Split Migration of Large Memory Virtual Machines

Masato Suetake
Kyushu Institute of Technology
masato@ksl.ci.kyutech.ac.jp

Hazuki Kizu
Kyushu Institute of Technology
hazuki@ksl.ci.kyutech.ac.jp

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

Abstract

Recently, Infrastructure-as-a-Service clouds provide VMs with a large amount of memory, e.g., X1 instances with 2 TB in Amazon EC2. Such large memory VMs make VM migration difficult because VM migration needs sufficient free memory at the destination host. Even in clouds, it is costly to always reserve hosts with a large amount of free memory. This paper proposes *S-memV*, which enables *split migration* of large memory VMs. Split migration migrates a large memory VM to multiple hosts by dividing its memory. It transfers core information and frequently accessed memory of a VM to the main host, whereas it transfers infrequently accessed memory to the sub-hosts. When the VM requires the memory stored in the sub-hosts, S-memV performs *remote paging* between the main host and the sub-hosts. Since split migration is aware of remote paging, S-memV can keep the performance of migrated VMs. In addition to such *one-to-N migration*, split migration supports *N-to-one* and *partial migration*. We have implemented S-memV in KVM and showed that migration time was much shorter than that of the traditional migration with virtual memory.

1. Introduction

Infrastructure-as-a-Service (IaaS) clouds provide virtual machines (VMs) to users. Many VMs are consolidated into a small number of hosts to reduce costs. Recently, as the needs to IaaS clouds are diversified, IaaS clouds also provide VMs with a large amount of memory. In Amazon EC2, for example, each 8xlarge instance has 244 GiB and new X1 instances have 2 TB. Such large memory VMs are required for big data analysis, e.g., using Apache Spark [2] and Facebook Presto [7], because big data can be analyzed faster by maintaining data in memory as much as possible. Fast in-memory databases such as SAP HANA [15] and Microsoft SQL Server [14] are another application of a large amount of memory.

There are two issues to migrate such large memory VMs. One is the migration time because that time is basically proportional to the memory size of a migrated VM. This issue has been resolved by parallelizing VM migration [16] and using fast interconnects such as 40 Gigabit Ethernet (GbE) [13]. The other unresolved issue is the availability of the destination host. VM migration needs sufficient free memory at the destination host. However, it is costly to always reserve hosts with a large amount of free memory, even if possible in clouds. If large memory VMs cannot be migrated, they have to be stopped during host maintenance and big data analysis is disrupted for a long time. In addition, the whole data in memory is lost and it takes much time to restore the lost data in memory by reading disks or redoing computation. This largely degrades performance for a long time after VMs are restarted.

On the other hand, there are many hosts with a small amount of free memory in clouds. The total amount is often sufficient for accommodating a migrated large memory VM. To integrate such fragments of free memory into one, virtual memory with remote paging [4, 5, 8, 12] has been proposed. Traditional virtual memory enables the system to use a larger amount of memory than physical memory by paging out part of the memory to disks. Instead of local disks, remote paging can use free memory in other hosts to store paged-out memory. However, using virtual memory largely degrades the migration and execution performance of large memory VMs because virtual memory is *incompatible* with VM migration and excessive paging is caused during and after VM migration. At worst, VM migration is not completed endlessly due to thrashing.

To solve this problem, this paper proposes *S-memV* for *split migration* of large memory VMs. Split migration enables a large memory VM to be migrated to multiple hosts by dividing its memory. In split migration, the destination of VM migration is not always one host as traditional migration and consists of one *main host* and zero or more *sub-hosts*. Split migration transfers core information of a VM such as CPU and device states to the main host at the destination. It also transfers memory that is likely to be accessed

APSys 2016, August 4–5, 2016, Hong Kong, China..

http://dx.doi.org/10.1145/2967360.2967368

frequently after VM migration to the main host as much as possible. It transfers memory that cannot be stored in the main host to the sub-hosts. After split migration, the VM runs at the main host and S-memV performs remote paging between the main host and the sub-hosts when the VM needs memory in the sub-hosts. Since remote paging does not occur at all during split migration, S-memV can achieve fast VM migration. Thanks to the awareness of memory access patterns, S-memV can keep the performance of VMs after split migration. Split migration supports not only such *one-to-N migration* but also *N-to-one migration* and *partial migration*, which migrates part of a VM running across multiple hosts.

We have implemented S-memV in KVM and developed a memory server that manages part of the memory of a VM at a sub-host. We have developed a remote paging system using the userfaultfd mechanism in Linux 4.3. In addition, we have developed a mechanism for collecting memory access data of VMs using the extended page tables (EPT). According to our experiments, split migration could achieve less migration time and downtime than the traditional VM migration with virtual memory. In particular, it could suppress the increase in migration time under a memory-intensive workload.

This paper is organized as follows. Section 2 describes an issue in migrating VMs with a large amount of memory. Section 3 proposes S-memV for split migration and Section 4 describes its implementation. Section 5 shows experimental results of one-to-N migration in S-memV. Section 6 describes related work and Section 7 concludes this paper.

2. Migration of Large Memory VMs

VM migration enables a running VM to be moved to another host without stopping it. Using VM migration, administrators can maintain a host without service disruption after they migrate all the VMs running at that host. Basic VM migration first creates a new VM at the destination host and copies the memory contents of a VM running at the source host to the memory of the newly created VM via the network. Then it re-transfers updated memory contents because the memory of the VM at the source host continues to be modified during the memory transfer. VM migration repeats this re-transfer and stops a VM at the source host when the amount of memory to be transferred is small enough. Finally, it transfers CPU and device states and memory contents that are not synchronized yet.

Recently, large memory VMs are being widely used. For example, Amazon EC2 provides several 8xlarge instance types with 244 GiB of memory. Recently, it added the x1.32xlarge instance type with 1,952 GiB of memory. Such VMs are used to process a large amount of data, e.g., big data analysis [2, 7] and in-memory database [14, 15]. However, large memory VMs make VM migration difficult because it is not cost-efficient to always reserve hosts with a large amount of free memory as the destination of VM migration. If there is such a host but the host is used for running many small VMs, administrators have to first migrate these VMs to obtain necessary free memory. This is a time-consuming task and increases the time until the migration of a large memory VM is completed.

2.1 VM Migration with Virtual Memory

When there is not sufficient free memory at the destination host for VM migration, the virtual memory technology is traditionally used. Virtual memory enables the system to use a larger amount of memory than physical memory by paging out the memory pages that cannot be stored in physical memory to disks. However, virtual memory is *incompatible* with VM migration of a large memory VM. Since all the memory pages of a VM are transferred in order in the first iteration of VM migration, memory pages with lower addresses are unconditionally paged out using the least recently used (LRU) algorithm, regardless of memory access patterns inside the VM. This can result in degrading the performance of virtual memory after VM migration.

In the following iterations of VM migration, paging also occurs frequently. When memory pages are re-transferred due to memory updates, those that are not resident in physical memory are first paged in from disks and then overwritten. The number of such updated pages can be large for a large memory VM because it takes a long time to transfer all the memory pages in the first iteration. When VM migration is completed, frequently updated pages are likely to reside in physical memory. However, frequently accessed *read-only* pages are likely to be paged out because VM migration does not re-transfer memory pages that are not updated. Therefore, page-ins/-outs frequently occur between physical memory and slow disks after VM migration, leading to performance degradation. Using SSDs instead of HDDs remedies this problem, but even SSDs are still two orders of magnitude slower than memory.

Such excessive paging could be alleviated if the source host first transfers infrequently accessed pages and then frequently accessed ones to the destination host. Since frequently modified pages reside in the physical memory at the destination host at the end of the first iteration, they can be updated without paging in the following iterations. However, many page-outs still occur for infrequently accessed pages, which are transferred earlier. In addition, page-ins also occur if the memory access pattern changes while a large memory VM is being migrated.

To reduce the overhead of paging with disks, remote paging [4, 5, 8, 12] has been proposed. It pages in/out memory pages from/to the memory at other hosts via the network, instead of local disks. If the network is fast enough, remote paging is faster than paging with slow disks. However, remote paging is also incompatible with VM migration. While VM migration transfers all the memory pages of a VM to the destination host, the destination host has to transfer paged-

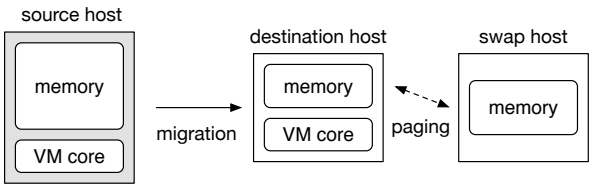


Figure 1. VM migration using remote paging.

out pages to swap hosts, as illustrated in Fig. 1. Therefore, the network bandwidth between the destination and swap hosts is consumed. In addition, the system load at the destination host increases due to remote paging and also affects the migration performance.

2.2 Post-copy Migration with Virtual Memory

Unlike the above pre-copy migration, post-copy VM migration [10] can reduce the frequency of paging. Post-copy migration first transfers only core information that is necessary for VM execution and immediately switches the execution to a VM newly created at the destination host. To transfer the memory of the VM, the on-demand transfer is basically used. When the VM running at the destination host requires memory pages, the source host transfers them. Since the destination host can selectively page out infrequently accessed pages at this time, compulsory paging as in pre-copy migration can be avoided. However, the on-demand transfer increases the delay of memory access by a VM.

Therefore, the background transfer is usually used in combination with the on-demand transfer. It transfers memory pages in background when no memory access is done by a VM. Like pre-copy migration, memory pages are transferred regardless of memory access patterns inside the VM. As a result, even frequently accessed memory pages can be paged out at the destination host if those pages are selected using the LRU algorithm.

3. Split Migration

This paper proposes S-memV for enabling split migration of large memory VMs. Split migration divides a large amount of memory of a VM into smaller pieces and directly migrates them to multiple hosts. Unlike traditional VM migration, split migration is aware of remote paging performed after VM migration.

3.1 One-to-N Migration

Split migration enables a large memory VM to be migrated from one host to multiple hosts. Multiple hosts at the destination consist of one main host and zero or more sub-hosts, as illustrated in Fig. 2. Split migration transfers core information such as CPU and device states to the main host. It also transfers memory pages that the VM is likely to access frequently to the main host as much as possible so that the VM can access them without remote paging after VM migration. On the other hand, split migration transfers memory pages that cannot be accommodated in the main host to the sub-hosts. To divide the memory of the VM appropriately, S-memV monitors the memory access pattern inside the VM and predicts future access. When migrating the VM, S-memV chooses appropriate destination hosts in a cloud, considering free memory of each host. To migrate a large memory VM faster, split migration transfers memory pages to multiple hosts in parallel.

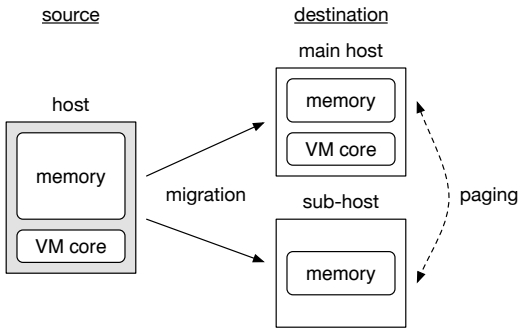


Figure 2. One-to-N migration in S-memV.

After split migration, the VM runs at the main host. When a memory page accessed by the VM does not exist in the main host, S-memV performs remote paging. It swaps the requested memory page at one of the sub-hosts with a page infrequently used at the main host. On the other hand, S-memV does not cause remote paging at all *during* split migration. While a VM is being migrated, memory pages that cannot be accommodated in the main host are not paged out to the sub-hosts via the main host. Instead, they are *directly* transferred to the sub-hosts. Therefore, there is no wasteful network transfer between the main host and the sub-hosts at both the first memory transfer and the memory re-transfers of VM migration. This enables fast migration of large memory VMs. In addition, since memory pages that are likely to be frequently accessed are stored in the memory of the main host, it is expected that the frequency of remote paging is too low just after split migration.

Split migration can be applied to not only pre-copy migration but also post-copy migration. At the on-demand transfer, it transfers requested memory pages from the source host to the destination main host. If there is no free memory at the main host, S-memV pages out infrequently accessed pages to the sub-hosts. At the background transfer, like pre-copy migration, split migration transfers frequently accessed memory pages from the source host to the destination main host, whereas it transfers the other pages to the destination sub-hosts. In either case, once split migration has transferred memory pages to the destination, S-memV swaps them between the destination main host and sub-hosts.

3.2 N-to-One Migration

Split migration can migrate a large memory VM whose memory is divided into multiple hosts to one host again.

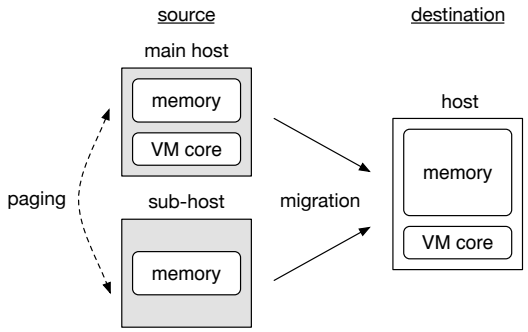


Figure 3. N-to-one migration in S-memV.

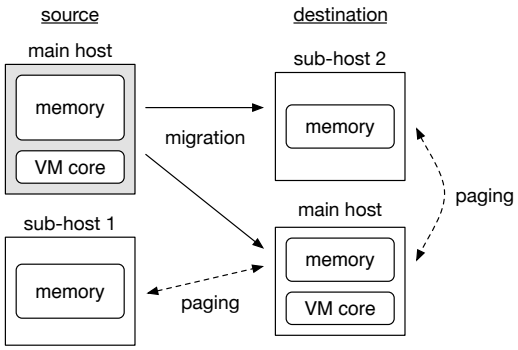


Figure 4. Partial migration in S-memV.

This N-to-one migration is used after the maintenance of the originally used host is completed or when another host with sufficient free memory is prepared. Split migration transfers memory pages at both the source main host and sub-hosts to the destination host in parallel, as illustrated in Fig. 3. At the main host, it migrates a VM normally except that it does not transfer non-existent memory at that host. At the sub-hosts, it simply transfers the memory of the VM. For memory pages paged in/out during VM migration, split migration transfers them without redundancy or omission. If a memory page is paged in from a sub-host to the main host during VM migration, split migration transfers it to the destination host only when it has not been transferred yet or when it is modified. It does similarly for a memory page paged out from the main host to a sub-host during split migration. In post-copy migration, split migration requests memory pages of either the source main host or sub-hosts at the on-demand transfer.

3.3 Partial Migration

Split migration can migrate part of a large memory VM or the entire VM running across multiple hosts to a different host group. This partial migration is used when some or all of the hosts running the VM need to be maintained. Fig. 4 shows an example of split migration of only the source main host to the destination main host and sub-host. After this migration, the VM is across the destination main host and sub-host and the source sub-host. When migrating part of a VM running at the main host, split migration migrates the VM to the destination main host in a manner similar to one-to-N migration. If some of memory pages cannot be accommodated at the destination main host, split migration transfers them to existing sub-hosts or newly allocated sub-hosts. When migrating part of a VM at sub-hosts, it transfers memory pages of the VM between sub-hosts.

4. Implementation

We have implemented S-memV in QEMU-KVM 2.4.1 and Linux 4.3. In the current implementation, S-memV supports one-to-N pre-copy VM migration. As illustrated in Fig. 2, the system consists of one source host, one destination main host, and one or more destination sub-hosts. The source host and the destination main host run QEMU-KVM in which S-memV is implemented and run VMs on top of it. Each destination sub-host runs a memory server, which manages part of the memory of VMs.

4.1 Extension to QEMU-KVM

To migrate a VM to multiple hosts, QEMU-KVM at the source host connects to not only QEMU-KVM at the destination main host but also the memory servers at the destination sub-hosts. The sub-hosts are chosen appropriately by a server that manages free memory of all the hosts in a cloud. Existing cloud management systems such as OpenStack already provide such servers for VM placement. In addition, S-memV needs information on network latency between hosts. If it is estimated that network traffic increases due to remote paging, network latency is more critical. Using such information, the source QEMU-KVM divides the memory of a VM for the main host and selected sub-hosts. Once it determines a destination host for each memory page, it always transfers the data of the page to the same host at the re-transfer of modified pages. In the current implementation, the sub-hosts are pre-determined and the memory of a VM is divided by a fixed size.

For a memory page accommodated in the main host, the source QEMU-KVM transfers a pair of the offset to a memory block and the data of the page to the destination QEMU-KVM. The destination QEMU-KVM writes the data to the memory of a newly created VM. For a page accommodated in a sub-host, in contrast, the source QEMU-KVM transfers only the physical memory address, instead of the offset, to the sub-host. This is because sub-hosts do not have information on the memory blocks used by QEMU-KVM. In addition, the source QEMU-KVM also transfers the following information to the main host: the IP address of the destination sub-host and the offset to a memory block. The destination QEMU-KVM maintains received information using a radix tree for managing the memory resident in the sub-hosts. A radix tree is a data structure whose memory utilization is good and is used for the management of the page cache in the Linux kernel.

4.2 Memory Server

A memory server runs at a sub-host and manages part of the memory of VMs in a 4-KB page granularity. For the memory management, it uses a radix tree whose key is a physical memory address in a VM and whose value is the pointer to the data of the corresponding memory page. A memory server handles page-out and -in requests. A page-out request consists of a physical memory address and the data of the corresponding memory page. When a memory server receives a page-out request, it allocates 4-KB memory, copies the received data to it, and registers it to the radix tree. In contrast, a page-in request consists of only a physical memory address. When a memory server receives a page-in request, it searches the radix tree and returns the data of the corresponding memory page. At the same time, it removes the data from the radix tree and releases it.

4.3 Collecting Memory Access Data

To accommodate frequently accessed memory pages in the main host as much as possible, S-memV keeps track of recently used pages inside a VM as approximation. To obtain information on memory access, QEMU-KVM issues the extended ioctl system call to KVM in the Linux kernel. At that time, it allocates a bitmap whose bit corresponds to each memory page and passes it to the system call. Next, KVM traverses the extended page table (EPT) for all the pages of the target VM and obtains page table entries. Since their access bit is set when the corresponding page is accessed, KVM records the value of the access bit to the passed bitmap. Finally, KVM resets the access bit so that CPUs can record new memory access in EPT for the next period.

QEMU-KVM periodically obtains that bitmap and preserves a sequence of such bitmaps for a certain period. It uses the bitmaps to search for recently used pages when migrating the target VM. In contrast, it searches for least recently used pages using the bitmaps when page-out is necessary.

4.4 Remote Paging

To achieve remote paging for a VM at the main host, S-memV uses the userfaultfd system call, which was introduced in Linux 4.3. The memory of a VM is allocated by anonymous memory mapping using the mmap system call. When QEMU-KVM receives data from the source host and writes it to the corresponding memory page, a physical memory page is assigned. For the other pages, physical memory is not assigned yet. To trap access to non-existent memory pages, QEMU-KVM issues the userfaultfd system call at the end of VM migration and obtains a file descriptor. Then it issues the ioctl system call for userfaultfd and registers all the memory pages of the migrated VM to userfaultfd.

Fig. 5 shows how a memory page is paged in/out using userfaultfd. When the VM accesses a non-existent page, a page fault occurs and an event is notified to QEMU-KVM through the file descriptor of userfaultfd. QEMU-KVM

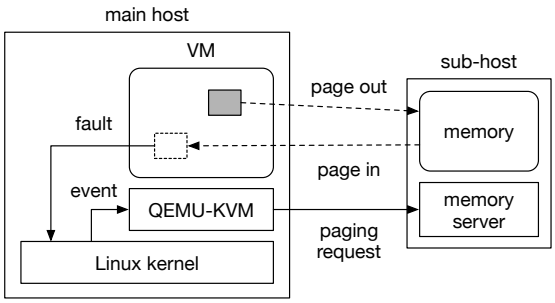


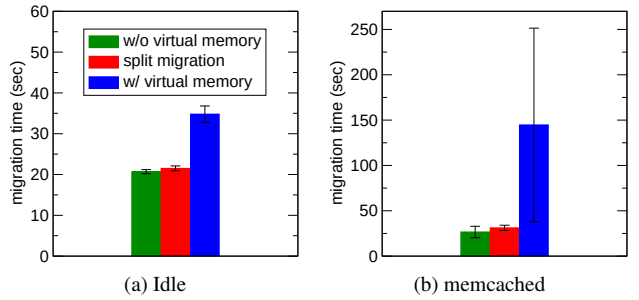
Figure 5. Remote paging using userfaultfd.

translates the notified host memory address into the physical memory address used inside the VM and sends a page-in request to the memory server at the sub-host that manages the corresponding memory page. It finds that sub-host by searching the radix tree for managing the memory resident in the sub-hosts. When QEMU-KVM receives the data of the corresponding page from the sub-host, it writes the data to the faulting page via the ioctl system call for userfaultfd. After that, the VM can access the faulting page. In addition, QEMU-KVM modifies the radix tree so that the paged-in page exists in the main host.

At the same time, QEMU-KVM at the main host pages out one page to balance the amount of memory. First, QEMU-KVM chooses most recently used memory page, using information on the bitmaps obtained as in Section 4.3. Then it sends a page-out request to the memory server at the same sub-host and stores the data of the paged-out page in the sub-host. In addition, QEMU-KVM modifies the radix tree so as to reflect the paged-out page. To evict that page from the VM, QEMU-KVM unmaps the page once and re-allocates that page using anonymous memory mapping. Thus, the physical memory assigned to the page is abandoned.

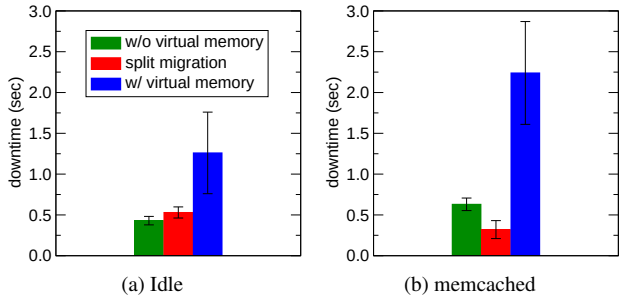
5. Experiments

To show the effectiveness of split migration, we measured the migration performance in S-memV. For comparison, we executed VM migration with and without virtual memory. For the source host, we used a PC with an Intel Xeon E3-1270v3 processor and 16 GB of memory. For the destination hosts, we used a PC with an Intel Xeon E3-1270v2 processor, 2 GB of memory, and 600 GB of HDD as the main host and one with an Intel Xeon E5640 processor and 2 GB of memory as the sub-host. At the main host, the amount of free memory was approximately 1 GB and we configured 16 GB of a swap space. Only when we executed VM migration without virtual memory, we increased the memory of the main host to 4 GB, which was sufficient to accommodate the migrated VM. These PCs are connected with Gigabit Ethernet. For a VM, we assigned one virtual CPU and 2 GB of memory.



(a) Idle (b) memcached

Figure 6. The time for VM migration.



(a) Idle (b) memcached

Figure 7. The downtime of a migrated VM.

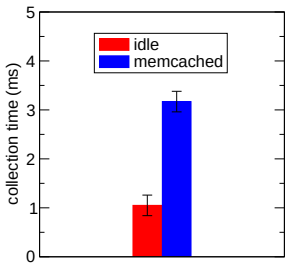


Figure 8. The time for collecting memory access data.

5.1 Migration Time

First, we migrated a VM without explicitly running applications inside it. We configured S-memV so as to transfer 1 GB of memory to the main host and the rest to the sub-host. Since QEMU-KVM optimizes the transfer of zero-filled memory pages, we disabled this optimization because S-memV does not support such optimization yet. The migration time is shown in Fig. 6a. Compared with when the main host had sufficient free memory, VM migration with virtual memory increased the migration time by 87%. For split migration, the increase in migration time was only 17%. This means that split migration could suppress the performance degradation more than VM migration with virtual memory although S-memV does not support parallel transfers to the two destination hosts yet.

Next, we ran in-memory database, memcached [9], in a VM and modified its memory frequently using the memaslap benchmark [1]. We configured the ratio of set and get in memaslap to 0.6:0.4. The migration time is shown in Fig. 6b when we migrated the VM under this workload. Even when the main host had sufficient free memory, the migration time increased by 29%. This is because a large amount of memory was re-transferred due to memory modification inside the VM. Compared with this, the migration time in VM migration with virtual memory became 5.4 times longer because frequent page-ins/-outs occurred due to memory re-transfers. Note that the variance was very large due to the complex behavior of paging. For split migration, in contrast, the performance degradation was 17%. This result shows that split migration could suppress the increase in migration time even under a memory-intensive workload.

From the above result, when a VM has 256 GB of memory, the migration time in split migration is estimated to be more than one hour. However, it can be reduced to 115 seconds using 10 GbE and 21 seconds using 40 GbE [13].

5.2 Downtime

Fig. 7 shows the downtime of a migrated VM under the two workloads in the previous section. When the VM was idle, the downtime in split migration was only 0.1 seconds longer than that in VM migration without virtual memory. When the VM ran memcached, in contrast, the downtime in split migration was shorter. Although the reason is under investigation, it was shown that split migration does not increase the downtime so much. In VM migration with virtual memory, on the other hand, the downtime increased significantly.

5.3 Collection Time of Memory Access Data

We measured the time needed for collecting access data on the entire memory of a VM by traversing its EPT. As shown in Fig. 8, the collection time was only 1 ms when the VM was idle, while the time was 3 ms when memcached frequently accessed the memory of the VM. This is because EPT grew as more memory pages of the VM were accessed. If S-memV collects such data, e.g., every second, the overhead is 0.3%. However, when the memory size of a VM is 2 TB, we would need longer intervals if the collection time became 3 seconds. Fortunately, since EPT shrinks when memory pages of the VM are not accessed, the collection time is expected to be less than that worst estimation.

5.4 VM Performance after Split Migration

Since our implementation of remote paging is still incomplete, we discuss VM performance after one-to-N migration using the results of previous work [12]. Except when VMs are migrated, the behavior of S-memV was almost the same as that of existing remote paging systems. That previous work measured application performance using remote paging over InfiniBand when the sizes of local and remote memory were 512 MB and 1 GB, respectively. For quick sort, the execution time was 1.45 times slower by using remote paging because the working set size was 1 GB and

remote paging occurred frequently. Using Gigabit Ethernet instead of InfiniBand, the execution time was twice slower. For Barnes in the SPLASH-2 suite [17], on the other hand, the performance was almost not degraded because the working set was only slightly larger than the size of local memory. From these previous results, it is expected that S-memV would not largely degrade VM performance as far as the working set is not much larger than the memory size of the main host and does not change.

6. Related Work

Post-copy VM migration [10] switches the execution of a VM to the destination host immediately after it transfers only the minimum states of the VM. While the migrated VM is running at the destination host, the source host transfers the memory of the VM on demand or in the background. This means that post-copy migration can run a VM using two hosts, which is a special case of one-to-N migration in S-memV. However, since this situation is transient, the VM finally runs at only one destination host after VM migration is completed. Consequently, post-copy migration requires two hosts with a large amount of memory unlike S-memV.

Scatter-Gather live migration [6] uses multiple intermediate hosts between the source and destination hosts in post-copy migration. It transfers the memory of a VM to intermediate hosts as fast as possible and reduces the time needed until stopping the source host. The destination host obtains the memory from the intermediate hosts using the on-demand and background transfers in post-copy migration. This is similar to one-to-N migration in S-memV in that the source host transfers the memory of a VM to multiple hosts. However, Scatter-Gather live migration finally transfers the whole memory of a VM to only one destination host. S-memV considers that one destination host cannot accommodate the whole memory of a VM.

Unlike S-memV, MemX [5] runs a VM using the memory of multiple hosts by default. In the MemX-VM mode, the guest operating system in a VM provides a block device to access the memory at the other hosts. This mode allows VM migration without transferring the memory resident in the other hosts, which is one case of partial migration in S-memV. In the MemX-DD mode, Dom0 in Xen provides such a block device, but this mode does not support VM migration because the state of the block device is not transferred at VM migration. In the MemX-VMM mode, MemX provides a VM with the memory extension to access the memory at the other hosts transparently. When a VM accesses a memory page that does not exist at the host, MemX obtains the corresponding page at another host and allocates it to the VM. This mode could support VM migration, but the existing method allows only inefficient N-to-one migration. It has to gather all the memory pages of a VM at the source host and transfer them to the destination host.

Virtual Multiprocessor [11] and vNUMA [3] enable running one large VM with not only the memory but also CPUs of multiple hosts. The VM can transparently access the memory of all the hosts using distributed shared memory. While S-memV uses only one host for running a VM and the other hosts for providing memory, these systems uses all the hosts for running a VM. Therefore the overhead for the cooperation between multiple hosts is much larger. In addition, these systems do not support VM migration.

7. Conclusion

This paper proposed S-memV for split migration, which divides the memory of a large memory VM into small pieces and directly migrates them to multiple hosts. Split migration transfers the memory that cannot be accommodated in the destination main host to the destination sub-hosts. After split migration, the VM runs at the main host and S-memV performs remote paging between the main host and sub-hosts when necessary. However, the frequency of paging is much lower than traditional VM migration with virtual memory. In addition to this one-to-N migration, split migration provides N-to-one and partial migration. We have implemented S-memV in KVM and achieved one-to-N migration. From our experimental results, split migration was promising especially when the memory of a VM was modified frequently during VM migration.

Our future work is as follows. The first is to integrate mechanisms for collecting memory access data of VMs and remote paging into S-memV. Then we need to show that S-memV can successfully reduce the number of memory pages paged in/out after split migration. The second is to evaluate split migration for a larger memory VM using real-world workloads and compare it with previous methods. For better performance of split migration, it is necessary to enable parallel memory transfers to multiple hosts. The third is to support N-to-one and partial migration in S-memV. Unlike one-to-N migration, a new mechanism is needed for synchronizing multiple source hosts at VM migration. The fourth is to investigate how the value of N affects the migration performance. Larger N will reduce the migration time in terms of parallelism, while performance improvement is limited by the number of CPU cores, memory bandwidth, and NICs. The fifth is to recover from failures during split migration. In traditional pre-copy VM migration, it is reasonable to cancel VM migration when only one destination host fails. In split migration, it may be preferred to switch only failed destination hosts to other hosts.

Acknowledgments

We would like to thank the anonymous reviewers and our shepherd Mainak Chaudhuri for their valuable feedback. We also gratefully appreciate Surote Wongpaiboon for helping our implementation. This research was supported in part by the Telecommunications Advancement Foundation.

References

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

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

[3] M. Chapman and G. Heiser. vNUMA: A Virtual Shared-Memory-Multi Processor. In *Proceedings of the 2009 USENIX Annual Technical Conference*, 2009.

[4] D. Comer and J. Griffioen. A New Design for Distributed Systems: The Remote Memory Model. In *Proceedings of the Summer 1990 USENIX Conference*, pages 127–135, 1990.

[5] U. Deshpande, B. Wang, S. Haque, M. Hines, and K. Gopalan. MemX: Virtualization of Cluster-Wide Memory. In *Proceedings of the 39th International Conference on Parallel Processing*, pages 663–672, 2010.

[6] U. Deshpande, Y. You, D. Chan, N. Bila, and K. Gopalan. Fast Server Deprovisioning through Scatter-Gather Live Migration of Virtual Machines. In *Proceedings of the 7th IEEE International Conference on Cloud Computing*, pages 376–383, 2014.

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

[8] M. J. Feeley, W. E. Morgan, E. P. Pighin, A. R. Karlin, H. M. Levy, and C. A. Thekkath. Implementing Global Memory Management in a Workstation Cluster. In *Proceedings of the 15th ACM Symposium on Operating Systems Principles*, pages 201–212, 1995.

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

[10] M. R. Hines and K. Gopalan. Post-Copy Based Live Virtual Machine Migration Using Adaptive Pre-Paging and Dynamic Self-Ballooning. In *Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, pages 51–60, 2009.

[11] K. Kaneda, Y. Oyama, and A. Yonezawa. A Virtual Machine Monitor for Providing a Single System Image. `http://web.yl.is.s.u-tokyo.ac.jp/~kaneda/dvm/`, 2004.

[12] S. Liang, R. Noronha, and D. K. Panda. Swapping to Remote Memory over InfiniBand: An Approach Using a High Performance Network Block Device. In *2005 IEEE International Conference on Cluster Computing*, 2005.

[13] Mellanox Technologies. Accelerating Virtual Machine Migration over vSphere vMotion and Mellanox End-to-End 40GbE Interconnect Solutions. `http://www.mellanox.com/related-docs/solutions/SB_Accelerating_Virtual_Machine_Migration.pdf`, 2016.

[14] Microsoft Corporation. SQL Server 2014. `https://www.microsoft.com/en/server-cloud/products/sql-server/`.

[15] SAP SE. SAP HANA. `https://hana.sap.com/`.

[16] X. Song, J. Shi, R. Liu, J. Yang, and H. Chen. Parallelizing Live Migration of Virtual Machines. In *Proceedings of the 9th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, pages 85–96, 2013.

[17] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta. The SPLASH-2 Programs: Characterization and Methodological Considerations. In *Proceedings of the 22th Annual International Symposium on Computer Architecture*, pages 24–36, 1995.