The Continuity of Out-of-band Remote Management Across Virtual Machine Migration in Clouds

Sho Kawahara
Department of Creative Informatics
Kyushu Institute of Technology
Fukuoka, Japan
kawasho@ksl.ci.kyutech.ac.jp

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

***Abstract*—In Infrastructure-as-a-Service (IaaS) clouds, users remotely manage the systems in virtual machines (VMs) called *user VMs*, e.g., through VNC. To allow users to manage their VMs even on failures inside the VMs, IaaS usually provides *out-of-band* remote management. This VM management is performed indirectly via a VNC server in a privileged VM called the *management VM*. However, it is discontinued when a user VM is migrated from a source to a destination host. This is because a VNC server in the management VM at a source host is terminated on VM migration. Even worse, pending data is lost between a VNC client and a user VM. In this paper, we propose *D-MORE* for continuing out-of-band remote management across VM migration. D-MORE provides a privileged and migratable VM called *DomR* and performs out-of-band remote management of a user VM via DomR. On VM migration, it synchronously co-migrates DomR and its target VM and transparently maintains the connections between a VNC client, DomR, and its target VM. We have implemented D-MORE in Xen and confirmed that a remote user could manage his VM via DomR even after the VM has been migrated. Our experiments showed that input data was not lost during VM migration and the overhead of D-MORE was acceptable.**

***Keywords*-IaaS clouds, virtual machines, remote management, VM migration**

I. Introduction

Infrastructure as a Service (IaaS) provides virtual machines (VMs) hosted in data centers. Users can set up the systems in the provided VMs called *user VMs* and use them as necessary. They usually manage their systems through remote management software such as VNC. To allow users to access their systems even on failures inside their VMs, IaaS often provides *out-of-band* remote management via a privileged VM called the *management VM*. Unlike usual *in-band* remote management, a VNC server is run in the management VM, not in a user VM, and directly interacts with virtual devices for a user VM, such as a virtual keyboard and a virtual video card. Even if the network of a user VM is disconnected due to user's configuration errors or if system failures occur in a user VM, a user can continue to manage such a VM. In addition, users can perform VM management more securely because IaaS is responsible for the security of out-of-band remote management.

However, out-of-band remote management is discontinued when a user VM is migrated from a source to a destination host. During the VM migration, virtual devices in the management VM at a source host are removed. Thereby a VNC server using the virtual devices is also terminated. To restart remote management, a user has to identify the reason of the disconnection, look for a destination host, and reconnect to a new VNC server at the host. This is troublesome for a user. Even worse, inputs and outputs for remote management can be lost by VM migration. Pending data in a VNC server and virtual devices is abandoned when virtual devices and a VNC server are terminated. Since the network connection between VNC client and server is also terminated, in-flight packets are lost and are not retransmitted. A user has to recover lost inputs by himself, but it may be difficult to even notice the data loss itself. This is critical when a user executes important tasks using out-of-band remote management.

In this paper, we propose *D-MORE*, which is the system for continuing out-of-band remote management across VM migration. Our idea is running a VNC server and virtual devices in a privileged and migratable VM called *DomR* and co-migrating DomR with its target VM. To achieve the continuity of remote management, D-MORE transparently maintains the connections between a VNC client, DomR, and its target VM at the levels of the network and the virtual machine monitor (VMM). The VMM is a software layer underlying VMs and manages the interaction between VMs. In addition, D-MORE can prevent the data loss of inputs and outputs for remote management. Pending data in a VNC server and virtual devices is preserved in DomR because it is migrated as a part of DomR. In-flight network packets are retransmitted by TCP even if they are dropped during VM migration.

We have implemented D-MORE in Xen 4.3.2 [1]. To run virtual devices in DomR, D-MORE allows DomR to establish shared memory by mapping memory pages of a user VM. It also allows DomR to establish interrupt channels with a user VM using Xen's event channels. During the co-

migration of DomR and its target VM, D-MORE restores the states of memory mapping and event channels at a destination host. To safely perform this state restoration at appropriate timings and prevent data loss in shared memory, D-MORE synchronizes the migration processes of the two VMs. We conducted several experiments to examine the continuity of out-of-band remote management and the performance of D-MORE. From our experimental results, it was shown that the remote management was continued across VM migration and that no data was lost during co-migration. The downtime during co-migration was acceptable.

The organization of this paper is as follows. Section II describes issues in out-of-band remote management using the management VM. Section III proposes D-MORE for continuing out-of-band remote management across VM migration. Section IV explains the implementation details in Xen and Section V shows our experimental results. Section VI describes related work and Section VII concludes this paper.

II. Motivation

A. Out-of-band Remote Management

To manage user VMs in IaaS clouds, a user usually connects a VNC client at user's host to a VNC server running in a user VM. This is called *in-band* remote management because a user accesses a VM using functionalities provided inside the VM. However, this in-band remote management is not powerful enough to manage a user VM in many cases. If a user has just failed the configurations of the network or firewall in a VM, he cannot manage the VM at all. At that time, he would have to abandon that VM and recreate a new VM from scratch. As another case, when a VNC server in a user VM is not running normally, a VNC client cannot access the VM. For example, a VNC server may crash due to bugs. It is not started until the operating system in a VM has been booted normally.

To enable users to manage their VMs in such cases, it is necessary for IaaS to provide *out-of-band* remote management. As illustrated in Fig. 1, a VNC server is run for each user VM in a privileged VM called the *management VM*. The management VM is often provided in type-I VMMs such as Xen and Hyper-V and has privileges for accessing all user VMs. It also provides virtual devices to each user VM, e.g., a virtual keyboard, a virtual mouse, and a virtual video card. A VNC server in the management VM directly accesses virtual devices to interact with a user VM. Out-of-band remote management does not rely on the network or a VNC server in a user VM. A user can access his VM as if he locally logged into the VM even on network failures in the VM. For example, even if a user fails network configuration in a user VM, he could fix the problem by modifying the configuration through a virtual keyboard in the management VM.

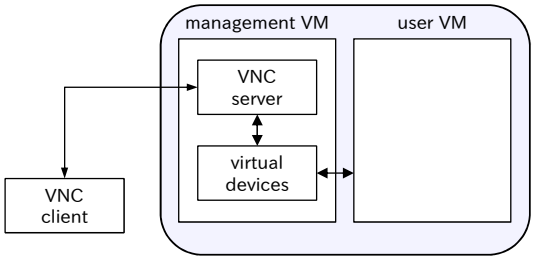

Figure 1. Out-of-band remote management of a user VM.

B. VM Migration in Out-of-band Remote Management

IaaS clouds migrate VMs for various purposes. VM migration allows a running VM to be moved between hosts. In particular, live migration [2] almost does not stop a VM during the migration process by transferring most of the states with the VM running. Using VM migration, IaaS providers can maintain physical hosts without interrupting services provided by VMs. They can perform load balancing by migrating heavily loaded VMs to other lightly loaded hosts. Conversely, they can save power if they consolidate lightly loaded VMs into a fewer hosts. In in-band remote management, it is possible to continue VM management across the migration of a user VM. A VNC server in a user VM is migrated as a part of the VM and the network connection between VNC client and server is preserved. At that time, a user using a VNC client is not aware that his VM is migrated.

However, in out-of-band remote management, VM management is discontinued on VM migration. When a user VM has been migrated, its virtual devices in the management VM at a source host are removed. The migrated VM uses new virtual devices created in the management VM at a destination host. At the same time, a VNC server in the management VM at a source host is terminated because it loses the access to the removed virtual devices. As a result, a VNC client is disconnected from the VNC server. To restart remote management, a big burden is imposed on users. First, a user has to identify the reason why a VNC client is disconnected. The possible cause is not only VM migration but also network failures or system failures in a user VM or the management VM. If the disconnection is due to VM migration, a user has to look for a destination host where a user VM has been migrated, e.g., from a management console provided in a cloud. Then he has to reconnect to a VNC server in the management VM at that host. This can lower the efficiency of the VM management.

Worse than that, keyboard and mouse inputs can be lost when a user VM is migrated. If input data has been sent from a VNC client but has not yet been received by a VNC server, in-flight network packets are dropped. Since the network connection between VNC client and server is terminated, dropped packets are not retransmitted at the network level.

Usually, a VNC client does not have a mechanism for data retransmission at the application level. If input data received by a VNC server has not yet been sent to a virtual device, it is lost by the termination of the VNC server. If input data received by a virtual device has not been sent to a user VM, it is also lost by the removal of the virtual device. When keyboard inputs are lost, a user has to type them again after the reconnection. However, it may be difficult to even notice the loss of inputs. As a critical example, when a user executes the reboot command to complete security updates, the input of the last enter key may be lost. While he believes that the system has been updated by the reboot, the real system is still vulnerable due to no reboot. This can largely affect the correctness of the VM management.

III. D-MORE

In this paper, we propose *D-MORE*, which can continue out-of-band remote management across the migration of user VMs. The system architecture of D-MORE is shown in Fig. 2. D-MORE provides a privileged and migratable VM called *DomR* for remote management of a user VM. DomR runs only a VNC server and virtual devices for its target VM. A VNC client connects to a VNC server in DomR and accesses a user VM through virtual devices in DomR. When a user VM is migrated, D-MORE co-migrates the corresponding DomR as well to the same destination host. Across the migration, D-MORE transparently maintains all the connections between a VNC client, DomR, and its target VM. Specifically, it preserves the connections between virtual devices in DomR and the target VM at the VMM level. In addition, it preserves the connection between a VNC client and a VNC server in DomR at the network level, as usual VM migration.

DomR has privileges necessary for running virtual devices. Traditionally, virtual devices could run only in the management VM because they need to access a user VM. First, DomR has a privilege for establishing shared memory with its target VM. Using buffers allocated in shared memory, virtual devices in DomR exchange data with device drivers in the target VM. For example, when a VNC server in DomR writes a keyboard input received from a VNC client to a virtual keyboard, the virtual keyboard writes it to a keyboard buffer in shared memory. A keyboard driver in the target VM reads the data from the keyboard buffer and processes it. Second, DomR has a privilege for establishing interrupt channels with its target VM. Via interrupt channels, virtual devices in DomR send virtual interrupts to device drivers in the target VM. For example, a virtual interrupt is sent when new data is written to the buffer in shared memory.

After the co-migration of DomR and its target VM, D-MORE reconnects DomR and its target VM. During VM migration, DomR is disconnected from the target VM once because any connections between VMs are lost except for

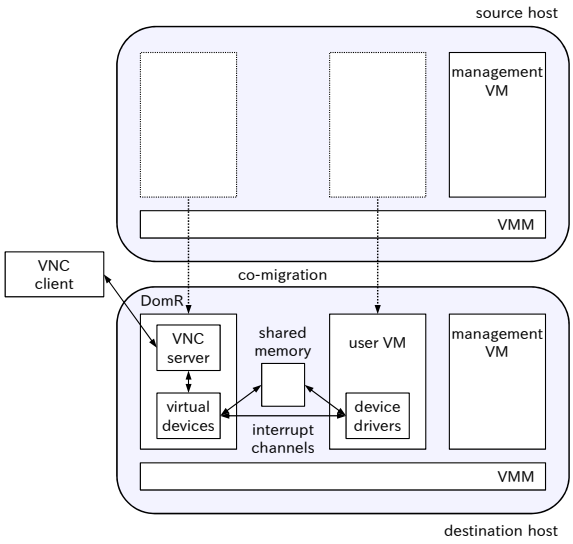

Figure 2. The system architecture of D-MORE.

network connections. Specifically, D-MORE re-establishes the shared memory and the interrupt channels between DomR and its target VM. To do this, D-MORE saves the states of memory sharing and interrupt channels at a source host. Then it transfers those states with the other states of VMs to a destination host. At a destination host, D-MORE restores the saved states transparently to DomR and the target VM. Consequently, virtual devices in DomR and device drivers in the target VM can continue their execution from arbitrary points even if they are accessing shared memory or interrupt channels.

D-MORE synchronizes two migration processes in the co-migration of DomR and its target VM for three purposes. The first purpose is to reconnect DomR and the target VM at appropriate timings. Since shared memory is constructed using the memory of the target VM, it is restored after the memory of the target VM has been restored. For consistency, interrupt channels are saved and restored while both VMs are stopped. The second purpose is to guarantee to transfer the latest data in shared memory to a destination host. Even if DomR modifies shared memory at any time, the migration process for the target VM is responsible for transferring its latest version. DomR must not modify it after the target VM has been migrated. The third purpose is to reduce the downtime of both DomR and the target VM by stopping them as late as possible.

Using D-MORE, no inputs or outputs for out-of-band remote management are lost during co-migration. First, pending data in a VNC server and virtual devices is preserved. Since a VNC server and virtual devices are migrated as a part of DomR, they can continue to process such pending data at a destination host. Second, it is guaranteed that data written to the buffer in shared memory is preserved by the above synchronization in co-migration. Third, in-

flight network packets from a VNC client to a VNC server are retransmitted by TCP although they may be dropped temporarily by migrating DomR. D-MORE preserves the TCP connection between VNC client and server by running a VNC server in DomR.

IV. Implementation

We have implemented D-MORE in Xen 4.3.2 [1]. In Xen, the VMM runs on top of hardware and executes VMs. The management VM is called *Dom0* and a user VM is called *DomU*. We have developed DomR by extending our guard VM [3], which can monitor the resources of DomU and can be migrated. DomR runs para-virtualized Linux for running virtual devices for DomU. In the current implementation, D-MORE supports DomU running para-virtualized Linux and targets the x86-64 architecture.

Supporting para-virtualized Linux is important because para-virtualization is often used with full virtualization to improve the performance. In Xen, for example, several modes are prepared as extensions of the HVM mode for full virtualization. The PV-on-HVM mode enables fully virtualized operating systems to use para-virtualized device drivers. The PVHVM mode uses para-virtualized interrupt channels instead of emulating hardware interrupt controllers. Recent Linux kernel supports these modes.

A. Virtual Devices in DomR

When DomU is booted, D-MORE binds the DomU to new DomR. To establish shared memory with target DomU, DomR maps memory pages of that DomU. Thereafter, both DomR and DomU can access the same memory pages. For this memory mapping, DomR invokes the VMM by issuing a hypercall with the page frame number corresponding to the memory page that DomR wants to share with DomU. We gave DomR a privilege for mapping memory pages of only target DomU.

In addition, DomR can establish event channels with DomU as para-virtualized interrupt channels. An event channel is a logical connection between two VMs and is used for sending events such as virtual interrupts. It consists of a local VM, a local port, a remote VM, and a remote port. After DomU allocates an event channel and obtains its port, DomR binds it using DomU's port. We gave DomR a privilege for intercepting event channels and establishing them with its target DomU. DomR can bind arbitrary event channels allocated by its target DomU. Without this privilege, only Dom0 can bind DomU's event channels because DomU specifies Dom0 as a remote VM of event channels.

DomR runs QEMU [4] customized for Xen to provide virtual devices and a VNC server necessary for out-of-band remote management. In traditional out-of-band remote management, QEMU runs in Dom0. For para-virtualized DomU, the split-driver model is used in Xen, as illustrated in Fig. 3. Virtual devices are implemented as backend drivers

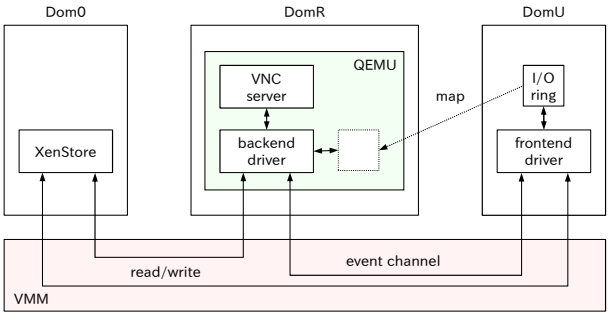

Figure 3. The split-driver model with DomR.

in QEMU. Frontend drivers in DomU communicate with the backend drivers to use virtual devices. For that communication, I/O rings and event channels are used. An I/O ring is a ring buffer for passing data and is allocated in shared memory. In addition, frontend and backend drivers exchange device configurations via XenStore in Dom0. XenStore is a filesystem-like database containing information shared between VMs.

For both a virtual keyboard/mouse and a virtual framebuffer (video card), the initialization of frontend and backend drivers is performed as follows. When a frontend driver in DomU connects to its backend driver in DomR, it first allocates a memory page for I/O rings and writes that page frame number to XenStore in Dom0. In addition, the framebuffer frontend driver allocates video memory. Next, a frontend driver allocates an unbound event channel and then writes the assigned local port number to XenStore. On the other hand, a backend driver in DomR watches XenStore and reads information written by a frontend driver in DomU. Traditionally, only Dom0 could read arbitrary entries in XenStore. We gave DomR a privilege for reading XenStore's entries only for its target DomU. From XenStore, a backend driver reads the frame number of the memory page containing I/O rings and maps that page. The framebuffer backend driver also maps the video memory allocated in DomU. Then a backend driver reads DomU's local port number of an event channel and binds the event channel. As such, it shares I/O rings and establishes an event channel with its frontend driver in DomU.

When a VNC server in DomR receives a keyboard or mouse input from a VNC client, it sends the input to the keyboard/mouse backend driver. The backend driver writes the input to an I/O ring for inputs in the shared memory. Then it sends an event to DomU to notify DomU of a new input in the I/O ring. When the keyboard/mouse frontend driver in DomU receives that event, it reads the I/O ring and obtains the input. On the other hand, when an application in DomU draws graphic objects, the framebuffer frontend driver updates its own video memory, which is shared with DomR. It writes an updated area to an I/O ring for outputs and sends an event to DomR. When the framebuffer backend

driver in DomR receives that event, it reads the I/O ring in the shared memory and obtains the update information. The update information is sent to a VNC server and the VNC server sends updated pixel data to a VNC client.

B. Reconnection between DomR and DomU

To maintain the connections between DomR and DomU after co-migration, D-MORE restores the mapping state of DomU's memory for DomR at a destination host, as shown in Fig. 4. Traditionally, it was not assumed to migrate a VM like DomR mapping other VM's memory. To migrate DomR, a migration manager running in Dom0 transfers the memory of DomR from a source to a destination host. In D-MORE, during this memory transfer, the migration manager at a source host inspects the page tables of DomR. Then it sets a monitor bit in a page table entry (PTE) where a memory page of DomU is mapped. The monitor bits are transferred together with the memory pages containing the page tables. At a destination host, the migration manager inspects the restored page tables of DomR. If a monitor bit is set in a PTE, the migration manager remaps the corresponding memory page of DomU to DomR. We have modified the VMM so that Dom0 could map a memory page of DomU to the corresponding DomR. For further details on this implementation, refer to our previous work [3].

D-MORE also restores the state of event channels at a destination host. Usually, all of the event channels are closed on VM migration because event channels are managed locally by the VMM. In D-MORE, the migration manager for DomR at a source host obtains a list of the event channels established between DomR and DomU. Then it transfers pairs of ports used for the event channels. At a destination host, the migration manager for DomR re-establishes event channels between DomR and DomU so that these VMs use the same pairs of ports as at a source host. It is guaranteed that any ports are still unused before a VM is restarted because event channels are allocated or bound by a VM itself. We have added two new hypercalls to the VMM for obtaining a list of event channels and establishing event channels with a pair of specified ports.

To reuse re-established event channels in the operating system kernels of DomR and DomU, we have modified the resume operation for virtual interrupts. In para-virtualized Linux kernel, an interrupt request (IRQ) is mapped to an event channel, and vice versa. In the original kernel, these mappings are discarded on resume because event channels are closed on VM migration. Since D-MORE has re-established event channels before the resume operation is executed, our kernel leaves the mappings only for re-established event channels. To do this, the kernel finds re-established event channels by issuing a hypercall to the VMM. For such event channels, it does not initialize the mapping between an IRQ and an event channel. This modification was done easily for only one function. As a result, QEMU in DomR and para-virtualized drivers in DomU can use re-established event channels via the same IRQs as before co-migration.

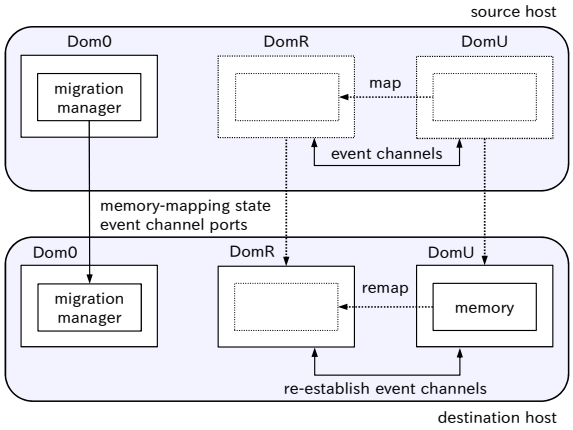

Figure 4. Restoring the states of memory mapping and event channels after co-migration.

Since D-MORE maintains the connections between DomR and DomU, we have also disabled the resume operations of keyboard/mouse and framebuffer frontend drivers in DomU. The original frontend drivers execute their resume handlers after DomU has been migrated. Each resume handler reinitializes I/O rings to prevent the inconsistency and reconnects to its backend driver by itself. Then several frontend drivers recover the data lost in I/O rings by re-executing pending requests. However, such recovery is difficult for the keyboard/mouse frontend driver because input data is not left anywhere. Therefore, our frontend drivers do not reinitialize I/O rings so that pending input and output data in I/O rings is not lost. I/O rings may be inconsistent just after co-migration, but they should become consistent by resuming DomR and DomU. Also, our frontend drivers do not reconnect to their backend drivers because their connections are preserved by D-MORE. These modifications to the drivers were also done easily by simply commenting out resume handlers.

C. Live Migration with Writable Shared Memory

Live migration enables a VM to be migrated in negligible downtime. A migration manager in Dom0 first transfers the memory of a VM to a destination host with the VM running. Then it repeats to transfer only dirty pages that are modified during the previous iteration. When the number of dirty pages to be transferred becomes small enough, the migration manager stops the VM and transfers the remaining dirty pages. Finally, it restarts a migrated VM at a destination host. To detect dirty pages during live migration, Xen provides the log dirty mode. In this mode, the VMM detects writes to memory pages and records dirty pages in a *log dirty bitmap*. At the end of each iteration of memory transfer, the migration manager obtains a log dirty bitmap from the

VMM. At the same time, a log dirty bitmap in the VMM is cleared to record newly modified pages. Then the migration manager transfers only dirty pages at the next iteration.

However, the log dirty mode cannot be used to detect writes to the shared memory between VMs. For DomU with the log dirty mode enabled, the VMM can detect writes only to the memory pages belonging to that DomU. In other words, even if DomR modifies memory pages shared with DomU, the VMM cannot recognize those pages as dirty. Therefore the migration manager for DomU cannot transfer the latest contents of memory pages shared with DomR when they are modified only by DomR. This results in the data loss of inputs and outputs for remote management after co-migration. For example, input data in an I/O ring may be lost when the keyboard backend driver in DomR has written data to the I/O ring but the frontend driver in DomU has not read them yet.

To prevent such data loss, D-MORE always considers DomU's memory pages shared with DomR in a writable manner as dirty. Thereby it is guaranteed that a migration manager transfers shared memory pages modified by DomR. Although such pages are always dirty, they are not transferred many times because a migration manager has a mechanism for preventing frequently modified pages from being transferred frequently. Specifically, a migration manager obtains a log dirty bitmap from the VMM not only at the end of each iteration but also just before the transfer of each memory block. If pages are modified between them, the migration manager does not transfer those pages. As a result, writable shared memory is transferred only once at the final stage of live migration. The modification to the shared memory after the last transfer is prevented by the synchronization in co-migration, which is described in the next section. In the current implementation, writable shared memory is always transferred even if it is not modified. However, it consists of only two pages for I/O rings.

For this purpose, we have extended the log dirty mode, as shown in Fig. 5. The VMM records the sharing status of DomU's memory in a bitmap prepared for each DomU, which is called a *may-write bitmap*. When DomU's memory page is mapped in a writable manner by DomR, the VMM sets the bit corresponding to its page frame number in the may-write bitmap. In contrast, when DomU's page is unmapped or is changed to read-only, the VMM clears the corresponding bit. When a migration manager issues a hypercall for obtaining a log dirty bitmap, the VMM merges the log dirty bitmap and the may-write bitmap.

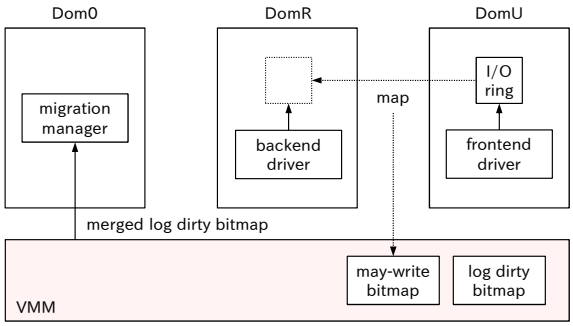

Figure 5. An extension to the log dirty mode.

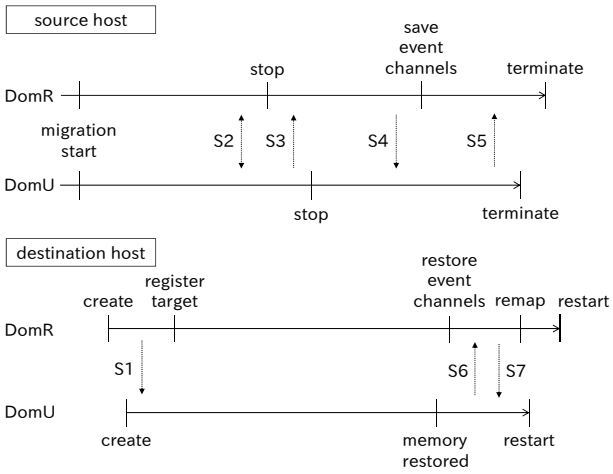

Figure 6. The synchronization in the co-migration of DomR and DomU.

D. Synchronization in Co-migration

For the continuity of out-of-band remote management, there are seven synchronization points in the co-migration of DomR and DomU, as illustrated in Fig. 6. After the migration managers for DomR and DomU start VM migration, they create new empty VMs at a destination host to store the transferred states. At synchronization point S_1 after DomR's creation, the migration manager for DomR waits for DomU's creation. Then it registers created DomU as a target of DomR.

Next, both migration managers at a source host transfer VM's memory and synchronize the entrances to the final stage of live migration at synchronization point S_2. This is two-way synchronization, which means that each migration manager waits until the other can enter the final stage. To reduce downtime, the migration managers perform extra iterations, while each examines the state of the other. The next synchronization point S_3 is at the beginning of the final stage. Before DomU's stop, the migration manager for DomU waits for DomR's stop. This guarantees that DomR does not modify the shared memory of DomU after DomU stops and the data in the shared memory has been transferred.

After the migration manager for DomR has transferred the remaining memory and the CPU state, it waits for DomU's stop at synchronization point S_4. Then it saves pairs of ports used for event channels. On the other hand, at synchronization point S_5 before DomU's termination, the migration manager for DomU waits until event channels

are saved. At a destination host, before restarting DomU, the migration manager for DomU waits at synchronization point S_6 until event channels are restored. These three synchronization points guarantee that the state of event channels is consistently saved and restored for VMs with the stopped state.

Finally, at synchronization point S_7, the migration manager for DomR waits until the whole memory of DomU is restored. After that, it can obtain a list of the frame numbers of memory pages allocated to DomU. This list is needed for remapping DomU's memory to DomR. Note that it is guaranteed that saving the memory-mapping state completes before DomU's termination because it is done before saving event channels.

V. Experiments

We conducted experiments to examine the continuity of out-of-band remote management across VM migration and to measure the performance of remote management and co-migration. For server machines hosting VMs, we used two PCs with one Intel Xeon E3-1270 3.40 GHz processor, 8 GB of memory, and gigabit Ethernet. We ran a modified version of Xen 4.3.2 and Linux 3.7.10 in Dom0, DomR, and DomU. By default, we allocated one virtual CPU and 128 MB of memory to DomR, one virtual CPU and 2 GB of memory to DomU, and eight virtual CPUs and the rest of the memory to Dom0. For a client machine, we used a PC with one Intel Xeon E5-1620 3.60 GHz processor, 8 GB of memory, and gigabit Ethernet. We ran a modified version of TightVNC Java Viewer 2.0.95 [5] on Windows 7. These PCs were connected with a gigabit Ethernet switch.

A. Data Loss on Co-migration

We examined that D-MORE could prevent data loss in out-of-band remote management on VM migration. We sent a key every 50 ms from a VNC client to a VNC server and monitored the data received by the keyboard backend driver during VM migration. We repeated this experiment 10 times and counted the number of lost keys. In the original Xen, the backend driver was removed by the migration of DomU. As a result, 1.4 keys were lost in the driver on average. In D-MORE, the backend driver was migrated with DomR on the co-migration of DomR and DomU. Across the co-migration, no keys were lost. During the migration of DomR, the number of TCP retransmission was 8.5 on average. These results showed that D-MORE prevented data loss successfully.

Next, we examined that D-MORE could prevent the loss of data written in shared memory. To confirm this, we disabled our extension to the log dirty mode. When the keyboard backend driver in DomR writes input data to the I/O ring in shared memory, the memory page may not be correctly transferred without our extension. We performed co-migration 10 times, as in the above experiment, and examined whether data was lost or not. By default, we could not observe any data loss. However, when we added a delay of 200 ms to DomU's read of event channels, keys were lost. This means that data in I/O rings can be lost during co-migration due to VM scheduling in the VMM and process scheduling in VMs.

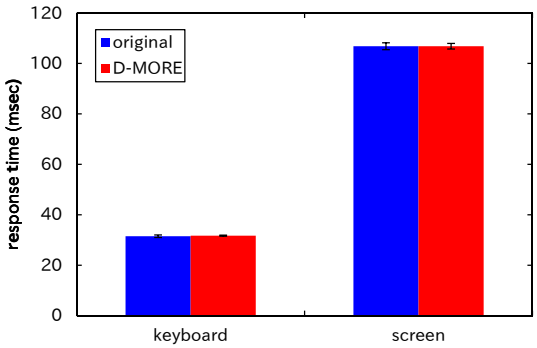

Figure 7. The response time of a keyboard input and a full-screen update.

B. Overhead of DomR

To examine the overhead of out-of-band remote management via DomR, we first measured the response time of a keyboard input. The response time was the time from when a VNC client sent a keyboard event until it received a screen update caused by its remote echo. For comparison, we measured the response time when a VNC server ran in Dom0 as usual. The left bars in Fig. 7 show the averages and standard deviations in the original Xen and D-MORE. From these results, the increase of the response time was negligible.

Next, we examined the response time of a full-screen update when we updated the full screen (800×600) of DomU. For a full-screen update, we ran a screen saver that redrew the full screen frequently in DomU. Compared with a keyboard input, a full-screen update requires that DomR handles a larger amount of data. We measured the response time when we used DomR or Dom0 for running a VNC server. As shown in Fig. 7, the response time was longer than that of a keyboard input. However, the difference of the response times between the original Xen and D-MORE was negligible.

C. Co-migration Time

We measured the time needed for co-migration of DomR and DomU. We changed the memory size of DomU from 256 MB to 2 GB. To examine the impact of out-of-band remote management during co-migration, we measured the co-migration time both when a VNC client did not connect to a VNC server and when it sent a keys every 50 ms. This key input rate is too fast for human, but it is possible when we copy and paste text. For comparison, we migrated two independent DomUs in parallel without synchronization,

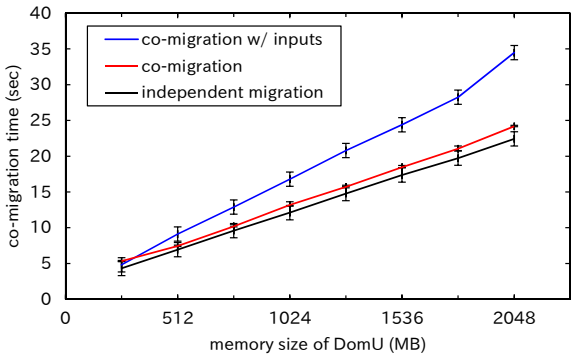

Figure 8. The co-migration time for various memory sizes of DomU.

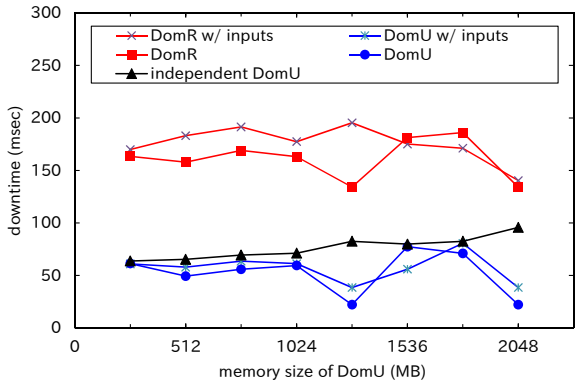

Figure 9. The downtime of DomR and DomU for various memory sizes of DomU.

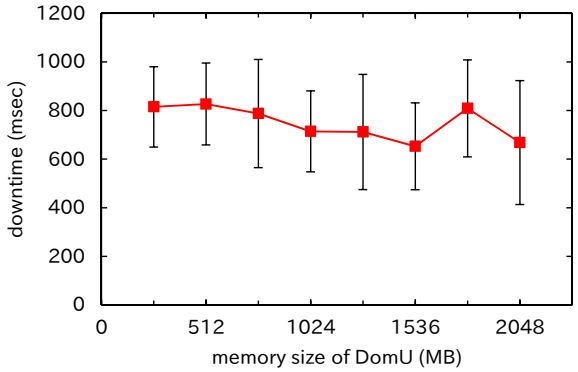

Figure 10. The user-perceived downtime for various memory sizes of DomU.

using the original Xen. We fixed the memory size of one DomU to 128 MB and changed that of the other DomU. The co-migration time we measured was from when we started co-migration until the migration of both VMs completed.

Fig. 8 shows the averages and standard deviations when we measured the co-migration time 10 times for each memory size. The co-migration time was proportional to the memory size of DomU. This is because the total migration time of two VMs depends on the total size of the memory to be transferred. Compared with independent migration of two DomUs, the co-migration time in D-MORE increased by only 1.7 seconds at maximum. When we performed remote management during co-migration, the co-migration time increased largely. The reason is that a larger amount of memory became dirty in a VNC server, virtual devices, and DomU. Therefore it took time to enter the final stage of VM migration.

D. Downtime

First, we measured the downtime of DomR and DomU during co-migration. The setup of this experiment was the same as the above. The downtime we measured was the time in which a VM was not running at either source or destination host. We conducted this experiment 10 times. The average downtime was shown in Fig. 9. Since the standard deviations of all of these downtimes were quite large (between 6 and 65 ms), we omitted error bars for visibility. In the independent migration of DomUs, the downtime was proportional to the memory size of DomU. However, those of DomR and DomU in D-MORE did not clearly depend on the memory size.

For DomU, the downtime in D-MORE was shorter than that in the independent migration of DomUs. This means that the impact of co-migration on DomU is small. For DomR, on the other hand, the downtime was much higher than DomU because DomR stopped before DomU by the synchronization in co-migration. In addition, DomR was usually restarted after DomU due to restoring the memory-mapping state just before its restart. When we performed out-of-band remote management during co-migration, the downtime of DomR often increased further, but that of DomU was almost not affected. The reason is probably that a larger amount of memory was modified in DomR by remote management.

Next, we measured the user-perceived downtime at a VNC client. We sent a key every 50 ms from a VNC client and measured the response time as in the above experiment. Then we considered the longest response time at the final stage of co-migration as the user-perceived downtime. In this experiment, we allocated memory to DomR and DomU as in the above experiment. Fig. 10 shows the averages and standard deviations when we measured the downtime 10 times. The average user-perceived downtime was inversely proportional to the memory size of DomU and the maximum was 827 ms. We believe that this downtime is acceptable for the purpose of remote management although users would perceive that their VMs are migrated.

E. Performance Degradation during Co-migration

The co-migration of DomR and DomU can affect the performance of out-of-band remote management. To examine the impact on the response time, we performed the co-migration and measured the changes of the response time during it. In this experiment, we used DomU with 2 GB of

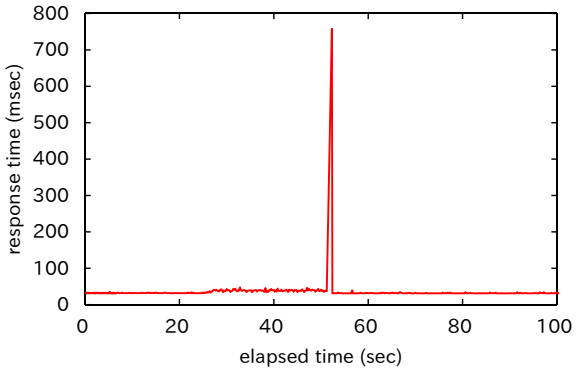

Figure 11. The changes of the response time during co-migration.

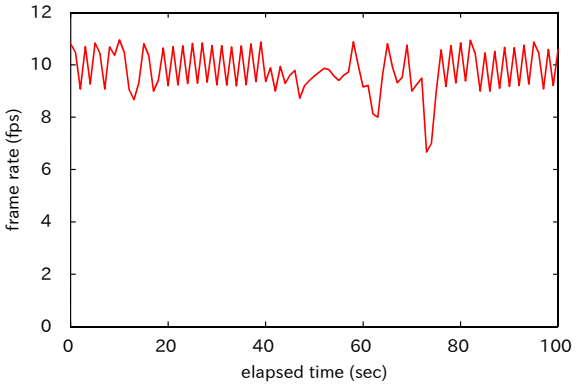

Figure 12. The changes of the frame rate during co-migration.

memory and sent a key every 50 ms. Fig. 11 shows the result. After we started co-migration, the response time increased by 5.4 ms on average. This performance degradation lasted for 30 seconds. At the final stage of the co-migration, the response time became 744 ms.

To examine the impact on the frame rate of screen updates by a screen saver in DomU, we measured the changes of the frame rate during co-migration. We calculated the average frame rate from the response times of a full-screen update. As shown in Fig. 12, the frame rate decreased by about 0.4 frame per second (fps) on average after the co-migration was started. The degradation of the frame rate lasted for 30 seconds and the frame rate was 6.7 fps at the final stage of co-migration.

VI. Related Work

To continue out-of-band remote management across VM migration, there are two approaches different from D-MORE. The first approach is using a VNC proxy. To manage a user VM, a VNC client accesses a VNC server in the management VM via a VNC proxy. When a user VM is migrated, a VNC proxy can switch the connection to a VNC server from a source to a destination host. For example, a console proxy in CloudStack [6] can support this functionality. Even if the connection between VNC proxy and server is switched, a user is not aware of VM migration because the connection between VNC client and proxy is preserved. However, all the pending data in a VNC server and virtual devices is lost at a source host.

The second approach is using SPICE [7], which is remote management software developed for KVM. SPICE supports VM migration at the protocol level. When a user VM is migrated, a SPICE server in the management VM notifies a SPICE client of the destination host. Then a SPICE client switches the connection to a SPICE server at the destination host. At that time, a user using a SPICE client is not aware that his VM is migrated. In addition, no data is lost when seamless migration is enabled. One disadvantage is that VM management depends on specific remote management software. D-MORE enables users to use any remote management software such as SSH. Another disadvantage is that the network address of a SPICE server changes every VM migration. This can make it difficult to configure client-side firewalls.

Special-purpose VMs proposed so far have similarities to DomR although most of them are not migratable. Stub domains [8], [9] in Xen and Qemu VM in Xoar [10] can run QEMU to provide virtual devices to DomU. They support fully virtualized DomU, while DomR supports para-virtualized DomU. Therefore, stub domains and Qemu VM do not need a privilege for intercepting event channels, which are used for para-virtualization. On the other hand, driver domains [11] in Xen can run backend drivers in the kernel for para-virtualized DomU. Since driver domains are unprivileged, they use grant tables to share the memory pages of DomU. Grant tables allow any VMs to map only memory pages permitted by other VMs. A driver domain can establish event channels with DomU by making DomU explicitly specify that driver domain as backend. From these reasons, only several backend drivers such as a network driver can run in driver domains.

A guard VM in VMCoupler [3] is a privileged and migratable VM, which is used as the base of DomR. It runs intrusion detection systems to monitor the inside of the target VM. It can map the memory of its target VM, access virtual disks of the target, and capture packets from/to the target. However, a guard VM does not have a privilege for intercepting event channels because it does not need to interact with the target VM. Service domains (SDs) in a self-service cloud computing platform [12] have privileges for accessing target VMs. SDs can monitor the memory of target VMs and intercept disk access between frontend and backend drivers. DomB [13] is used to boot DomU more securely. Instead of Dom0, it can load a kernel image into the DomU's memory and set up DomU. These VMs are not designed for running virtual devices for DomU.

Similar to D-MORE, VMCoupler [3] enables synchronized co-migration of a guard VM and its target VM. It synchronizes two migration processes mainly for secure

monitoring, while D-MORE does mainly for the continuity of out-of-band remote management. Therefore the synchronization in co-migration is largely different between VMCoupler and D-MORE. VMCoupler performs coarse-grained synchronization using only the VM states at only four points. D-MORE performs fine-grained synchronization using various states in migration managers at seven points. In addition, co-migration in VMCoupler does not consider writable shared memory because memory monitoring needs only read-only mapping.

For concurrent migration of multiple co-located VMs, live gang migration [14] has been proposed. It transfers memory contents that are identical across VMs only once to reduce the migration overhead. It tracks identical memory contents across VMs and performs memory de-duplication for all the migrated VMs. It also applies differential compression to nearly identical memory pages. Unlike D-MORE, live gang migration does not synchronize between the migration processes of multiple VMs. This approach can be incorporated into D-MORE to reduce the co-migration time.

For virtualization software different from Xen, KVM [15] runs QEMU including a VNC server and virtual devices in the host operating system. Since a VM also runs on QEMU in KVM, it might be possible to continue out-of-band remote management by migrating a QEMU process with the state of a VM. However, process migration [16] is not easier than VM migration. VMware vSphere Hypervisor [17] runs a VNC server and virtual devices in the VMM and enables out-of-band remote management without the management VM. However, when a user VM is migrated, the connection to a VNC server is terminated.

VII. Conclusion

In this paper, we proposed D-MORE for continuing out-of-band remote management across VM migration. D-MORE provides a privileged and migratable VM called DomR to run a VNC server and virtual devices, which are necessary for remote management of a target VM. D-MORE synchronously co-migrates DomR with its target VM and transparently maintains the connections between a VNC client, DomR, and its target VM at the network and VMM levels. During VM migration, D-MORE prevents the loss of inputs and outputs for remote management. We have implemented D-MORE in Xen and confirmed that the remote management of a target VM via DomR was not discontinued on the co-migration. Our experimental results showed that all the pending data was not lost and that the overhead of D-MORE was acceptable.

One of our future work is to support other remote management software in DomR. We could use SSH easily by disabling the resume operation in the console frontend driver. To support others, we need to port them so that they directly access virtual devices. Another future work is to support fully virtualized VMs in D-MORE. DomR could easily run virtual devices for them like stub domains in Xen, but we have to develop a mechanism for migrating them. In addition, we are planning to reduce resource consumption of DomR. Running Xen's Mini OS can decrease the memory footprint of DomR. Using network address and port translation (NAPT) enables DomR to use a private IP address.

Acknowledgment

This research was supported in part by JSPS KAKENHI Grant Number 25330086.

References

[1] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the Art of Virtualization," in *Proc. Symp. Operating Systems Principles*, 2003, pp. 164–177.

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

[3] K. Kourai and H. Utsunomiya, "Synchronized Co-migration of Virtual Machines for IDS Offloading in Clouds," in *Proc. Int. Conf. Cloud Computing Technology and Science*, 2013, pp. 120–129.

[4] F. Bellard, "QEMU," http://qemu.org/.

[5] TightVNC Group, "TightVNC," http://www.tightvnc.com/.

[6] Apache Software Foundation, "Apache CloudStack: Open Source Cloud Computing," http://cloudstack.apache.org/.

[7] Red Hat, "Spice," http://www.spice-space.org/.

[8] J. Nakajima and D. Stekloff, "Improving HVM Domain Isolation and Performance," in *Xen Summit September 2006*, 2006.

[9] S. Thibault, "Stub Domains," in *Xen Summit Boston 2008*, 2008.

[10] P. Colp, M. Nanavati, J. Zhu, W. Aiello, G. Coker, T. Deegan, P. Loscocco, and A. Warfield, "Breaking Up is Hard to Do: Security and Functionality in a Commodity Hypervisor," in *Proc. Symp. Operating Systems Principles*, 2011, pp. 189–202.

[11] K. Fraser, S. Hand, R. Neugebauer, I. Pratt, A. Warfield, and M. Williamson, "Safe Hardware Access with the Xen Virtual Machine Monitor," in *Proc. Workshop on Operating System and Architectural Support for the on demand IT InfraStructure*, 2004.

[12] S. Butt, H. A. Lagar-Cavilla, A. Srivastava, and V. Ganapathy, "Self-service Cloud Computing," in *Proc. Conf. Computer and Communications Security*, 2012, pp. 253–264.

[13] D. G. Murray, G. Milos, and S. Hand, "Improving Xen Security through Disaggregation," in *Proc. Int. Conf. Virtual Execution Environments*, 2008, pp. 151–160.

[14] U. Deshpande, X. Wang, and K. Gopalan, "Live Gang Migration of Virtual Machines," in *Proc. Int. Symp. High Performance Distributed Computing*, 2011, pp. 135–146.

[15] A. Kivity and M. Tosatti, "Kernel Based Virtual Machine," http://www.linux-kvm.org/, 2007.

[16] D. S. Milojíčić, F. Douglis, Y. Paindaveine, R. Wheeler, and S. Zhou, "Process Migration," *ACM Comput. Surv.*, vol. 32, no. 3, pp. 241–299, 2000.

[17] VMware Inc., "VMware vSphere Hypervisor," http://www.vmware.com/.