Flexible and Efficient Partial Migration of Split-memory VMs

doi: https://doi.org/10.1109/CLOUD49709.2020.00044

Flexible and Efficient Partial Migration of Split-memory VMs

Takahiro Kashiwagi
Kyushu Institute of Technology
kashiwagi@ksl.ci.kyutech.ac.jp

Kenichi Kourai
Kyushu Institute of Technology
kourai@ksl.ci.kyutech.ac.jp

***Abstract*—Recently, virtual machines (VMs) with a large amount of memory are being widely used. For flexible migration of such large-memory VMs without large hosts, split migration has been proposed. It transfers VM fragments to multiple smaller hosts and runs a split-memory VM across those hosts with remote paging. However, the traditional method cannot migrate a split-memory VM efficiently because it always migrates the entire VM. In addition, it has to gather all the VM fragments to one host and transfer them from that host. To address these issues, this paper proposes flexible and efficient partial migration of split-memory VMs. In particular, *subst migration* migrates only part of a split-memory VM to enable the maintenance of some of the hosts running the VM. *Merge migration* efficiently consolidates VM fragments distributed across multiple hosts into one host by directly transferring a VM fragment from each host. Even if a split-memory VM itself causes remote paging during such partial migration, the consistency of the VM is maintained by retransferring and invalidating target memory. We have implemented partial migration in KVM and showed its efficiency.**

***Index Terms*—virtual machines, large-memory VMs, VM migration, partial migration, remote paging**

1. Introduction

Recently, virtual machines (VMs) with a large amount of memory are being widely used. As an extreme example, Amazon EC2 provides High Memory instances with 24 TB of memory [1]. Using such large-memory VMs enables fast in-memory databases [2], [3] and efficient big data analysis [4], [5]. One advantage of using VMs is service continuity on host maintenance by VM migration. However, it becomes more difficult to find destination hosts with sufficient free memory when larger-memory VMs are migrated. Since hosts with a large amount of memory are expensive, private clouds may not be able to afford to prepare large hosts as the destination of occasional VM migration. Even in public clouds, it would not be cost-effective to always preserve many large hosts in preparation to large-scale maintenance of data centers.

To address this issue, split migration has been proposed [6], [7]. It divides a large-memory VM into small VM fragments and transfers them to multiple smaller hosts, i.e., one main host and one or more sub-hosts. It transfers the state of the VM core such as virtual CPUs and devices to the main host. Also, it transfers likely accessed memory of the VM to the main host. The rest of the memory is transferred to the sub-hosts. After split migration, the VM runs across these hosts by performing remote paging [8]–[11] between the main host and one of the sub-hosts. Such a VM is called a *split-memory VM*. However, if a split-memory VM is migrated using traditional migration, it is necessary to always migrate the entire VM even when only some of the hosts are maintained. In addition, the migration performance largely degrades due to frequent remote paging caused by gathering VM fragments to one source host.

In this paper, we propose flexible and efficient partial migration of split-memory VMs. The partial migration enables part of a split-memory VM to be directly transferred from source hosts to destination hosts only when necessary. It is classified into four types: *subst*, *merge*, *split*, and *split/merge migration*. Subst migration transfers a VM fragment in the main host or a sub-host to a new host and substitutes only one host among the hosts running a split-memory VM. This method enables only some of the hosts running a split-memory VM to be maintained. Merge migration transfers VM fragments in the hosts running a split-memory VM to one host without remote paging. This method enables efficient consolidation of a split-memory VM and eliminates the overhead of remote paging.

We have implemented subst migration and merge migration in a system named *IPmigrate*, which is based on KVM [12]. Merge migration is considered as the combination of subst migration for the main host and all the sub-hosts, but we have implemented it separately because of the consistency and efficiency of the migration. If a VM causes remote paging during subst migration, IPmigrate retransfers the target memory or invalidates already transferred memory at the destination host. This guarantees that the entire memory in the target host is transferred without excess or deficiency. According to our experiments, it was shown that subst migration and merge migration could largely reduce the migration time by reduced memory transfers and parallel memory transfers.

The organization of this paper is as follows. Section 2 describes split-memory VMs and issues of its migration. Section 3 proposes flexible and efficient partial migration of split-memory VMs. Section 4 explains the implementation

of IPmigrate and Section 5 shows our experimental results. Section 6 describes related work and Section 7 concludes this paper.

2. Migration of Split-memory VMs

2.1. Split-memory VMs

VM migration moves a VM running in one host to another host without stopping it. Using VM migration, any hosts running VMs can be maintained without service disruption. VM migration is also used for load balancing by VM deconsolidation and power saving by VM consolidation. It first creates a new VM at the destination host and then transfers the memory of the target VM to the new VM. It retransfers the memory updated during VM migration and stops the VM if the amount of memory to be retransferred is small enough. In the final phase, it transfers the rest of the updated memory and the state of the VM core such as virtual CPUs and devices. Finally, it starts the execution of the new VM at the destination host.

VM migration requires a larger amount of free memory in the destination host than the memory size of a VM to be migrated. Recently, the memory size of VMs is increasing for in-memory database and big data analysis [1]. As a result, it becomes more difficult to find appropriate destination hosts for such large-memory VMs. Private clouds may not be able to prepare hosts with a large amount of memory as the destination of occasional VM migration because such large hosts are expensive and require a measurable amount of power. Even in public clouds, it would not be cost-effective to always preserve many large hosts in preparation to large-scale maintenance of data centers. If there is no appropriate host, a VM cannot be migrated and the users cannot use services provided by the VM during host maintenance.

For flexible migration of large-memory VMs without large hosts, split migration has been proposed [6], [7]. As illustrated in Fig. 1, it divides the memory of a VM into small VM fragments and transfers them to multiple smaller hosts. The destination host running the VM core such as virtual CPUs and devices is called the main host, while the other destination hosts are called sub-hosts. Split migration transfers the VM core and likely accessed memory to the main host and the rest of the memory to the sub-hosts. Such access prediction of the memory is performed using LRU on the basis of the memory access history of the VM.

After split migration, the migrated VM runs across multiple hosts. This VM is called a *split-memory VM*. Since the memory of the VM is distributed, a split-memory VM runs by performing remote paging between the main host and one of the sub-hosts. When the VM core requires the memory existing in a sub-host, that memory is paged in from the sub-host to the main host. In exchange, the most unlikely accessed memory is paged out from the main host to that sub-host. Since likely accessed memory has been transferred to the main host in advance at the migration time, the frequency of remote paging is suppressed just after split migration.

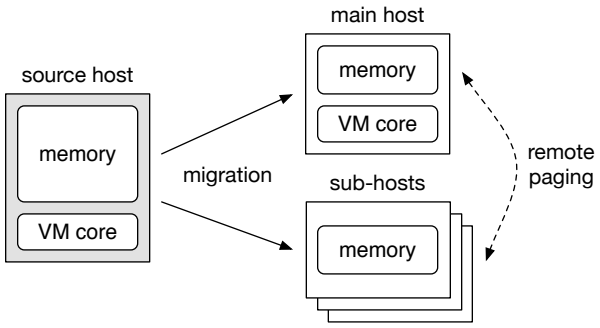


Figure 1: Split migration.

2.2. Migration Issues

In several situations, a split-memory VM is migrated again. When a host with a larger amount of free memory becomes available, it could be desirable to actively migrate a split-memory VM to that large host. When the working set size is less than the memory size in the main host, the performance of a split-memory VM is as high as before split migration [7]. However, if the working set size exceeds the memory size, the performance degradation is crucial due to frequent remote paging. The overhead of remote paging can be mitigated by using RDMA [8], [10], [11] and correctly identifying unused memory with machine learning [13], but it is difficult to make this overhead negligible. Even if the newly found host cannot accommodate the entire memory of a split-memory VM, that host could be used as a better main host by performing split migration.

In addition, when hosts running a split-memory VM are maintained, the VM has to be always migrated to other hosts. Since a split memory VM runs across multiple hosts, the probability of maintaining one of those hosts becomes higher. If possible, a split-memory VM should be migrated to one large host and be run as a normal VM to avoid remote paging. If there is no such large host, split migration needs to be applied to a split-memory VM again.

However, two issues arise if a split-memory VM is migrated using traditional one-to-one migration or split migration. First, unnecessary transfers of VM fragments can occur because traditional migration always migrates the entire split-memory VM. Even when only some of the hosts running a split-memory VM are maintained, VM fragments in the other hosts have to be transferred to the destination host(s) as well. This is inflexible and inefficient because these hosts are still available and their VM fragments do not have to be transferred. It would need a longer time before completing the migration and starting host maintenance. The migration would consume more CPU time and network bandwidth. This can lead to the performance degradation of the migrated VM.

Second, the migration performance largely degrades because remote paging frequently occurs during VM migra-

TABLE 1: The classification of partial migration. A, B, and C are the existing hosts, while P and Q are new hosts. Boldface means the main host running the VM core. a to d are VM fragments.

type	before migration	after migration
Subst	$\boldsymbol{A}[a], B[b], ...$	$\boldsymbol{P}[a], B[b], ...$
		$\boldsymbol{A}[a], P[b], ...$
Merge	$\boldsymbol{A}[a], B[b], ...$	$\boldsymbol{P}[ab], ...$
		$\boldsymbol{A}[ab], ...$
		$\boldsymbol{B}[ab], ...$
	$\boldsymbol{A}[a], B[b], C[c], ...$	$\boldsymbol{A}[a], P[bc], ...$
		$\boldsymbol{A}[a], B[bc], ...$
Split	$\boldsymbol{A}[ab], ...$	$\boldsymbol{P}[a], Q[b], ...$
		$\boldsymbol{A}[a], P[b], ...$
		$\boldsymbol{P}[a], A[b], ...$
	$\boldsymbol{A}[a], B[bc], ...$	$\boldsymbol{A}[a], P[b], Q[c], ...$
		$\boldsymbol{A}[a], B[b], P[c], ...$
Split/ Merge	$\boldsymbol{A}[a], B[bc], ...$	$\boldsymbol{A}[ab], P[c], ...$
		$\boldsymbol{A}[ab], B[c], ...$
	$\boldsymbol{A}[a], B[b], C[cd], ...$	$\boldsymbol{A}[a], B[bc], P[d], ...$
		$\boldsymbol{A}[ac], B[bd], ...$

tion. Traditional migration transfers the memory of a VM from only one source host to the destination host(s). For a split-memory VM, traditional migration causes many page-ins from sub-hosts to the main host because the main host has to transfer the memory existing in sub-hosts after the page-ins. According to our experiment, it took 6x longer time to migrate a split-memory VM using traditional one-to-one migration, compared with migrating a normal VM. For a VM with 24 GB of memory, the amount of paged-in memory reached about 24 GB. Accordingly, CPU utilization and network bandwidth would increase in the main host.

3. Partial Migration of Split-Memory VMs

To address the two issues of migrating split-memory VMs, this paper proposes flexible and efficient partial migration of split-memory VMs. Partial migration moves part of or the whole of a split-memory VM running in the source hosts to the destination hosts. The source and destination hosts can be completely different, partially different, or exactly the same. Each source host *directly* transfers part of a split-memory VM to some of the destination hosts. This enables efficient VM migration because it is unnecessary to transfer the memory of a split-memory VM existing in the source sub-hosts via the source main host. Table 1 classifies possible partial migration.

Subst migration is a type of partial migration that transfers a VM fragment in one host to a new host. It can be applied to either the main host or a sub-host. Subst migration for the main host transfers both the VM core and part of the memory of a split-memory VM, while that for a sub-host transfers only part of the memory of a split-memory VM. This migration enables efficient migration of a split-memory VM when only several hosts are maintained among multiple hosts running that VM.

Merge migration is a type of partial migration that consolidates VM fragments in several hosts into one host. When merge migration is applied to the main host and sub-hosts, the VM core in the main host and part of the memory of a split-memory VM existing in the source hosts are transferred to a new main host or one of the source hosts. When merge migration is applied only to sub-hosts, part of the memory of a split-memory VM existing in the source sub-hosts is transferred to a new sub-host or one of the source sub-hosts. If a new host with sufficient free memory is found, merge migration can be performed to that host. If the existing host releases memory, e.g., by terminating other VMs, it can be the destination of merge migration. This migration is useful for reducing remote paging caused by the split-memory VM and running VM fragments in high-performance sub-hosts.

Split migration is also a type of partial migration that divides a VM fragment in one host into multiple hosts. Previously proposed split migration [6], [7] is a special case of dividing a normal VM in one host into multiple new hosts. When split migration is applied to the main host, part of the memory of a split-memory VM is first divided into several pieces. One memory piece and the VM core are transferred to a new main host or are kept in the source main host. The other memory pieces are transferred to new sub-hosts or are kept in the source host, which is no longer the main host. When split migration is applied to a sub-host, all the memory pieces are transferred to new sub-hosts or only one piece is kept in the source sub-host. This migration is used when a substitute host with sufficient free memory is not found on subst migration. If a host used for a split-memory VM runs out of memory in some reason, split migration can transfer part of the VM fragment to other hosts.

Split/merge migration is complex partial migration that performs split migration for the source hosts and merge migration for the destination hosts. For example, it can divide a VM fragment in one host into multiple fragments and consolidates each into multiple existing hosts. It can transfer only several VM fragments to new hosts. Also, it can move only part of VM fragments between the main host and a sub-host or between sub-hosts.

In this paper, we focus on subst migration for the main host and a sub-host and merge migration for all hosts. This is because these are the basic forms of various types of partial migration and contains common research issues to be addressed.

3.1. Subst Migration

When subst migration is applied to the main host, it transfers the VM core and part of the memory of a split-memory VM existing in the source main host to a new destination main host, as in Fig. 2. At this time, it does not transfer the memory data existing in sub-hosts but only information on that memory. After subst migration is completed, the destination main host runs the migrated split-memory VM and performs remote paging between that host and the sub-hosts in which VM fragments are kept. Similarly, when subst migration is applied to a sub-host, it transfers only part of the memory of a split-memory VM existing in the source sub-host to a new destination sub-host, as illustrated in Fig. 3. After the migration, the main

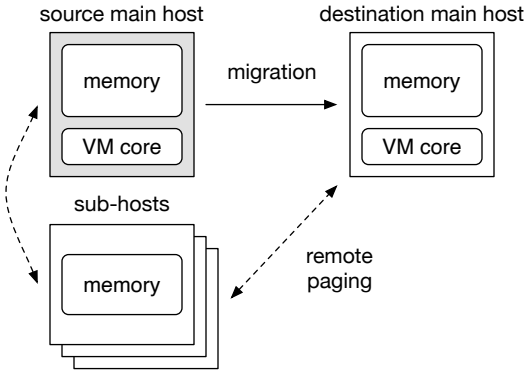


Figure 2: Subst migration for the main host.

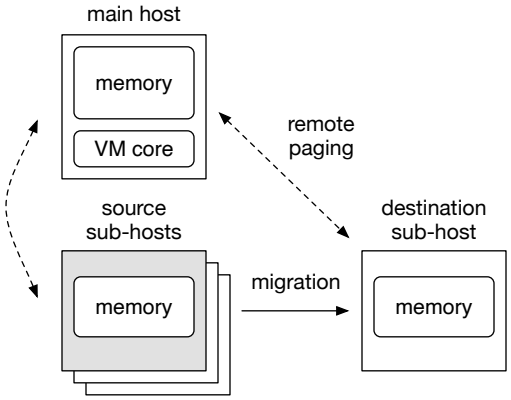


Figure 3: Subst migration for a sub-host.

host performs remote paging to the new sub-host and the sub-hosts in which VM fragments are kept.

When a split-memory VM causes remote paging by accessing the memory existing in a sub-host during subst migration, IPmigrate not only transfers the memory without excess or deficiency but also preserves consistency in the entire memory of the split-memory VM. For subst migration for the main host, when a page-in occurs to the main host during the migration, IPmigrate transfers the target memory from the source main host to the destination main host if necessary. This transfer is required when the target memory is never transferred yet or when it is not transferred after updated. When a page-out occurs from the main host during the migration, IPmigrate invalidates the target memory in the destination main host if the memory has been already transferred. This is necessary to prevent the same memory region from existing in both the sub-host and the destination main host.

For subst migration for a sub-host, IPmigrate also performs similar operations for consistency when remote paging occurs during the migration. When a page-out occurs to the sub-host, IPmigrate transfers the target memory to the destination sub-host if necessary. If the target memory has been already transferred to the destination sub-host, IPmigrate invalidates the memory in the destination sub-host.

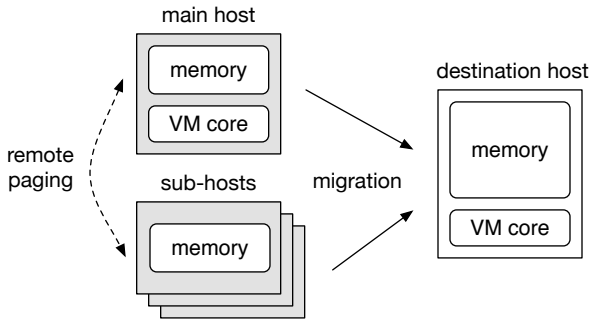


Figure 4: Merge migration for all hosts.

3.2. Merge Migration

Fig. 4 illustrates merge migration for all hosts. Conceptually, merge migration can be achieved by combining subst migration for the main host and all the sub-hosts and specifying the same host as the destination. However, it is difficult to efficiently perform merge migration without excess or deficiency only by that simple combination. To achieve efficient merge migration, IPmigrate supports parallel memory transfers in the network and the destination host and enables subst migration for multiple hosts in parallel.

When remote paging occurs during merge migration, IPmigrate performs page-ins and page-outs with additional information on whether the memory should be transferred or not. It is necessary to pass such information between the source main host and sub-hosts because each source host independently transfers part of the memory of a split-memory VM in merge migration. Even if the memory that is not yet transferred is paged in or out, it is guaranteed that the memory is transferred to the destination host. In addition, IPmigrate does not invalidate the transferred memory even if remote paging occurs for that memory. Since the entire memory is finally transferred to one destination host in merge migration, the same memory region never exists in multiple hosts unlike subst migration for a single host.

4. IPmigrate

We have implemented subst migration for the main host and a sub-host and merge migration for all of the hosts running a split-memory VM. For these migration methods, we have developed a system named *IPmigrate* using QEMU-KVM 2.4.1 and Linux 4.3.

4.1. System Architecture

The system architecture of IPmigrate is illustrated in Fig. 5. The main host runs QEMU-KVM modified for IPmigrate, while each sub-host runs a memory server maintaining part of the memory of a split-memory VM. The modified QEMU-KVM maintains the *network page table* and the *sub-host table*. The network page table manages the mapping from a page frame number to a host identifier to enable

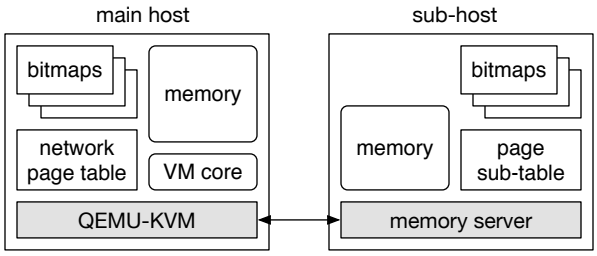


Figure 5: The system architecture of IPmigrate.

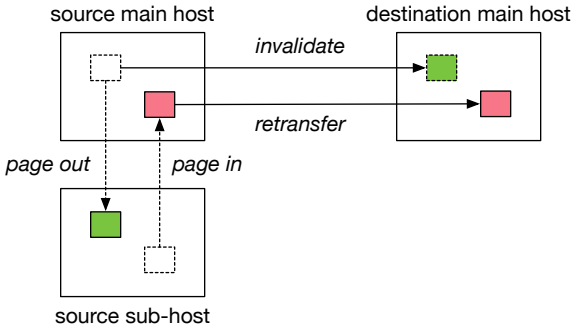


Figure 6: Handling remote paging during subst migration for the main host.

remote paging. The sub-host table manages the mapping from a host identifier to the IP address of the host. A memory server maintains the *page sub-table* for managing the mapping from a page frame number to the pointer to the memory data contained in the page.

In addition to these tables, each source host involved in partial migration maintains three bitmaps: transfer, retransfer, and invalidate bitmaps. The *transfer bitmap* manages whether each page has been transferred or not. The *retransfer bitmap* manages whether each page should be (re)transferred or not. For this bitmap, the main host uses the dirty bitmap, which manages updated pages for VM migration in the existing QEMU-KVM. The *invalidate bitmap* manages whether each transferred page should be removed or not at the destination host.

4.2. Subst Migration for the Main Host

At the source main host, QEMU-KVM transfers part of the memory of a split-memory VM to the destination main host using the network page table. For a page existing in the main host, it transfers the memory data and sets the corresponding bit in the transfer bitmap. For a page existing in a sub-host, it transfers only the IP address of the sub-host to the destination main host. If pages are updated during subst migration, QEMU-KVM retransfers them on the basis of the retransfer bitmap. Finally, it waits for the completion of pending paging requests and stops the VM. Then, it transfers the rest of the memory and the state of the VM core and completes the migration.

At the destination main host, QEMU-KVM registers the entire memory region used by a new VM to the userfaultfd mechanism in Linux when the migration is started. userfaultfd is a mechanism for notifying the QEMU-KVM process of information on page faults that occur for the registered memory region. When QEMU-KVM receives memory data from the source main host, it allocates a new page to the VM and stores the data in the page. Then, it registers the page to the network page table to record that the page exists in the main host. When it receives only the IP address of a sub-host, it registers the page to the table to record that the page exists in that sub-host. After QEMU-KVM receives information on the entire memory and the VM core, it establishes the connections to the source sub-hosts and restarts the execution of the split-memory VM. After the migration, IPmigrate performs remote paging between the destination main host and the source sub-hosts.

When a page-in occurs from a sub-host during the migration, QEMU-KVM in the source main host examines the transfer bitmap. If the target page has not yet been transferred, QEMU-KVM sets the corresponding bit in the retransfer bitmap. Then, it transfers the page to the destination main host later, as illustrated in Fig. 6. When a page-out occurs to a sub-host, QEMU-KVM similarly examines the transfer bitmap. Unlike the case of page-ins, it sets the corresponding bit in the invalidate bitmap if the target page has been already transferred to the destination main host. On the basis of the invalidate bitmap, QEMU-KVM transfers an invalidation request to the destination main host later. Then, QEMU-KVM in the destination main host removes the target page from the memory region of the VM.

4.3. Subst Migration for a Sub-host

To execute subst migration for a sub-host, IPmigrate first sends a special command to QEMU-KVM in the main host using the QEMU machine protocol (QMP). This is because this migration does not move the VM core to the destination host unlike traditional migration. When QEMU-KVM receives this command, it sends a request for subst migration to the specified sub-host. At that source sub-host, the memory server transfers the memory data of each page existing in that host to the destination sub-host. Then, it sets the corresponding bit in the transfer bitmap.

If remote paging occurs during the migration, IPmigrate handles it in a similar manner to subst migration for the main host, as illustrated in Fig. 7. When a page-in occurs to the main host, the memory server in the source sub-host examines the transfer bitmap. If the target page has been transferred to the destination sub-host, the memory server sets the corresponding bit in the invalidate bitmap. The memory server sends an invalidation request to the destination sub-host later. Unlike subst migration for the main host, the memory server in the destination sub-host removes the page from the page sub-table and releases it. When a page-out occurs from the main host, the memory

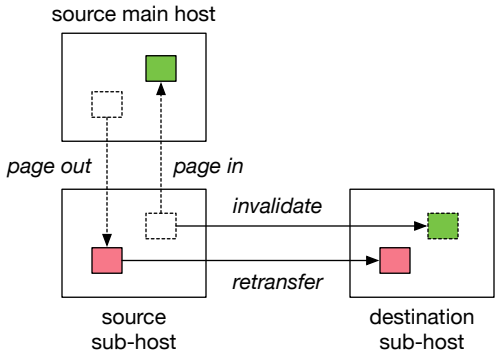


Figure 7: Handling remote paging during subst migration for a sub-host.

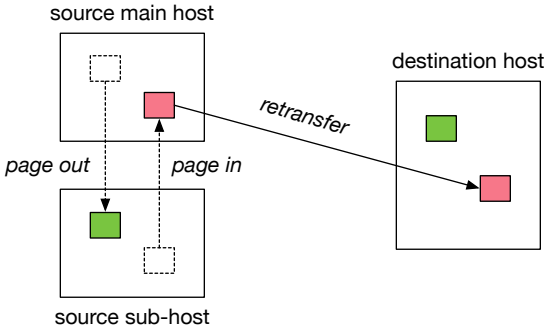


Figure 8: Handling remote paging during merge migration.

server sets the corresponding bit in the retransfer bitmap. It transfers the page to the destination sub-host later.

If the number of pages to be transferred becomes small enough, the memory server sends a request for temporarily stopping remote paging to the main host. When QEMU-KVM in the main host receives this request, it acquires a lock so that it does not send any page-in or page-out requests to the sub-host. In addition, it waits for the completion of pending paging requests. After that, the memory server transfers the rest of the pages to be transferred and sends completion notification to the main host. QEMU-KVM in the main host updates the sub-host table so that the sub-host identifier is mapped to the IP address of the destination sub-host. Finally, it establishes the connection to the destination sub-host and releases the lock for remote paging.

4.4. Merge Migration

Merge migration basically performs subst migration for the main host in parallel with that for all the sub-hosts to the same destination host. When QEMU-KVM in the destination host receives memory data from the source main host, it handles the data in a similar manner to subst migration for the main host. Unlike single subst migration, QEMU-KVM in the source main host does not transfer memory information on the pages existing in the sub-hosts. Such information is not necessary because memory data of those pages are transferred from the sub-hosts. For memory data transferred from the source sub-hosts, QEMU-KVM in the destination host handles received data unlike subst migration for sub-hosts. The thread dedicated for each source sub-host receives data and stores it in the memory of a new VM.

IPmigrate enables parallel memory transfers from the main host and sub-hosts by using multiple NICs at the destination host. Using policy-based routing in iproute2, IPmigrate creates a routing table for each NIC and adds a rule for referring to the routing table on the basis of the IP address assigned to the NIC. Without such special routing, only one NIC can be used to send packets in one network segment. It is possible to use network interface bonding, but this configuration depends on network switches. The link aggregation function in several switches transfers data to only one NIC.

When the memory transfer is completed from one of the source hosts, IPmigrate finishes the migration for that host. If the memory transfer is completed earlier from the source main host, QEMU-KVM in the destination host establishes the connections to the source sub-hosts for which the migration is not completed. Then, it runs the migrated VM whose memory is still split using remote paging. Note that page-outs are not performed because the destination host has sufficient free memory in merge migration. After that, merge migration becomes simple subst migration for sub-hosts. If the memory transfer is completed from a source sub-host, IPmigrate just finishes subst migration for that sub-host. The VM can directly access the memory that has been transferred from the sub-host in the destination host.

If the memory transfer from a sub-host is completed earlier than from the main host, QEMU-KVM in the main host connects to the destination host. Then, QEMU-KVM in the destination host needs to perform remote paging as a memory server. Otherwise, we need to run a memory server to share the memory of the VM and perform remote paging. Since neither implementation was easy, the memory server in the sub-host does not complete the migration but waits at this time. When it receives a page-in or page-out request, it handles that request. After QEMU-KVM in the source main host stops the VM in the final phase, it sends a completion request to the sub-host. Then, the memory server in the sub-host finishes the migration.

When a page-in occurs to the source main host during the migration, the memory server in the source sub-host sends the target page as well as whether the page has to be transferred or not to the source main host. A page needs to be transferred if the corresponding bit is not set in the transfer bitmap or if that is set in the retransfer bitmap. According to the received information, QEMU-KVM in the source main host transfers the target page to the destination host by setting the corresponding bit in the retransfer bitmap, as illustrated in Fig. 8. Unlike single subst migration for a sub-host, the memory server does not invalidate the paged-in page.

Similarly, when a page-out occurs to a source sub-host during the migration, QEMU-KVM in the source main host

sends the target page as well as whether the page has to be transferred or not to the destination host. The memory server in the sub-host transfers the page only if the page has to be transferred. Unlike single subst migration, QEMU-KVM does not invalidate the paged-out page.

5. Experiments

We conducted experiments for examining the performance of subst migration and merge migration. For comparison, we performed traditional one-to-one migration for a normal VM. For the (main) hosts running the VM core, we used two PCs with an Intel Xeon E3-1270 v3 processor, 32 GB of memory, and two Intel X540-T2 dual-port adaptors for 10 Gigabit Ethernet (GbE). We ran Linux 4.3 and QEMU-KVM 2.4.1. For sub-hosts, we used up to three PCs. Two PCs equipped with an Intel Core i7-8700 processor, 64 GB of memory, and an Intel X550-T2 adaptor and ran Linux 4.15. The other PC equipped with an Intel Xeon E3-1270 v2 processor, 16 GB of memory, and an Intel X540-T2 adaptor and ran Linux 4.3. These five PCs were connected with a 10 GbE switch.

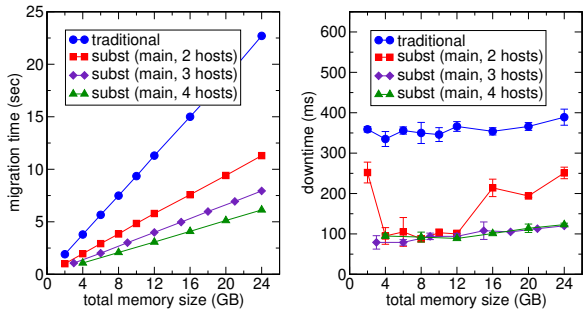
We created a VM with one virtual CPU and changed the memory size from 4 to 24 GB. This VM did not have so large memory due to the hardware limitation, but that was sufficient to show the effectiveness of partial migration. For a split-memory VM, we equally divided the memory into 2 to 4 hosts. We measured the migration time and the downtime at least 5 times.

5.1. Performance of Subst Migration

To examine the performance of subst migration, we first measured the migration time and the downtime in subst migration for the main host. In this experiment, we used an idle VM without running any active applications. Fig. 9(a) shows the time needed for migrating a split-memory VM across various numbers of hosts. Compared with traditional migration of a normal VM, the migration time was reduced to approximately 50%, 33%, and 25% for a split-memory VM across 2, 3, and 4 hosts, respectively. This is because the migration time is basically proportional to the amount of transferred memory. In subst migration, only the memory existing in the main host was transferred.

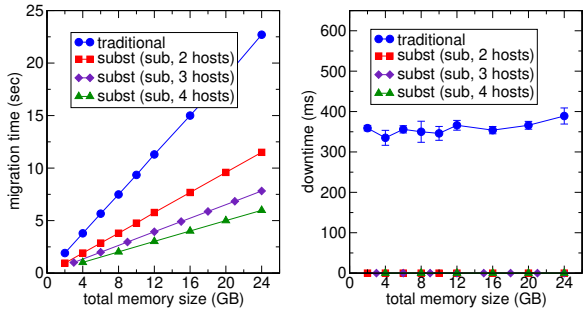
Fig. 9(b) shows the downtime during the migration. The downtime of subst migration for the main host was always shorter than that of traditional migration. This is because QEMU-KVM could not accurately estimate the downtime. QEMU-KVM enters the final phase of the migration when it estimates that the rest of the memory will be transferred in 300 ms. In subst migration, the rest of the memory often includes the pages existing in sub-hosts. Therefore, the final memory transfer was completed earlier because only a smaller amount of memory was actually transferred to the destination host than during the estimation.

The downtime tended to slightly increase as the total memory size of a VM became larger. This is because it took a longer time for QEMU-KVM to check the entire



(a) Migration time (b) Downtime

Figure 9: The performance of subst migration for the main host.



(a) Migration time (b) Downtime

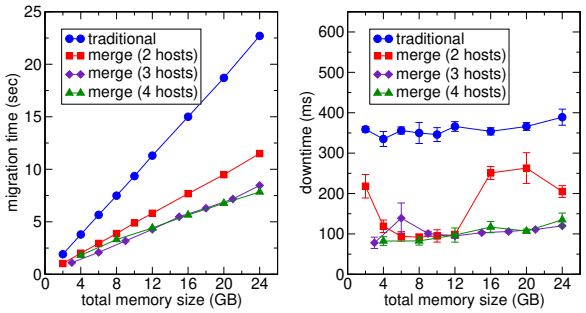
Figure 10: The performance of subst migration for a sub-host.

dirty bitmap for memory retransfers if the memory size was larger. For a split-memory VM across 2 hosts, the downtime increases largely in several memory sizes, i.e., 2 GB and more than 12 GB. This reason is under investigation.

Next, we measured the migration time in subst migration for a sub-host. Fig. 10(a) shows the migration time of a split-memory VM across various numbers of hosts. The migration time was very similar to that in subst migration for the main host because the amount of transferred memory was almost the same. However, the proportion of memory-unrelated tasks was smaller than that in subst migration for the main host. This is because subst migration for a sub-host is simpler than that for the main host. Unlike subst migration for the main host, the downtime was zero, as shown in Fig. 10(b). Since this migration does not transfer the VM core, it does not need to stop the VM. If the split-memory VM causes remote paging during the final phase of the migration, only that virtual CPU is suspended until the final phase finishes. This can become the downtime, but remote paging did not occur at the final phase in our experiment.

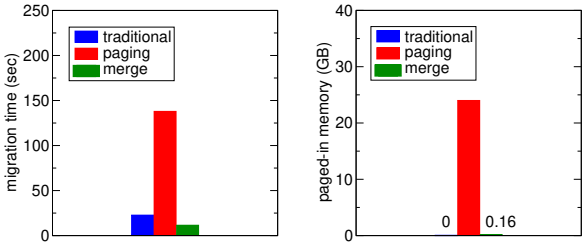
5.2. Performance of Merge Migration

To examine the performance of merge migration, we measured the migration time and the downtime. Fig. 11(a)



(a) Migration time (b) Downtime

Figure 11: The performance of merge migration.



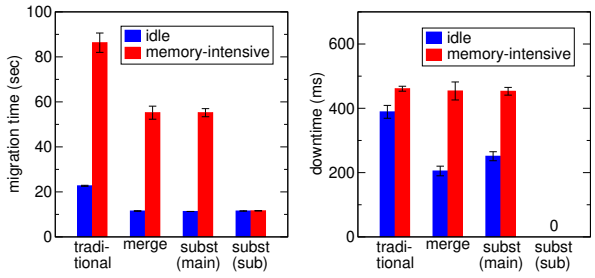
(a) Migration time (b) Page-ins

Figure 12: Comparison between merge migration and paging migration.

shows the migration time of a split-memory VM across various numbers of hosts. Compared with traditional migration of a normal VM, the migration time was reduced to approximately 50% and 33% for a split-memory VM across 2 and 3 hosts, respectively. However, the proportion of memory-unrelated tasks was larger than that in subst migration. This is because of the complexity of the mechanism of merge migration. On the other hand, the migration time of a split-memory VM across 4 hosts was almost the same as that across 3 hosts or even increased. This is due to the hardware limitation. The PC used for the destination main host did not have a sufficient bandwidth of PCI Express for four 10 GbE communications. Therefore, we used only three out of four network ports provided by two NICs.

Fig. 11(b) shows the downtime. This result is very similar to that in subst migration for the main host. The downtime in merge migration was less than that in traditional migration. It tended to increase gradually as the memory size of a VM increased. For a split-memory VM across 2 hosts, the downtime was long for several memory sizes of a VM.

Finally, we compared merge migration with *paging migration*, which migrated a split-memory VM to one host using traditional one-to-one migration. As pointed out in Section 2.2, paging migration caused frequent remote paging. In this experiment, we ran a split-memory VM with 24 GB of memory across 2 hosts. Fig. 12(a) shows the migration time. Paging migration of a split-memory VM took 6.1x longer time than traditional migration of a normal VM. Merge migration could improve the performance by



(a) Migration time (b) Downtime

Figure 13: The performance of various types of migration for a memory-intensive VM.

12x. This performance improvement came from the dramatic reduction of page-ins, as shown in Fig. 12(b). In merge migration, all of the page-ins are caused by the internal activity of the VM.

5.3. Migration of a Memory-intensive VM

To examine the migration performance of a memory-intensive VM, we performed partial migration of a VM with 24 GB of memory across two hosts. A benchmark program modified 6 GB of the memory repeatedly. Fig. 13(a) shows the migration time of the memory-intensive VM and, for comparison, an idle VM. Except for subst migration for a sub-host, the migration time of the memory-intensive VM became longer than that of the idle VM. This is because a large amount of memory was retransferred due to continuous memory updates. The reason why the migration time of subst migration for a sub-host did not increase is that memory retransfers were not performed. Since the size of the working-set memory was smaller than the physical memory size of the main host, updated memory was not paged out to the sub-host. It should be noted that the increase in migration time of merge migration and subst migration for the main host was smaller than that of traditional migration.

Fig. 13(b) shows the downtime of the two VMs. Compared with the idle VM, the memory-intensive VM increased the downtime in all the migration methods. As a result, the downtime became almost the same. This means that the amount of memory to be transferred in the final phase became almost the same. As described in the above experiments, QEMU-KVM stops a VM if the amount of memory to be transferred becomes less than the threshold. In traditional migration, that amount of transferred data is almost the same between the memory-intensive VM and the idle VM because the entire memory of a VM exists in the source host. In merge migration and subst migration for the main host, in contrast, this is not always the case. The amount of transferred data in the memory-intensive VM is often larger than that in the idle VM. The memory to be transferred can include pages existing in the sub-host for the idle VM. For the memory-intensive VM, that did not include such pages because the transferred memory is updated one, which usually exists in the source main host. This means that

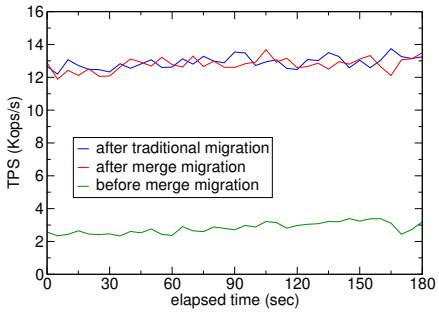


Figure 14: The performance of memcached after merge migration.

the downtime of the memory-intensive VM is rather correct in terms of the migration algorithm of QEMU-KVM.

5.4. Performance after Merge Migration

To examine the performance of a VM after merge migration, we ran memcached [14] in a split-memory VM and measured its throughput after the migration. For comparison, we also measured that after traditional one-to-one migration of a normal VM. We used a VM with 24 GB of memory and assigned 12 GB of memory to memcached running in it. Before the migration, we set data to memcached for 300 seconds using the memaslap benchmark [15] with 100% SET operation. After the migration, we accessed data in memcached for 180 seconds using the benchmark with 60% SET and 40% GET operations.

Fig. 14 shows the changes in throughput after VM migration. The throughput after merge migration was 12.7 Kops/s and almost the same as that after traditional migration of a normal VM. Since the size of working-set memory exceeded the physical memory size of the main host in this experiment, the throughput before merge migration was only 2.8 Kops/s. This performance degradation can be mitigated by using InfiniBand RDMA [8], [10], [11] instead of TCP/IP used in IPmigrate, but it is difficult to make the overhead of remote paging negligible. Merge migration could resolve this performance issue successfully by migrating a VM to one host.

6. Related Work

Post-copy VM migration [16] transfers only the VM core indispensable for running a VM to the destination host and then restarts the VM quickly. This is considered one type of split migration in that the VM core and the memory exist in different hosts. After that, the memory left in the source host are transferred on demand like remote paging or in the background. This background transfer is one type of merge migration and corresponds to the case that the memory transfer is completed earlier from the main host. Unlike merge migration, remote paging does not need to be considered so specially during this migration. Page-ins during the migration are a simple on-demand transfer because the VM core is moved to the destination host at first. Paged-in pages are not transferred in the background transfer. Page-outs are not performed during the migration.

Scatter-Gather live migration [17] uses multiple intermediate hosts between the source and destination hosts. It pushes the memory of a VM to intermediate hosts as fast as possible and the destination host obtains the memory from these hosts. This migration also performs both the on-demand and background transfers. The first half is similar to split migration, while the background transfer in the latter half is similar to merge migration. Like post-copy VM migration, this migration does not need to specially consider remote paging caused during the migration.

MemX [9] runs a VM using the memory of multiple hosts from the boot time like a split-memory VM. The guest operating system in a VM explicitly provides a block device to access the memory of the MemX servers in the other hosts. The migration of this VM is considered subst migration for the main host in that the memory of the MemX servers are not transferred. Since the guest operating system itself performs remote paging, the memory of the VM can be transferred without excess or deficiency even if remote paging occurs during the migration. MemX also provides the other modes transparent to the operating system, but VM migration is not supported in those modes.

MemX supports page migration, which transfers the memory of the MemX server to another server. This is similar to subst migration for a sub-host. However, remote paging during page migration is not considered or the migration performance is not evaluated. In addition, it is proposed to run the MemX sever in another VM and migrate the MemX server with the memory of the target VM. It is pointed out that this method can increase the overhead of remote paging.

vMotion provides two different migration methods in terms of swap space [18]. Unshared-swap vMotion uses different swap spaces between the source and destination hosts. After it transfers the memory of a VM accommodated in physical memory and restarts the VM at the destination host, it transfers memory data stored in the swap space of the source host to the destination host with the VM running. This is similar to merge migration if we consider the swap space in the source host as a sub-host. It corresponds to the case that subst migration for the main host is first completed and then that for the sub-host is performed. However, the memory transfers from the source host and the swap space cannot be done in parallel.

Shared-swap vMotion uses one swap space shared between the source and destination hosts. It transfers only the memory of a VM accommodated in physical memory to the destination host. The memory data in the swap space is not transferred. If we consider the shared swap space as a sub-host, this is similar to subst migration for the main host. Agile live migration [19] is similar to shared-swap vMotion, but it performs page-outs of the memory of a VM to the shared swap space as much as possible before the migration. This can reduce the amount of memory to be transferred during the migration. For this purpose, it keeps track of the

working-set memory of a VM and pages out the memory that is not needed by the VM. Since these migration methods do not consider memory paged-out during the migration, such memory can exist in both physical memory and the swap space at the destination host.

Jettison [20] performs partial migration of desktop VMs to reduce power consumption of desktop PCs. To rapidly consolidate inactive desktop VMs, it transfers only the VM core and the working-set memory of a desktop VM to the server. When the desktop VM becomes active, partial migration is performed again so that the VM can locally access the rest of the memory in the desktop PC. The former is one type of split migration, while the latter is one type of merge migration.

S-memV [6], [7] is the first system for enabling split migration. The first paper [6] proposes the concept of not only split migration but also merge migration and subst migration. Split migration has been implemented and evaluated thoroughly in the following paper [7], but merge migration and subst migration are first implemented in this paper. We clarified several technical challenges to implement these migration methods and achieved correct and efficient migration. In addition, this paper generalizes the original concept of merge migration so that only several hosts running part of a split-memory VM are consolidated into one host. Also, it generalizes the original concept of split migration so that a VM fragment in one host can be further divided into multiple hosts. As more complicated partial migration, this paper proposes split/merge migration. These extended concepts have not been yet implemented, but our implementation of basic merge migration and subst migration can be extended easily.

7. Conclusion

This paper proposed flexible and efficient partial migration of split-memory VMs. The partial migration is classified into four types: subst, merge, split, and split/merge migration. We have implemented a system named IPmigrate for the first two migration methods. Subst migration substitutes one of the hosts running a split-memory VM and merge migration consolidates VM fragments in multiple hosts into one host. Even if a split-memory VM causes remote paging between hosts during partial migration, IPmigrate preserves consistency by memory retransfers and invalidation and then transfers the memory of the VM without excess and deficiency. From our experiments, we showed the efficiency of partial migration in IPmigrate.

One of our future work is to improve the implementation of merge migration. In the current implementation, sub-hosts have to wait for the completion of the migration for the main host. It is desirable to finish the migration of a sub-host soon after the sub-host completes the memory transfer. Another direction is to implement various types of partial migration, e.g., merge migration only for several hosts and split/merge migration. We believe that all types of partial migration can be implemented by extending subst migration and merge migration for all the hosts.

Acknowledgements

The research results have been achieved by the "Resilient Edge Cloud Designed Network (19304)," the Commissioned Research of National Institute of Information and Communications Technology (NICT), Japan.

References

[1] Amazon Web Services, Inc. Amazon EC2 High Memory Instances. https://aws.amazon.com/ec2/instance-types/high-memory/, 2019.

[2] SAP SE. What is SAP HANA? An Unrivaled Data Platform for the Digital Age. https://www.sap.com/products/hana.html.

[3] Microsoft Corporation. SQL Server 2017 on Windows and Linux. https://www.microsoft.com/en-us/sql-server/sql-server-2017.

[4] Apache Software Foundation. Apache Spark – Lightning-Fast Cluster Computing. http://spark.apache.org/.

[5] Facebook, Inc. Presto: Distributed SQL Query Engine for Big Data. https://prestodb.io/.

[6] M. Suetake, H. Kizu, and K. Kourai. Split Migration of Large Memory Virtual Machines. In *Proc. ACM SIGOPS Asia-Pacific Workshop of Systems*, 2016.

[7] M. Suetake, T. Kashiwagi, H. Kizu, and K. Kourai. S-memV: Split Migration of Large-memory Virtual Machines in IaaS Clouds. In *Proc. IEEE Int. Conf. Cloud Computing*, pages 285–293, 2018.

[8] S. Liang, R. Noronha, and D. Panda. Swapping to Remote Memory over InfiniBand: An Approach using a High Performance Network Block Device. In *Proc. IEEE Cluster Computing*, 2005.

[9] U. Deshpande, B. Wang, S. Haque, M. Hines, and K. Gopalan. MemX: Virtualization of Cluster-Wide Memory. In *Proc. Int. Conf. Parallel Processing*, pages 663–672, 2010.

[10] J. Gu, Y. Lee, Y. Zhang, M. Chowdhury, and K. Shin. Efficient Memory Disaggregation with Infiniswap. In *Proc. USENIX Symp. Networked Systems Design and Implementation*, 2017.

[11] E. Amaro, C. Branner-Augmon, Z. Luo, A. Ousterhout, M. Aguilera, A. Panda, S. Ratnasamy, and S. Shenker. Can Far Memory Improve Job Throughput? In *Proc. European Conf. Computer Systems*, 2020.

[12] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori. kvm: the Linux Virtual Machine Monitor. In *Proc. Ottawa Linux Symp.*, pages 225–230, 2007.

[13] A. Lagar-Cavilla, J. Ahn, S. Souhlal, N. Agarwal, R. Burny, S. Butt, J. Chang, A. Chaugule, N. Deng, J. Shahid, G. Thelen, K. Yurtsever, Y. Zhao, and P. Ranganathan. Software-Defined Far Memory in Warehouse-Scale Computers. In *Proc. Int. Conf. Architectural Support for Programming Languages and Operating Systems*, pages 317–330, 2019.

[14] B. Fitzpatrick. memcached – A Distributed Memory Object Caching System. http://memcached.org/.

[15] B. Aker. memaslap – Load Testing and Benchmarking a Server. http://docs.libmemcached.org/bin/memaslap.html.

[16] M. R. Hines and K. Gopalan. Post-Copy Based Live Virtual Machine Migration Using Adaptive Pre-Paging and Dynamic Self-Ballooning. In *Proc. Int. Conf. Virtual Execution Environments*, pages 51–60, 2009.

[17] U. Deshpande, Y. You, D. Chan, N. Bila, and K. Gopalan. Fast Server Deprovisioning through Scatter-Gather Live Migration of Virtual Machines. In *Proc. Int. Conf. Cloud Computing*, pages 376–383, 2014.

[18] I. Banerjee, P. Moltmann, K. Tati, and R. Venkatasubramanian. VMware ESX Memory Resource Management: Swap. *VMware Technical J.*, 3(1), 2014.

[19] U. Deshpande, D. Chan, T. Guh, J. Edouard, K. Gopalan, and N. Bila. Agile Live Migration of Virtual Machines. In *Proc. IEEE Int. Parallel and Distributed Processing Symp.*, 2016.

[20] N. Bila, E. Lara, K. Joshi, H. Lagar-Cavilla, M. Hiltunen, and M. Satyanarayanan. Jettison: Efficient Idle Desktop Consolidation with Partial VM Migration. In *Proc. ACM European Conf. Computer Systems*, pages 211–224, 2012.