	17.2	369.6

speed of the VM was higher than that of the migration bandwidth, the precopy migration algorithm could not reduce the remaining size of the VM states to a sufficiently small value in order to finalize the migration process. On the other hand, our proposed mechanism was able to complete the migration under the same conditions in 88.9 seconds. Without pre-caching, the migration process took 654.1 seconds.

On the real Internet, we confirmed that the postcopy storage migration algorithm enables a finite migration time for the tested workloads, and that the pre-caching mechanism can further reduce the migration time.

5. Related Work

5.1 Disk Synchronization for Wide-Area Migration

CloudNet [13] studied a software platform of wide-area VM migration, which shares some technical aspects with our study. For storage migration, they utilize DRBD [14], a disk mirroring system over a network. In normal operation, their system works without a secondary node of DRBD, not performing disk mirroring. All written blocks are stored only in its primary server at a source site. When initializing the whole state migration, the system adds a secondary DRBD node at a destination site, and starts disk mirroring between them. This disk mirroring is performed in the background of disk I/O of the VM. Once disk blocks are completely synchronized, the system performs memory migration. After the memory migration is finished, the system stops disk mirroring and remove the primary storage node at the source.

In our project, it would be possible to exploit DRBD for synchronizing disk blocks in normal operation. If a VM requires strict data backup as well as minimizing an evacuation time, we consider that DRBD is a good option to be used. However, many use-cases of IaaS clouds will not require strict data backup at the sacrifices of performance degradation. Assuming that there is a time window to evacuate VMs upon a disaster, we can take more relaxed synchronization approach, which does not strictly synchronize but pre-caches disk blocks on a best-effort basis. Our proposed mechanism is intended to minimize performance degradation in normal operation and safely evacuate as many VMs as possible in disaster operation.

5.2 WAN-Optimized Migration Systems

There are several studies that enables the migration of the whole VM state, including virtual disks, to a remote site. [15] and [16] implemented a precopy-based block migration for Xen. After a migration is initialized, disk blocks and memory pages are transferred to the destination. When the size of VM states becomes lower than a threshold value, the VM is stopped, the rest of the states are transferred, and the VM is resumed at the destination. The downside of the precopy approach is that a migration time depends on how intensely the VM updates memory pages and disk blocks. Updated memory pages and disk blocks during a migration must be transferred repeatedly. As stated in Sect. 3, we took a postcopy approach for storage migration in disaster operation, which reduces the migration time and network traffic.

In order to reduce the data size of storage migration, [17] employs a copy-on-write disk mechanism. A template disk image is stored at both source and destination nodes in advance. All written blocks by a VM are saved in other disk spaces than the template image. Upon a storage migration, only updated blocks are transferred to destination. Obviously, this mechanism will effectively reduce migration traffic and a migration time, if the number of updated blocks is small. However, if normal operation continues for a long time, the number of updated blocks will be large, which results in a long migration time. Our proposed mechanism allows a system to periodically update cached blocks at destination in response to VM activity during normal operation.

CloudNet also implemented a data compression mechanism in the iterative memory copy phase of a precopy live migration. In the iterative copy phase, if a memory page that has already been transferred is updated, only the updated part of the page is transferred to destination. [18] studied a scheduling algorithm that reduces storage migration times. It computes an efficient ordering of disk block transfers by analyzing I/O locality of workloads. Shrinker [19] developed a data deduplication mechanism for virtual cluster migration. It calculates a content-based hashing value for each memory page and disk block of all VMs, and skips to transfer the memory page or disk block with the same hash value between two sites. The use of these techniques would further improve our project.

5.3 High Availability Systems

High availability (HA) systems using VM technologies have been developed to provide uninterruptible IT services upon sudden hardware and power failures([20], [21]). They synchronously replicate software states between primary and backup servers. When a primary server fails, they promptly activate the backup server. In an ideal case, this fail-over operation is not noticeable from the outside of this HA system. However, they require a low-latency and high-bandwidth interconnect between primary and backup servers, in order to synchronously replicate software states.

Several studies developed a mechanism that reduces synchronization traffic over a WAN ([22], [23]). They relaxed checkpointing intervals and exploited a logging-replay technique. For example, [23] captures disk I/O operations at a primary server and replays them in a backup server. It is not necessary to transfer memory states updated by disk read operations. Instead, it transfers read requests to the backup server, e.g., *read 10 blocks from disk offset 100 to memory page 0x12345*. In most cases, the size of transferred messages is much smaller compared to memory states.

These studies contribute to alleviating performance degradation. However, it is not easy to deploy these systems in a WAN environment where network latencies are over tens of milliseconds. Although there are use-cases where

HA systems need to be used in a WAN at the sacrifices of performance degradation, our project aims to cover other use-cases where HA systems do not meet economical or performance requirements.

6. Discussion

6.1 Design Decisions

In this section, we present the design decisions made for our storage migration mechanism. Before our project, live storage migration was available only in LAN environments. [24] describes the storage migration techniques used in their hypervisors. Dirty block tracking was used to implement a precopy-based storage migration. In the later version of their hypervisor, the I/O mirroring mechanism was used during storage migration; concurrently to bulk transfer of the entire disk image, update disk blocks are iteratively committed to destination storage. These mechanisms, however, were designed for LAN environments. With WAN environments and disaster recovery scenarios in mind, we carefully considered whether we should exploit or not these techniques.

6.1.1 Dirty Block Tracking v.s. Optimistic Best Effort Synchronization

The dirty block tracking mechanism of [24] is intended to be working only during a migration. It is designed to finish data transfer as soon as possible, using as much as possible the available network bandwidth. On the other hand, our pre-caching mechanism works 24/7 until a disaster happens. It is important not to adversely affect daily performance of a VM.

The dirty block tracking mechanism is typically implemented as a busy loop composed of dirty bitmaps scanning and data transfer processes. On the other hand, our pre-caching mechanism sporadically scans dirty bitmaps and transfers update data periodically (e.g., every 10 seconds). This behavior mitigates an impact on the available bandwidth of a WAN and the CPU resource of the host operating system during regular operation.

In addition, our pre-caching mechanism is designed to be robust against unstable WAN environments. All synchronization messages include an epoch number, which is used to detect potential data loss in the backup site due to network connection failures or synchronization bandwidth shortages. When the connection is recovered, synchronization is restarted with already pre-cached blocks. On the other hand, the dirty block tracking is designed for stable LAN environments. If network connectivity is lost during data transfer, all pre-cached data is discarded.

6.1.2 I/O Mirroring v.s. Postcopy Storage Migration

We consider that the I/O mirroring technique of [24] can be potentially used in our disaster operation. It intercepts all storage write requests and commits them also in the destination storage. In parallel, the disk image is copied to the destination. The disk I/O speed of the VM is throttled down to that of write commits on the destination. The migration time does not increase even for an IO-intensive workload. This characteristic contributes to a shorter migration time in the same manner as postcopy storage migration does.

A potential downside of the I/O mirroring technique is that its application to WAN environments has not been discussed in [1]. Probably, the synchronous write commits over a WAN will greatly degrade the disk I/O performance of the VM. Although there will be potential performance degradation also in postcopy storage migration over a WAN, we observed that in some cases disk I/O performance was recovered promptly because of the affinity of disk I/O offsets; most disk I/O requests are issued for particular regions of a disk image (i.e., hot spots), and once host spots are copied by the postcopy mechanism, performance degradation will be greatly alleviated. Further discussion will be given in future work.

The I/O mirroring technique itself is reliable for disconnection of a network. However, we consider that more discussion would be necessary to study how a storage migration mechanism contributes to the reliability of a disaster recovery system.

There is a potential risk that the network reachability to the disaster site is lost during a migration. If the postcopy storage migration is on-going, the disk I/O of the VM will be blocked until the network connection is recovered. In this case, we can suspend the VM and save the state of the VM to non-volatile storage. After the network is recovered, we can resume the execution of the VM. This assumption is based on the results of our former study [1], where we observed IT systems of data centers were not physically damaged due to a severe earthquake.

Even though the I/O mirroring technique is used, this kind of fallback solution is necessary. If the network is lost before all VMs are evacuated, we need to save remaining VMs to non-volatile storage on the disaster site before a power backup system gets down.

In this paper, we have discussed about our storage migration mechanism. We are also working on a feedback control mechanism optimizing concurrent wide-area migrations. We are also developing a programming framework that can simulate the evacuation of thousands of VMs. More realistic feasibility study integrating these technologies will be done in our future work. Detail of how our storage migration mechanism is integrated with a VM management system will be discussed there.

To the best of our knowledge, the I/O mirroring mechanism is not available for Qemu/KVM, and our postcopy storage migration mechanism is the only one that provides such a short migration time.

6.1.3 Data Backup v.s. Server Evacuation

Our system is intended to provide a mechanism to evacuate

VMs upon a disaster and continue IT services on safe locations. It does not cover data loss due to a misoperation by an administrator or a program bug. We assume that an administrator will make a backup of his system to save important data, e.g., daily backup of a database. Our system is basically independent of backup systems. An administrator can use any backup system as well as our system.

It is possible to extend our system to save a consistent disk image at a point of time. We want to investigate a feasible design for this feature through future development.

7. Conclusions

In this paper, we propose a WAN-optimized live storage migration mechanism that can relocate the entire VM state to a remote site just in tens of seconds. During normal operation, it proactively caches the state of a virtual disk to a backup site. When VM migration is triggered, our system performs live storage migration of not-yet-cached blocks concurrently with the memory and system state relocation. By transferring the data in advance, the proposed mechanism can dramatically reduce the amount of data transferred and consequently the time required to complete the live migration of the whole VM state. We developed a prototype of the proposed mechanism extending our postcopy-based storage migration system. Through experiments, we confirmed that the proposed mechanism dramatically reduced the live migration time of the whole VM state running various workloads under different network conditions. Using a real system, it was possible to relocate a VM from Japan to the United States in just 39.7 seconds - a 97.5% reduction of migration time from the case without our mechanism.

This research project is partially supported by JST/CREST ULP, KAKENHI 23700048, and NSF/JST RAPID.

References

[1] M. Tsugawa, R. Figueiredo, J. Fortes, T. Hirofuchi, H. Nakada, and R. Takano, “On the use of virtualization technologies to support uninterrupted IT services,” ICC2012 Workshop on Re-think ICT infrastructure designs and operations (Accepted), pp.1–5, June 2012.

[2] C. Clark, K. Fraser, S. Hand, J.G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, “Live migration of virtual machines,” Proc. 2nd Symposium on Networked Systems Design and Implementation, pp.273–286, 2005.

[3] M. Nelson, B.H. Lim, and G. Hutchins, “Fast transparent migration for virtual machines,” Proc. USENIX Annual Technical Conference, pp.25–25, 2005.

[4] A. Mirkin, A. Kuznetsov, and K. Kolyshkin, “Containers checkpointing and live migration,” Proc. Linux Symposium, pp.85–92, July 2008.

[5] T. Hirofuchi, H. Nakada, H. Ogawa, S. Itoh, and S. Sekiguchi, “A live storage migration mechanism over wan and its performance evaluation,” Proc. 3rd International Workshop on Virtualization Technologies in Distributed Computing, pp.67–74, Press, June 2009.

[6] T. Hirofuchi, H. Nakada, S. Itoh, and S. Sekiguchi, “Kagemusha: A guest-transparent mobile IPv6 mechanism for wide-area live VM migration,” Proc. IEEE/IFIP Network Operations and Management Symposium 2012, pp.1319–1326, April 2012.

[7] D.B. Johnson, C.E. Perkins, and J. Arkko, “Mobility Support in IPv6,” RFC 3775 (Proposed Standard), June 2004.

[8] T.S. Kang, M. Tsugawa, J.A. Fortes, and T. Hirofuchi, “Reducing the migration times of multiple vms on wans,” SC12 ACM Student Research Competition Poster Session, Nov. 2012.

[9] P.T. Breuer, A.M. Lopez, and A.G. Ares, “The network block device,” 1999.

[10] A. Kivity, Y. Kamay, D. Laor, and A. Liguori, “kvm: the Linux virtual machine monitor,” Proc. Linux Symposium, pp.225–230, 2007.

[11] 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,” Proc. Nineteenth ACM Symposium on Operating Systems Principles, pp.164–177, 2003.

[12] I. Yamahata and T. Hirofuchi, “Yabusame update on postcopy live migration for QEMU/KVM,” KVM Forum 2012, Nov. 2012.

[13] T. Wood, P. Shenoy, K.K. Ramakrishnan, and J. van der Merwe, “CloudNet: Dynamic pooling of cloud resources by live wan migration of virtual machines,” Proc. 7th ACM SIGPLAN/SIGOPS international conference on Virtual execution environments, pp.121–132, March 2011.

[14] P. Reisner and L. Ellenberg, “Drbd v8: Replicated storage with shared disk semantics,” Proc. 12th International Linux System Technology Conference, pp.1–11, May 2005.

[15] R. Bradford, E. Kotsovinos, A. Feldmann, and H. Schiöberg, “Live wide-area migration of virtual machines including local persistent state,” Proc. 3rd International Conference on Virtual Execution Environments, pp.169–179, 2007.

[16] Y. Luo, B. Zhang, X. Wang, Z. Wang, and Y. Sun, “Live and incremental whole-system migration of virtual machines using block-bitmap,” Proc. Cluster 2008: IEEE International Conference on Cluster Computing, 2008.

[17] C.P. Sapuntzakis, R. Chandra, B. Pfaff, J. Chow, M.S. Lam, and M. Rosenblum, “Optimizing the migration of virtual computers,” ACM SIGOPS Operating System Review, vol.36, no.SI, pp.377–390, 2002.

[18] J. Zheng, T.S.E. Ng, and K. Sripanidkulchai, “Workload-aware live storage migration for clouds,” Proc. 7th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, pp.133–144, 2011.

[19] P. Riteau, C. Morin, and T. Priol, “Shrinker: Improving live migration of virtual clusters over wans with distributed data deduplication and content-based addressing,” Proc. 17th international conference on Parallel processing - Volume Part I, pp.431–442, 2011.

[20] Y. Tamura, K. Sato, S. Kihara, and S. Moriai, “Kemari: Virtual machine synchronization for fault tolerance,” USENIX08 Poster, 2008.

[21] B. Cully, G. Lefebvre, D. Meyer, M. Feeley, N. Hutchinson, and A. Warfield, “Remus: high availability via asynchronous virtual machine replication,” Proc. 5th USENIX Symposium on Networked Systems Design and Implementation, pp.161–174, 2008.

[22] S. Rajagopalan, B. Cully, R. O’Connor, and A. Warfield, “Secondsite: disaster tolerance as a service,” Proc. 8th ACM SIGPLAN/SIGOPS conference on Virtual Execution Environments, pp.97–108, 2012.

[23] O. Kei, “Rapid VM synchronization with I/O emulation logging-replay,” KVM Forum 2011, 2011.

[24] A. Mashtizadeh, E. Celebi, T. Garfinkel, and M. Cai, “The design and evolution of live storage migration in VMware ESX,” Proc. USENIX conference on USENIX annual technical conference, pp.14–14, 2011.

Takahiro Hirofuchi is a researcher of National Institute of Advanced Industrial Science and Technology (AIST) in Japan. He is working on virtualization technologies for advanced cloud computing and Green IT. He obtained a Ph.D. of engineering in March 2007 at the Graduate School of Information Science of Nara Institute of Science and Technology (NAIST). He obtained the BS of Geophysics at Faculty of Science in Kyoto University in March 2002. He is an expert of operating system, virtual machine, and network technologies.

Mauricio Tsugawa received his BS and MS degrees in Electrical Engineering from the Universidade de Sao Paulo in 1998 and 2001 respectively, and his PhD degree in Electrical and Computer Engineering from the University of Florida in 2009. He joined the faculty of the Department of Electrical and Computer Engineering at the University of Florida as Research Scientist in 2009. His research interests include computer networks, distributed computing, computer architecture and virtualization technologies.

Hidemoto Nakada received his Ph.D. degree from the University of Tokyo in 1995. He joined the Electrotechnical Laboratory in 1995, which was merged into the National Institute of Advanced Industrial Science and Technology (AIST) in 2001. He also served as a visiting associate professor at the Tokyo Institute of Technology from 2001 to 2005. His interests lie in the area of parallel / distributed computing, including Grid and Cloud technologies. He is a member of ACM and IPSJ.

Tomohiro Kudoh received his Ph.D. degree from Keio University in Japan in 1992. He joined the National Institute of Advanced Industrial Science and Technology (AIST) in 2002. He currently serves the deputy director of the Information Technology Research Institute, AIST. In the past few years his research has focused on the network as a Grid infrastructure. His recent work also includes the G-lambda project, which is attempting to define an interface to manage the network as a Grid resource.

Satoshi Itoh obtained a Ph.D. in physics from University of Tsukuba, Japan, in 1987. From 1987 to 2002 he worked for high-performance and parallel computing in the both area of material science and business application at Central Research Laboratory, Hitachi, Ltd. In 2002, he moved to National Institute of Advanced Industrial Science and Technology (AIST), Japan and has researched on Grid computing, Cloud computing, and Green IT. He is currently the Deputy Director of Information Technology Research Institute, AIST.