Journal of Information Processing Vol.24 No.4 1–11 (July 2016)

[DOI: 10.2197/ipsjjip.24.1]

Regular Paper

Virtual Machine Co-migration for Out-of-band Remote Management

Sho Kawahara[1,a) Kenichi Kourai[1,b)

Received: March 3, 2015, Accepted: March 4, 2016

Abstract: In Infrastructure-as-a-Service (IaaS) clouds, users manage the systems in virtual machines (VMs) called *user VMs* through remote management systems (RMSes). To allow users to manage their VMs during failures inside the VMs, IaaS usually provides *out-of-band* remote management. This management is performed indirectly via an RMS server in a privileged VM called the *management VM*. However, it is discontinued when a user VM is migrated. This is because an RMS server in the management VM at the source host is terminated on VM migration. Even worse, pending data is lost between an RMS 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. During VM migration, it synchronously co-migrates DomR and its target VM and transparently maintains the connections between an RMS 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 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

1. 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 systems (RMSes) such as Virtual Network Computing (VNC) and Secure Shell (SSH). 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, an RMS server is run in the management VM, not in a user VM, and directly interacts with the virtual devices of 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.

However, out-of-band remote management is discontinued when a user VM is migrated from the source to the destination host. During VM migration, virtual devices in the management VM at the source host are removed. Therefore an RMS server interacting with the virtual devices is also terminated. To restart remote management, a user has to identify the reason for the disconnection, look for a destination host, and reconnect to a new RMS server at the host. This is a laborious task for a user and becomes downtime in remote management. Even worse, inputs and outputs for remote management can be lost by VM migration. Pending data in an RMS server and virtual devices is abandoned when an RMS server and virtual devices are terminated. Since the network connection between RMS client and server is also terminated, in-flight packets containing inputs and outputs are lost and are not retransmitted. Therefore, a user has to recover lost inputs by himself after migration. This causes inconvenience to a user and becomes hidden downtime in remote management. If an RMS redirects local devices such as USB storage to a remote user VM, data loss between them is critical.

In this paper, we propose *D-MORE* [1], which is a system for continuing out-of-band remote management across VM migration. Our idea involves running an RMS 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 an RMS client, DomR, and its target VM at the levels of the virtual machine monitor (VMM) and the network. The VMM is a software layer underlying VMs and manages the interaction between VMs. In addition, D-MORE can prevent data loss of inputs and outputs for remote management. Pending data in a RMS server and virtual devices is preserved in DomR because it is migrated as a part of DomR. In-flight network packets for delivering inputs and outputs are retransmitted by TCP even if they are dropped during VM migration.

We have implemented D-MORE in Xen 4.3.2 [2]. 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 intercept and establish event channels with a user

[1] Kyushu Institute of Technology, 680-4 Kawazu, Iizuka, Fukuoka 820-8502, Japan

a) kawasho@ksl.ci.kyutech.ac.jp

b) kourai@ci.kyutech.ac.jp

VM. During the co-migration of DomR and its target VM, D-MORE restores the states of memory mapping and event channels to the destination host. To safely perform this state restoration at the appropriate time and to prevent loss of data 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 performance of remote management degraded during co-migration, but the overhead of D-MORE was small during a normal run. The downtime during co-migration was much less than that in traditional manual reconnection and was deemed acceptable.

This paper is an extended version of our previous paper [1]. In this paper, we support not only VNC but also SSH as RMSes to show the generality of D-MORE. Then we also show the performance of remote management using SSH in D-MORE. In addition, we examine the overhead of D-MORE in further detail.

The organization of this paper is as follows. Section 2 describes issues in out-of-band remote management using the management VM and related work. Section 3 proposes D-MORE for continuing out-of-band remote management across VM migration. Section 4 explains the implementation details in Xen and Section 5 shows our experimental results. Section 6 concludes this paper.

2. Motivation

2.1 Out-of-band Remote Management

To manage user VMs in IaaS clouds, a user usually connects an RMS client on the user's host to an RMS 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. Examples of RMSes are VNC for graphical user interface (GUI) management and SSH for character user interface (CUI) management. However, this in-band remote management is not powerful enough to manage a user VM at all times. If a user has just failed the configurations of the network or firewall inside 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 an RMS server in a user VM is not running normally, an RMS client cannot access the VM. For example, an RMS 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**, an RMS 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 and a virtual video card. An RMS 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 an RMS 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. Even when the system in a user VM crashes, the user may check kernel messages through the virtual video card.

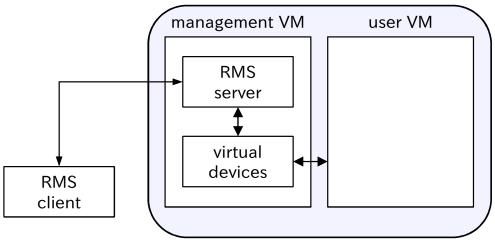


Fig. 1 Out-of-band remote management of a user VM.

2.2 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 [3] 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. It has been reported that VM migration for such dynamic consolidation was done for 25% of VMs every four hours on average in the private cloud of a large airline company [4]. This means that each VM was migrated every 16 hours on average. In in-band remote management, it is possible to continue VM management across the migration of a user VM. An RMS server in a user VM is migrated as a part of the VM and the network connection between RMS client and server is preserved. At that time, a user using an RMS client is not aware that his VM is migrated.

However, in out-of-band remote management, VM management is discontinued on VM migration, as shown in **Fig. 2**. When a user VM has been migrated, its virtual devices in the management VM at the source host are removed. The migrated VM uses new virtual devices created in the management VM at the destination host. At the same time, an RMS server in the management VM at the source host is terminated because it loses the access to the removed virtual devices. As a result, an RMS client is disconnected from the RMS server. To restart remote management, a big burden is imposed on the user. First, a user has to identify the reason why an RMS 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 an RMS server in the management VM at that host. At that time, he has to enter the password for the connection again. This can lower the efficiency of the VM management.

Worse than that, inputs and outputs for remote management can be lost when a user VM is migrated. If input data has been sent from an RMS client but has not yet been received by an RMS

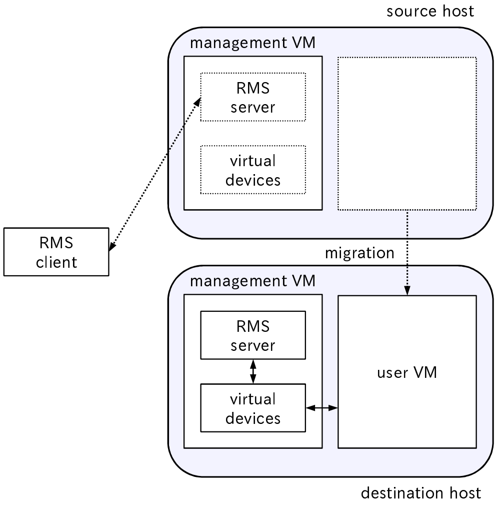


Fig. 2 VM migration during out-of-band remote management.

server, in-flight network packets containing that data are dropped. Since the network connection between RMS client and server is terminated, dropped packets are not retransmitted at the network level. Usually, an RMS client does not have a mechanism for data retransmission at the application level. If input data received by an RMS server has not yet been sent to a virtual device, it is lost when the RMS server terminates. If a virtual device has received input data but does not send it to a user VM, the data is also lost by the removal of the virtual device. Similarly, output data can be lost in virtual devices, an RMS server, or the network.

Such data loss causes inconvenience to users and is critical in some cases. When keyboard inputs are lost, a user has to type them again to recover data loss. This is troublesome, for example, when he inputs large text in a copy-and-paste manner. He has to identify which part of the text has been pasted correctly and then paste only the lost part again. When console outputs are lost, such manual recovery is impossible because output data is generated by a user VM. Unless a user records all the outputs in a log file or using the GNU screen utility, he can never receive lost information. When an RMS redirects local devices such as USB storage to a remote user VM, data loss between them is critical. If data sent from a user VM to local USB storage is lost, files in the storage can be corrupted. Conversely, if data sent from local USB storage to a user VM is lost, read errors occur in a user VM.

Furthermore, security on a client side can become lower because the network address of an RMS server is changed every VM migration. Nowadays, for security, client-side firewalls are often configured so that they permit outbound packets only to minimum hosts and network ports. However, the IP address of an RMS server is changed from that of the management VM at the source host to that of the destination host. Likewise, the port number assigned to an RMS server is changed. Since the management VM runs many RMS servers for the corresponding user VMs, it assigns a port number in order. To support VM migration in out-of-band remote management, client-side firewalls have to permit accesses to all the IP addresses used in all the management VMs in a cloud and all the port numbers that can be used by an RMS server. If attackers intrude into client hosts, they can mount stepping-stone attacks against more hosts in the cloud.

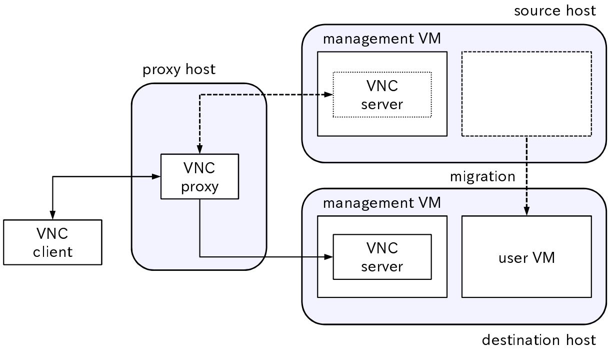


Fig. 3 Switching VNC servers by a VNC proxy.

2.3 Previous Approaches

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, as illustrated in **Fig. 3**. To manage a user VM, a VNC client accesses the VNC server in the management VM via a VNC proxy. When a user VM is migrated, a VNC proxy can switch the connection of the VNC server from the source to the destination host. For example, a console proxy in CloudStack [5] can support this functionality. Even if the connection between the VNC proxy and the server is switched, the connection between the VNC client and the proxy is preserved. However, a user cannot access his VM until the VNC proxy detects the VM migration and then reconnects to the VNC server at the destination host. This reconnection can take a certain time. In addition, all of the pending data in the VNC server and virtual devices are lost at the source host. Since the connection is preserved but some data is lost, it is difficult for the user to notice any data loss. Another disadvantage is that an RMS proxy has to be developed for each RMS. D-MORE does not need the development of RMS proxies.

The second approach is using SPICE [6], which is an RMS developed for KVM and allows USB redirection. 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, as shown in **Fig. 4**. 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 a specific RMS. D-MORE enables users to use any RMSes such as VNC and SSH. Another disadvantage is that the network address of a SPICE server changes every VM migration. In D-MORE, the address of an RMS server does not change because the server is migrated with DomR.

2.4 Related Work

Special-purpose VMs proposed so far have similarities to DomR although they are not migratable except for a guard VM in VMCoupler [7], [8]. Stub domains [9], [10] in Xen and Qemu VM in Xoar [11] can run QEMU to provide virtual devices to a user VM. They support fully virtualized user VMs, while DomR supports para-virtualized user VMs. Therefore, stub domains and Qemu VM do not need a capability for intercepting event

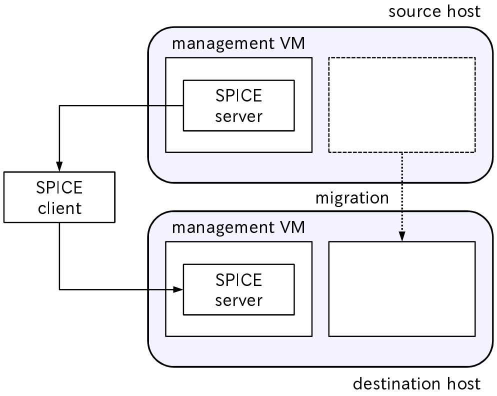


Fig. 4 VM migration support in SPICE.

channels, which are used as para-virtualized interrupt channels. On the other hand, driver domains [12] in Xen can run backend drivers in the kernel for para-virtualized VMs. Since driver domains are unprivileged, they use grant tables to share the memory pages of user VMs. Grant tables allow any VMs to map only memory pages permitted by other VMs. A driver domain can establish event channels with a user VM by making it explicitly specify that driver domain as the backend. Currently, only several backend drivers such as a network driver can run in driver domains.

A guard VM in VMCoupler [7], [8] 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 capability 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 [13] have a capability for accessing target VMs. SDs can monitor the memory of target VMs and intercept disk access between frontend and backend drivers. DomB [14] is used to boot user VMs more securely. Instead of Dom0, it can load a kernel image into the user VM's memory and set up the user VM. These VMs are not designed for running virtual devices for user VMs.

Similar to D-MORE, VMCoupler enables synchronized co-migration of a guard VM and its target VM. It synchronizes two migration processes mainly for secure monitoring, while D-MORE maintains 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 four points. D-MORE performs fine-grained synchronization using various states of migration 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 [15] 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 co-migration time.

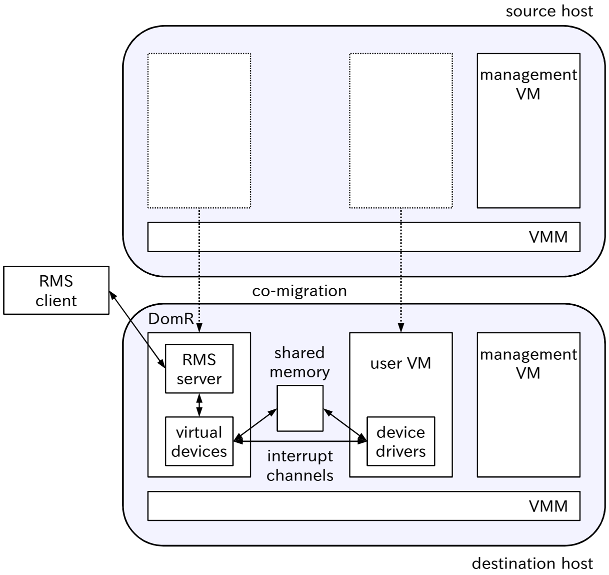


Fig. 5 The system architecture of D-MORE.

For virtualization software different from Xen, KVM [16] runs QEMU including a VNC server and virtual devices on 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 [17] is not easier than VM migration. VMware vSphere Hypervisor [18] 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.

3. D-MORE

In this paper, we propose *D-MORE*, which can continue out-of-band remote management after the migration of user VMs. The system architecture of D-MORE is shown in **Fig. 5**. D-MORE provides a privileged and migratable VM called *DomR* for remote management of a user VM. DomR runs only an RMS server and virtual devices for its target VM. Therefore DomR does not require many resources. For example, one virtual CPU and 128 MB of memory are sufficient. An RMS client connects to an RMS 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. During migration, D-MORE transparently maintains all the connections between an RMS 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 an RMS client and an RMS server in DomR at the network level, similar to typical VM migration.

DomR has capabilities 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 capability 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 an RMS server in DomR writes keyboard input received from an RMS 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 capability for establishing interrupt channels with its target VM. Through 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 to its target VM. During VM migration, DomR is disconnected from the target VM once because all connections between VMs are lost except for network connections. Specifically, D-MORE re-establishes the shared memory and interrupt channels between DomR and its target VM. To do this, D-MORE saves the states of memory sharing and interrupt channels at the source host. Then it transfers those states with the other states of the VMs to the destination host. At the 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 to 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 transfer of the latest data in shared memory to the 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 an RMS server and virtual devices is preserved. Since an RMS server and virtual devices are migrated as part of DomR, they can continue to process such pending data at the 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 an RMS client to an RMS server are retransmitted by TCP although they may be dropped temporarily by migrating DomR. D-MORE preserves the TCP connection between the RMS client and server by running an RMS server in DomR. This can address the users' inconvenience of manually recovering input data after VM migration and eliminate hidden downtime in remote management. It is not necessary that users themselves prepare data loss. Furthermore, data corruption during VM migration can be prevented even when users redirect local USB devices.

D-MORE can support any RMS servers that can run in an out-of-band manner in DomR. This is because DomR provides a virtual keyboard, a virtual mouse, a virtual video card, and a virtual serial console as virtual devices. For GUI management, RMS servers such as a VNC server can basically run with a virtual keyboard, a virtual mouse, and a virtual video card. For CUI management, RMS servers such as an SSH server can run with only a virtual serial console. Since D-MORE does not depend on the protocols used between RMS clients and servers, it can perform co-migration even if any RMS servers are used in out-of-band remote management. At that time, any data is not lost without any support of RMSes.

In addition, no network address of an RMS server is changed when a user VM is migrated. Since an RMS server runs inside DomR, it can continue to use the same IP address as DomR. The IP address of DomR is not changed even after DomR has been migrated. Likewise, an RMS server can continue to use the same port number as the one before migration. Therefore client-side firewalls need only one rule for permitting access to an RMS server in DomR. Even if a client host is compromised, attackers can access only the DomR and not any management VMs in the cloud. Instead of these advantages, DomR is created per user VM and consumes one IP address. Using an extra global IP address is costly for clouds. To solve this problem, D-MORE can assign a private IP address to DomR and perform network address and port translation (NAPT). For example, when a user accesses the global IP address of a user VM and port 5910, the access is translated to the private IP address of DomR and port 5900 for an RMS server.

4. Implementation

We have implemented D-MORE in Xen 4.3.2 [2]. In Xen, the VMM runs on top of the 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 [7], [8]. DomR runs para-virtualized Linux for running virtual devices. In the current implementation, D-MORE supports DomU running para-virtualized Linux. It targets the x86-64 architecture and supports VNC and SSH as RMSes.

In this section, we mainly focus on the differences from our previous work [1].

4.1 Connection between DomR and DomU

When DomU is booted, D-MORE binds the DomU to new DomR. To establish shared memory with the target DomU, DomR maps memory pages of that DomU. Thereafter, both DomR and DomU can access the same memory pages. In addition, DomR can intercept and 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 The. interrupts implementation details are explained in Refs. [1], [7], [8].

4.2 Virtual Devices in DomR

DomR runs QEMU [19] that is customized for Xen to provide virtual devices and a VNC server necessary for out-of-band remote management. In traditional out-of-band remote manage-

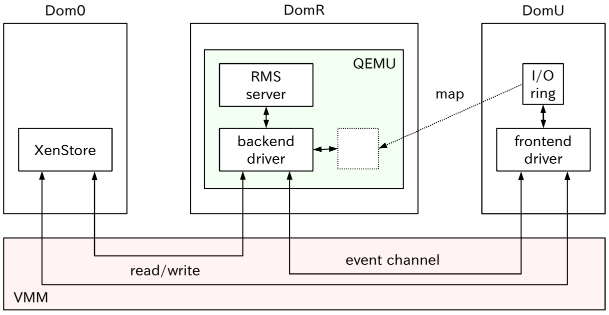


Fig. 6 The split-driver model with DomR.

ment, QEMU runs in Dom0. For para-virtualized DomU, the split-driver model is used in Xen, as illustrated in **Fig. 6**. Virtual devices are implemented as backend drivers 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.

For the support of remote management using VNC, see our previous paper [1].

4.2.1 SSH Support

To support remote management using SSH, DomR provides a virtual serial console for both inputs and outputs. The initialization is similar to that of the other para-virtual devices. One of the differences is that the console frontend and backend drivers use console rings for communication, instead of I/O rings. An I/O ring stores device-specific events, whereas a console ring simply stores input and output characters. When the console frontend driver connects to its backend driver, the backend driver maps the page for console rings and binds an event channel. Unlike the other para-virtual devices, the page for console rings and an unbound event channel are allocated by a migration manager in Dom0, instead of the frontend driver in DomU, just after a VM is created.

Unlike a VNC server, an SSH server in DomR is connected to the console backend driver in QEMU via the xenconsole command. This is because an SSH server runs outside QEMU. After a user logs in DomR using SSH, he executes xenconsole, specifying a target VM. When an SSH server receives a console input from an SSH client, it sends the input to the standard input of xenconsole. Then xenconsole sends it to the backend driver via a pseudo-terminal. The backend driver writes the input to a console ring for inputs and sends an event to the frontend driver. On the other hand, the frontend driver writes a console output to a console ring for outputs and sends an event to the backend driver. The output is sent to the pseudo-terminal of xenconsole, an SSH server, and then an SSH client.

4.2.2 Support for Other RMSes

Using the above SSH support, D-MORE can support any types of the RMS servers for CUI such as telnet and rsh servers although most of them are not recommended for security reasons. Since such RMS servers and a virtual serial console are separated completely, D-MORE can run arbitrary RMS servers for CUI in DomR easily. For GUI remote management, D-MORE can also support arbitrary RMS servers, but it currently supports only a VNC server. This is because RMS servers for GUI are not well separated from virtual devices such as a virtual keyboard/mouse and a virtual framebuffer in Xen. Therefore, RMS servers have to be embedded into QEMU. We believe that D-MORE can easily support SPICE, which is already embedded into QEMU. However, D-MORE needs additional support for virtual devices because SPICE supports the redirection of USB devices. We think that D-MORE can support other RMS servers for GUI by embedding them into QEMU or exporting virtual devices to the outside of QEMU.

4.3 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 the destination host. D-MORE also restores the state of event channels. To reuse re-established event channels in the operating system kernels of DomR and DomU, we have modified the resume operation for virtual interrupts. For further details on the implementation, refer to our previous work [1], [7], [8].

Also, we have disabled the resume operations of frontend drivers in DomU. This is necessary for preventing re-initialization of I/O and console rings, re-allocation of unbound event channels, and reconnection to their backend drivers by themselves. For the console frontend driver, the resume handler almost does nothing. Instead, the migration manager for DomU re-allocates an unbound event channel for it during migration. Therefore, our migration manager does not re-allocate an event channel for the virtual serial console at resume time and, instead, enables D-MORE to re-establish the event channel for it. For the keyboard/mouse and framebuffer frontend drivers, see our previous paper [1].

In addition, we have modified the xenconsole command so that it can continue to connect to the virtual serial console for migrated DomU. Since xenconsole is connected to DomU by specifying its identifier, it is disconnected when the identifier of DomU is changed by migration. We disabled the mechanism for detecting the change of DomU's identifier in xenconsole.

Another possible approach is to reconnect frontend drivers in DomU and backend drivers in DomR but re-initialize no I/O (or console) rings after co-migration. This driver-level reconnection seems to be done more easily because it is not necessary to restore the states of memory mapping and event channels at the VMM level. However, it is not the case. If backend drivers access I/O rings or event channels before the driver-level reconnection, they would fail such operations. Since backend drivers run as user-level drivers in the QEMU process, such failures can occur. Unlike kernel-level drivers in DomU, it is difficult to guarantee that user-level drivers execute resume operations just after DomR restarts. If DomU is migrated while user-level drivers are accessing I/O rings or event channels, these drivers continue that access after the migration. To prevent such a situation, QEMU in DomR needs a mechanism for suspending and resuming user-level backend drivers.

4.4 Live Migration with Writable Shared Memory

In live migration, a migration manager in Dom0 first transfers the memory of a VM to the 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 the 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*. According to the bitmap, 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 for remote management after co-migration.

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 multiple times because a migration manager has a mechanism for transferring frequently modified pages only at the final stage of live migration. For this purpose, we have extended the log dirty mode so that a bit is set in a log dirty bitmap when the corresponding memory page is mapped by DomR. For the implementation details, refer to our previous paper [1].

4.5 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. 7**. After the migration managers for DomR and DomU start VM migration, they create new empty VMs at the destination host to store the transferred states. At synchronization point S_1, after DomR is created, the migration manager for DomR waits for DomU to be created. Then it registers the created DomU as a target of DomR.

Next, both migration managers at the source host transfer the 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 is stopped, the migration manager for DomU waits for DomR to be stopped. 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 to be stopped at synchronization point S_4. Then it saves the event channels. On the other hand, at synchronization point S_5, before DomU is terminated, the migration manager for DomU waits until event channels are saved. At the 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.

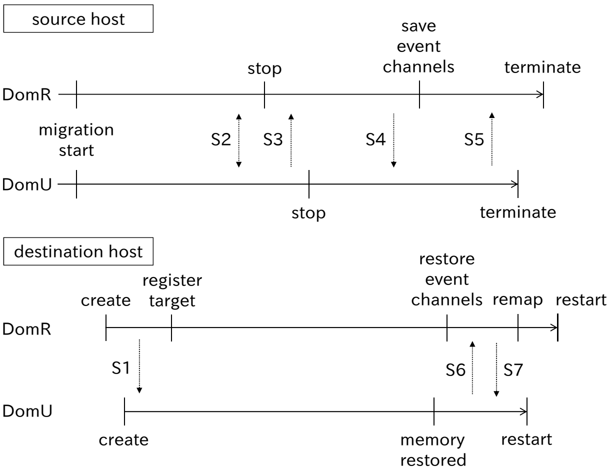


Fig. 7 The synchronization in the co-migration of DomR and DomU.

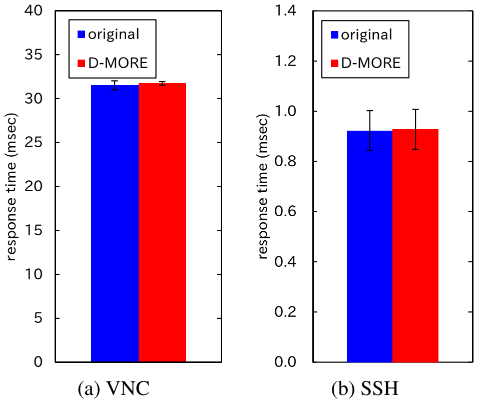
Finally, at synchronization point S_7, the migration manager for DomR waits until the whole memory of DomU is restored. After that, it can obtain a list of memory pages allocated to DomU. This list is needed for remapping DomU's memory to DomR. Note that it is guaranteed that saving the memory-mapping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

5. 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 TightVNC Java Viewer 2.0.95 [20] on Windows 7 and an OpenSSH 6.0 client [21] on Linux 3.16.0. These PCs were connected with a gigabit Ethernet switch.

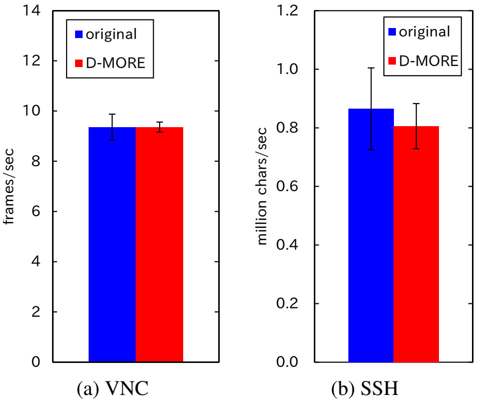
5.1 Data Loss on Co-migration

We investigated whether 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



(a) VNC (b) SSH

Fig. 8 The response time of remote management.



(a) VNC (b) SSH

Fig. 9 The throughput of remote management.

was removed by the migration of DomU. As a result, 1.4 keys were lost by co-migration on average. In D-MORE, the backend driver was migrated with DomR when DomR and DomU were co-migrated. Across the co-migration, no keys were lost. During the migration of DomR, the number of TCP retransmissions was 8.5 on average. These results showed that D-MORE prevented data loss successfully.

Next, we examined whether D-MORE could prevent the loss of data written in shared memory. 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 to the log dirty mode. 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 only when we disabled our extension. 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.

5.2 Performance in D-MORE

To examine the performance of out-of-band remote management via DomR, we first measured the response time. In VNC, the response time was the time from when a VNC client sent a keyboard event until it received a screen update by displaying a corresponding character. For comparison, we measured the response time when a VNC server ran in Dom0. **Figure 8**(a) shows the results in the original Xen and D-MORE for VNC. The increase in response time was negligible. Similarly, we measured the time from when an SSH client sent a console input until it received a console output caused by its remote echo. Figure 8(b) is the result for SSH. The standard deviation was larger than that of VNC, but the response time was almost the same.

Next, we examined the throughput of remote management. For VNC, we measured the frame rate of a full-screen update of DomU at a VNC client. For a full-screen update, we ran a screen saver that redrew the full screen (800×600) as fast as possible in DomU. As shown in **Fig. 9**(a), the difference in throughput between the original Xen and D-MORE was negligible. For SSH, we ran a benchmark that wrote characters to a serial console as fast as possible in DomU. Then we measured the output rate of characters at an SSH client. Figure 9(b) shows that D-MORE suffered from slight overhead. One of the reasons for this overhead is network virtualization in DomR because the network in Dom0 is not virtualized.

5.3 Co-migration Time

We measured the time needed for the 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 an RMS client did not connect to an RMS server and when it sent an input every 50 ms to the server. This input rate is too fast for humans, but it is possible when we copy and paste text. For comparison, we migrated two independent DomUs in parallel without synchronization, 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.

Figures 10 and **11** show the average co-migration time for VNC and SSH, respectively, when we measured that 10 times for each memory size. In both cases, 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 without remote management increased by 1.6 seconds in VNC and 0.5 seconds in SSH at most. This is the overhead of the synchronization in D-MORE. When we performed remote management during co-migration, the co-migration time increased even more. The reason is that a larger amount of memory became dirty in DomR and DomU. Therefore it took time to enter the final stage of VM migration.

5.4 Downtime

We measured the downtime of DomR and DomU during co-migration. This experiment was conducted for VNC using the same setup as the above. The downtime we measured was the time in which a VM was not running at either the source or destination host. The average downtime of 10 runs was shown in **Fig. 12**. Since the standard deviations of all of these downtimes were quite large (tens of milliseconds), we omitted error bars for visibility. The downtime of DomR was higher than that of DomU

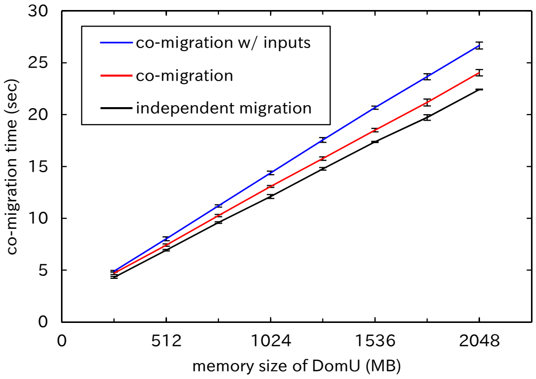


Fig. 10 The co-migration time for VNC.

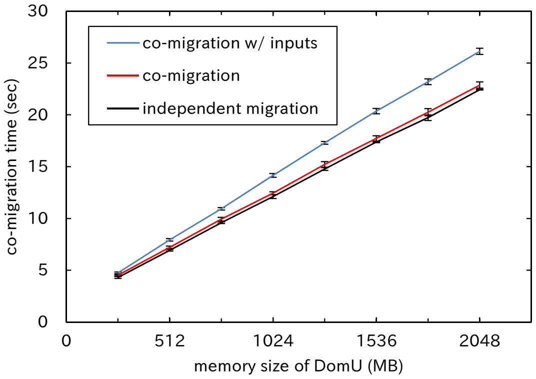


Fig. 11 The co-migration time for SSH.

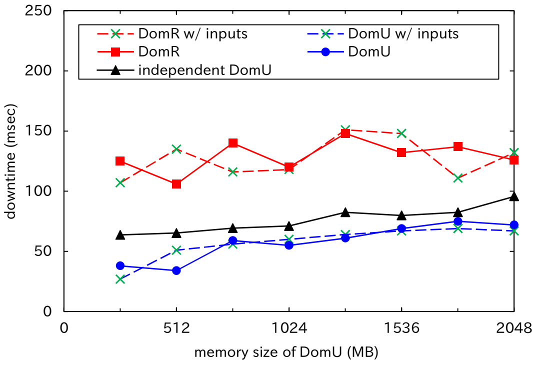


Fig. 12 The downtime of DomR and DomU.

because DomR was 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 it was restarted. For DomU, in contrast, the downtime in D-MORE was shorter than that of independent migration. The reason is probably that DomU could execute its migration process with less resource conflicts while the migration process of DomR waited for synchronization. It should be noted that the downtime did not become longer when we performed out-of-band remote management during co-migration.

Next, we measured the user-perceived downtime at RMS clients. We sent an input every 50 ms from an RMS client and measured the response time. Then we considered the longest response time at the final stage of co-migration as the user-perceived downtime. This experiment was conducted for VNC and SSH. **Figure 13** shows the average downtime when we measured that 10 times. For VNC, the user-perceived downtime was 827 ms at most. The standard deviation was quite large (more than 100 ms) because the output processing in a VNC server was very complicated. For SSH, on the other hand, the maximum of the downtime was 667 ms and the standard deviation was small. Although these downtimes are not small, they are much smaller than the downtimes in traditional systems that need manual reconnection. If a user reconnects to a new RMS server by himself after VM migration, it would take several tens of seconds at least. In addition, it may take a longer time to recover inputs lost during VM migration.

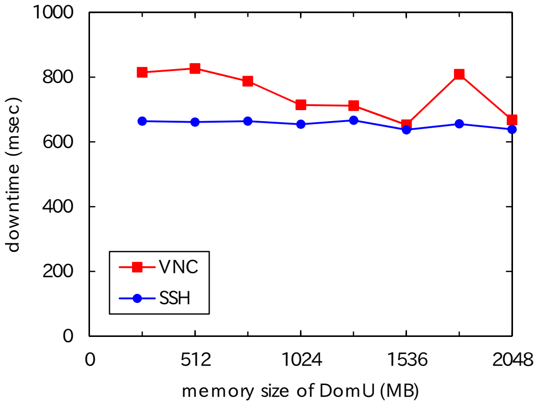


Fig. 13 The user-perceived downtime.

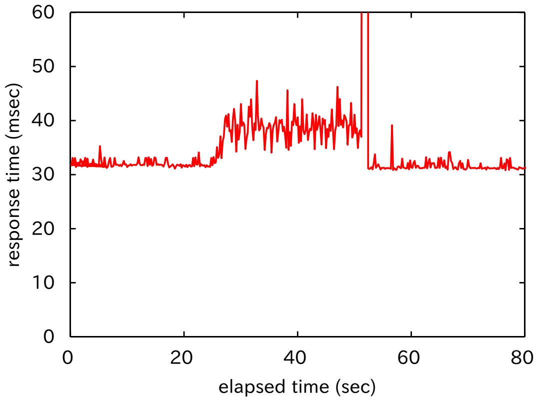


Fig. 14 The changes in response time during co-migration for VNC.

5.5 Performance Degradation during Co-migration

Even except for the downtime, 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 measured the changes in response time during co-migration. In this experiment, an RMS client sent input every 50 ms to an RMS server. **Figures 14** and **15** show the results for VNC and SSH, respectively. After we started co-migration at time 25 seconds, the response time increased by 5.4 ms for VNC and 2.9 ms for SSH on average. This performance degradation lasted for 28 seconds for VNC and 25 seconds for SSH.

To examine the impact on the throughput, we measured the changes of the frame rate of full-screen updates for VNC and the rate of console outputs for SSH during co-migration. For a full-screen update, we ran the screen saver in Section 5.2. **Figure 16** shows the average frame rate every second. The frame rate decreased by 0.4 frames per second (fps) on average after the

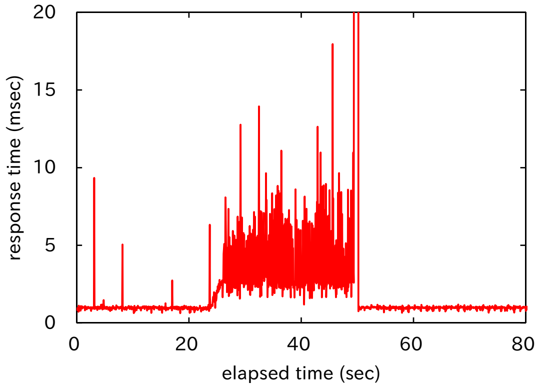


Fig. 15 The changes in response time during co-migration for SSH.

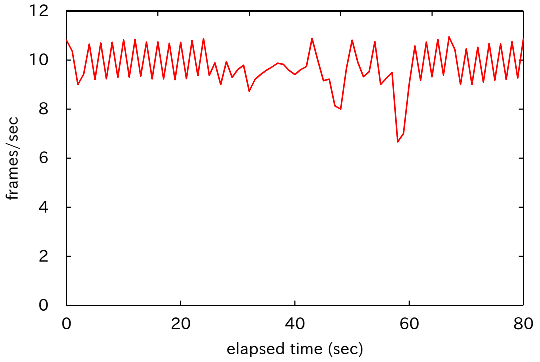


Fig. 16 The changes in frame rate in VNC during co-migration.

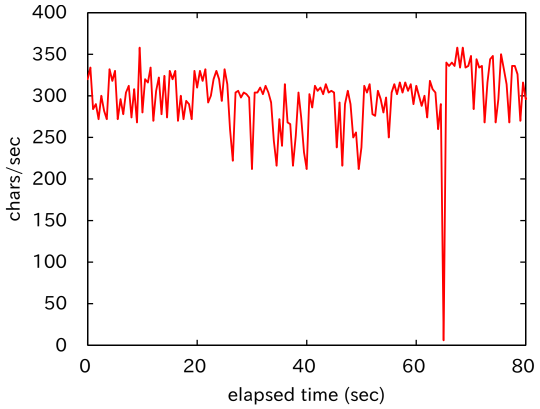


Fig. 17 The changes in output rate in SSH during co-migration.

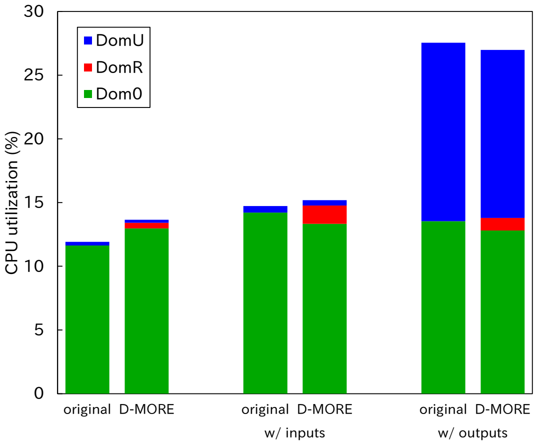


Fig. 18 CPU utilization in remote management with VNC.

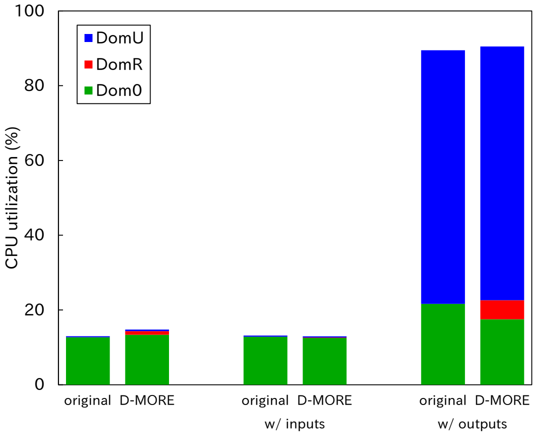


Fig. 19 CPU utilization in remote management with SSH.

co-migration was started at time 25 seconds. The degradation of the frame rate lasted for 33 seconds and the frame rate was 6.7 fps at the final stage of co-migration. For SSH, we ran a benchmark that wrote one character to a serial console every millisecond. **Figure 17** shows the average output rate every 0.5 seconds. The output rate degraded by 17.6 characters per second on average for 40 seconds. It was nearly zero at the final stage of co-migration.

5.6 CPU Overhead

We examined the CPU utilization of Dom0, DomR, and DomU in the original Xen and D-MORE. **Figures 18** and **19** show the average CPU utilization in out-of-band remote management with VNC and SSH, respectively. When an RMS client just connected to an RMS server, the total CPU utilization in D-MORE slightly increased, as shown in the left two bars. However, DomR did not use so much CPU time. When an RMS client sent an input every 50 ms, the total CPU utilization in D-MORE was almost the same as that in the original Xen, as in the middle two bars. For VNC, the CPU utilization of DomR became larger. When DomU executed full-screen updates in VNC or console outputs in SSH, the total CPU utilization was almost the same in the right two bars. For SSH, the CPU utilization of DomR was the largest among all workloads, but it was only 5% on average.

Next, we examined the changes of the CPU utilization during the co-migration of DomR and DomU. We measured the CPU utilization of Dom0s at the source and destination hosts, DomR, and DomU. **Figure 20** shows the result when a VNC client sent a key stroke every 50 ms. After co-migration was started at time 25 seconds, the CPU utilization of both Dom0s increased because migration managers ran. In addition, that of DomR slightly increased due to the overhead of the log dirty mode. **Figure 21** shows the CPU utilization when we used SSH and DomU wrote output to the console every millisecond. In this case, the CPU utilization of DomR increased to 19% on average due to co-migration. Since not only DomU but also DomR used more CPU time, the CPU utilization of Dom0 at the source host was much less than that of Fig. 21.

6. 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 an

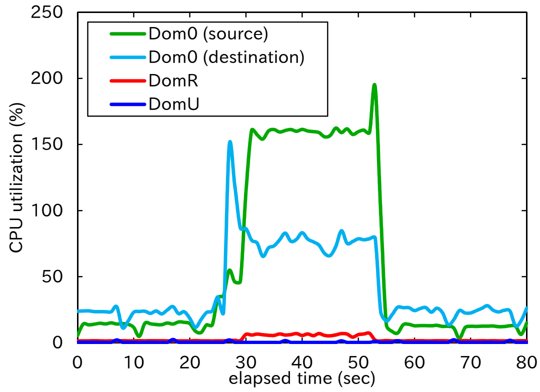


Fig. 20 The changes in CPU utilization during co-migration for VNC.

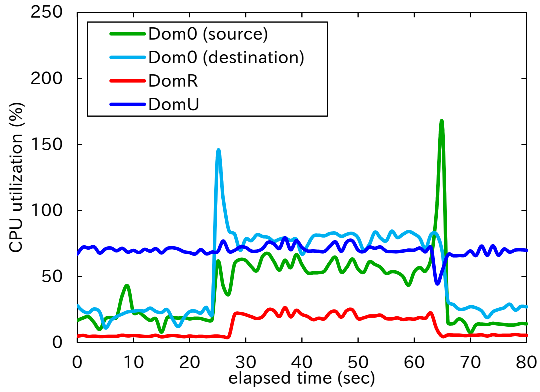


Fig. 21 The changes in CPU utilization during co-migration for SSH.

RMS server and virtual devices, which are necessary for the remote management of a target VM. D-MORE synchronously co-migrates DomR with its target VM and transparently maintains the connections between an RMS 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 after the co-migration. Our experimental results showed that all the pending data was not lost. The overhead of D-MORE was small during a normal run although performance degradation was caused during co-migration. The downtime during co-migration was not small, but it was much less than that of the traditional systems that need manual reconnection.

One of our future work is to support fully virtualized VMs in D-MORE. DomR could easily run virtual devices for them like stub domains [9], [10] in Xen, but we have to develop a mechanism for migrating them. Another future work is to reduce resource consumption by DomR. Running Xen's Mini OS can decrease the memory footprint of DomR.

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

References

[1] Kawahara, S. and Kourai, K.: The Continuity of Out-of-band Remote Management across Virtual Machine Migration in Clouds, *Proc. Intl. Conf. Utility and Cloud Computing*, pp. 176–185 (2014).

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

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

[4] Verma, A., Kumar, G., Koller, R. and Sen, A.: CosMig: Modeling the Impact of Reconfiguration in a Cloud, *Proc. Intl. Symp. Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, pp. 3–11 (2011).

[5] Apache Software Foundation: Apache CloudStack: Open Source Cloud Computing, available from ⟨http://cloudstack.apache.org/⟩.

[6] Red Hat: Spice, available from ⟨http://www.spice-space.org/⟩.

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

[8] Kourai, K. and Utsunomiya, H.: Co-migration of Virtual Machines with Synchronization for IDS Offloading, *IPSJ Trans. Advanced Computing Systems*, Vol. 7, No. 4, pp. 45–55 (2014).

[9] Nakajima, J. and Stekloff, D.: Improving HVM Domain Isolation and Performance, *Xen Summit September 2006* (2006).

[10] Thibault, S.: Stub Domains, *Xen Summit Boston 2008* (2008).

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

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

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

[14] Murray, D. G., Milos, G. and Hand, S.: Improving Xen Security through Disaggregation, *Proc. Intl. Conf. Virtual Execution Environments*, pp. 151–160 (2008).

[15] Deshpande, U., Wang, X. and Gopalan, K.: Live Gang Migration of Virtual Machines, *Proc. Intl. Symp. High Performance Distributed Computing*, pp. 135–146 (2011).

[16] Kivity, A. and Tosatti, M.: Kernel Based Virtual Machine, available from ⟨http://www.linux-kvm.org/⟩.

[17] Milojičić, D. S., Douglis, F., Paindaveine, Y., Wheeler, R. and Zhou, S.: Process Migration, *ACM Comput. Surv.*, Vol. 32, No. 3, pp. 241–299 (2000).

[18] VMware Inc.: VMware vSphere Hypervisor, available from ⟨http://www.vmware.com/⟩.

[19] Bellard, F.: QEMU, available from ⟨http://qemu.org/⟩.

[20] TightVNC Group: TightVNC, available from ⟨http://www.tightvnc.com/⟩.

[21] The OpenBSD Project: OpenSSH, available from ⟨http://www.openssh.com/⟩.

Sho Kawahara received his Master's degree in Computer Science and Systems Engineering from Kyushu Institute of Technology in 2015. His current research interests are cloud computing and virtual machines.

Kenichi Kourai is an associate professor in the Department of Creative Informatics at Kyushu Institute of Technology. He received his Ph.D. degree from the University of Tokyo in 2002. He has been working on operating systems. His current research interests are dependable and secure systems using virtual machines.