Efficient Migration of Large-Memory VMs Using Private Virtual Memory

Efficient Migration of Large-memory VMs Using Private Virtual Memory

Yuji Muraoka and Kenichi Kourai

Abstract Recently, Infrastructure-as-a-Service clouds provide virtual machines (VMs) with a large amount of memory. Such large-memory VMs can be migrated to other hosts on host maintenance, but it is costly to always preserve hosts with sufficient free memory as the destination of VM migration. Using virtual memory in destination hosts is a possible solution, but the performance of VM migration largely degrades because traditional general-purpose virtual memory causes frequent paging during the migration. This paper proposes *VMemDirect*, which achieves efficient migration of large-memory VMs using *private virtual memory*. VMemDirect creates private swap space for each VM on fast NVMe SSDs. Then it transfers likely accessed memory data to physical memory and the other data to the private swap space *directly*. This *direct memory transfer* can completely avoid paging during VM migration. We have implemented VMemDirect in KVM and showed that the performance of VM migration and the migrated VM was improved dramatically.

1 Introduction

As Infrastructure-as-a-Service (IaaS) clouds are widely used, they also provide VMs with a large amount of memory. For example, Amazon EC2 provides VMs with 12 TB of memory and plans those with 24 TB of memory in 2019. Such large-memory VMs are used for big data processing. When a host running a VM is maintained, the execution of the VM can be continued by migrating the VM to another host in advance. VM migration transfers the state of a VM, e.g., virtual CPUs and memory

Yuji Muraoka
Kyushu Institute of Technology, 680-4 Kawazu, Iizuka, Fukuoka, Japan, e-mail: murayu@ksl.ci.kyutech.ac.jp

Kenichi Kourai
Kyushu Institute of Technology, 680-4 Kawazu, Iizuka, Fukuoka, Japan, e-mail: kourai@ksl.ci.kyutech.ac.jp

via the network and restarts the VM at the destination host. If the memory of the VM is updated during the transfer, VM migration retransfers the updated memory data. As such, VM migration requires free memory that can accommodate the entire memory of a VM in a destination host. However, it is costly to always preserve such hosts as the destination of VM migration, particularly, for large-memory VMs. If VM migration is not possible, VMs have to be stopped during host maintenance.

One possible solution is to use virtual memory in a destination host. Virtual memory enables part of the memory of a VM to be stored in secondary storage and performs paging. When a VM requires memory data in the storage, the data is paged in from the storage to physical memory. In exchange for this, unnecessary memory data in physical memory is paged out to the storage. However, traditional general-purpose virtual memory is incompatible with VM migration. Since frequent paging occurs during VM migration, the performance of VM migration degrades largely. After the migration, the execution performance of the VM is also largely affected by paging because necessary memory data is often paged out.

This paper proposes *VMemDirect* for efficient migration of large-memory VMs using *private virtual memory*. VMemDirect creates *private swap space* for each VM on fast NVMe SSDs, which are becoming rapidly inexpensive, and integrates private virtual memory with VM migration. It directly transfers memory data of a VM to either physical memory or private swap space in a destination host. Since this *direct memory transfer* does not cause any paging during VM migration, the performance degradation can be avoided. VMemDirect also predicts future memory access of a VM and locates likely accessed memory data in physical memory as much as possible. Therefore, the execution performance of the VM can be preserved after the migration.

We have implemented VMemDirect in KVM to achieve efficient VM migration using private virtual memory. Private swap space is created using a special file called a *sparse file* to enable direct memory transfer to the swap space. Upon VM migration, VMemDirect determines the destination of memory data according to the memory access history of a VM and directly transfers the data to the same location even on retransfers. After VM migration, it performs paging using the userfaultfd mechanism in Linux. Our experimental results show that VMemDirect could reduce the migration time and the downtime and improve the performance of the migrated VM dramatically.

The organization of this paper is as follows. Sect. 2 describes issues on VM migration using traditional virtual memory. Sect. 3 proposes VMemDirect for efficient VM migration using private virtual memory and Sect. 4 explains its implementation. Sect. 5 shows experimental results. Sect. 6 discusses related work and Sect. 7 concludes this paper.

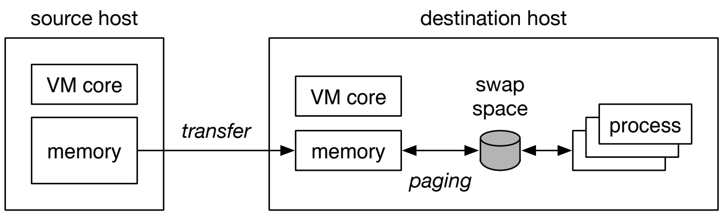


Fig. 1 VM migration using virtual memory.

2 Migration of Large-memory VMs

Even if a destination host does not have sufficient free memory, VM migration is possible by using virtual memory, as illustrated in Fig. 1. Using virtual memory, VM migration can transparently store part of the memory of a VM in swap space on secondary storage and use a necessary amount of memory. When the VM requires memory data in the swap space, a page-in is performed and the data is moved to physical memory. In exchange for this, a page-out is performed and memory data unlikely accessed in physical memory is moved to the swap space. Since the performance of storage is much lower than that of memory, the execution performance of the VM degrades. Fortunately, using recent NVMe SSDs can reduce this overhead.

However, traditional general-purpose virtual memory is incompatible with VM migration. VM migration first transfers the entire memory data of a VM to physical memory in the destination host. After the physical memory becomes full, the following transfers always cause page-outs from physical memory to swap space because transferred memory data has to be first stored in physical memory. This large number of page-outs lead to the increase in migration time. Also, retransfers of updated memory data cause paging. If the memory data to be updated exists in swap space, VM migration has to first page in that memory data to physical memory and then update it. At the same time, page-outs are necessary because physical memory is already full. For large-memory VMs, the first memory transfers take a long time and therefore the amount of memory data updated during the transfer increases.

In the final phase of VM migration, the downtime of the VM increases if paging occurs frequently. This is because the final phase stops the VM and transfers the rest of the state consistently. In particular, if the memory of virtualization software is managed by the same virtual memory as the memory of VMs, as in KVM, it is often paged out during VM migration. At the destination host, virtual devices provided by virtualization software are not used until the final phase of VM migration. Therefore, the memory for them is unlikely accessed data, which is a target of page-outs. When the state of virtual devices is restored, page-ins occur frequently. After VM migration, frequently updated memory data is often stored in physical memory. However, memory data that are just read frequently can be stored in swap space. Since such data is transferred only once, it is often paged out by following memory transfers. This can lead to frequent paging after the VM is resumed at the destination host.

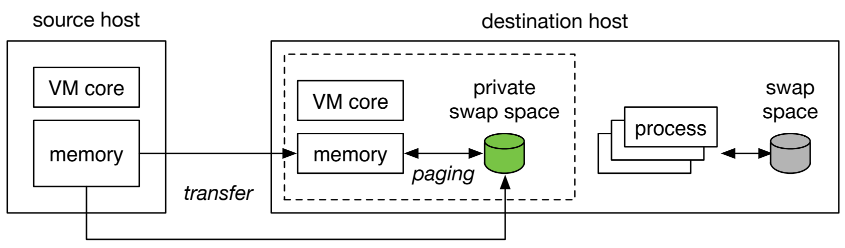


Fig. 2 VM migration using private virtual memory in VMemDirect.

To counteract these problems, split migration [9, 8] has been proposed using multiple destination hosts. Split migration divides the memory of a VM and transfers the memory fragments to multiple smaller hosts. It transfers the state of virtual CPUs and devices and likely accessed memory data to a main host and the other memory data to sub-hosts. If the VM requires memory data in sub-hosts after VM migration, the data is transferred to the main host by remote paging. Since no paging occurs during VM migration, split migration can improve the migration performance. Also, it can increase the execution performance of the migrated VM because necessary memory data is transferred to the main host in advance. However, split migration is more costly than VM migration using virtual memory. It requires small but multiple sub-hosts and, for efficient remote paging, high-speed network such as InfiniBand [6, 5]. In addition, migrated VMs are subject to host and network failures.

3 VMemDirect

For efficient migration of large-memory VMs, this paper proposes *VMemDirect* using *private virtual memory* and the technology of split migration. Fig. 2 illustrates VM migration in VMemDirect. VMemDirect creates *private swap space* for each VM on fast NVMe SSDs in a destination host and integrates private virtual memory with VM migration. Since recent NVMe SSDs are becoming more inexpensive, the cost can be lower than using multiple sub-hosts and expensive high-speed network as in split migration. In addition, the execution of VMs are not affected by host or network failures after VM migration.

Since private virtual memory targets only the memory of a VM, VMemDirect can prevent performance degradation due to paging of the memory of virtualization software itself. During memory transfers in VM migration, the memory of virtualization software is not paged out at the destination host. When a VM is created, VMemDirect creates private swap space on an NVMe SSD with the same size as the memory of the VM. This swap space is optimized for the memory of the VM. Memory areas in the VM correspond to blocks in the swap space. Memory data of the VM is stored in blocks in the swap space only when it does not exist in physical memory. The other blocks do not have actual data in the swap space.

Upon VM migration, VMemDirect directly transfers memory data to either physical memory or private swap space, instead of relying on paging of traditional virtual memory. At the source host, VMemDirect determines locations where each memory data is stored at the destination host when it starts VM migration. The locations are not changed during VM migration and VMemDirect retransfers memory data to the same location. This *direct memory transfer* can prevent paging from occurring during VM migration. No data in physical memory is paged out, while no memory data in private swap space is paged in. When VMemDirect retransfers memory data, it directly updates data in physical memory or private swap space. Since the structure of private swap space is designed for this direct access, it is easy to find the corresponding blocks.

The locations of memory data in the destination host are determined on the basis of the memory access history of a VM. VMemDirect transfers likely accessed memory data to physical memory and the other data to private swap space. This increases the probability that retransferred memory data is stored in physical memory. As a result, the overhead due to overwriting data in private swap space can be reduced. Unlike VM migration using traditional virtual memory, not only frequently updated memory data but also frequently read data is stored in physical memory. Therefore, the occurrence of paging can be suppressed after the VM is resumed.

4 Implementation

We have implemented VMemDirect in KVM using QEMU-KVM 2.4.1 and Linux 4.11. QEMU-KVM is virtualization software that runs on top of Linux.

4.1 Swap File for Private Swap Space

For private swap space, VMemDirect creates a swap file with the same size as the memory of a VM using a *sparse file*. A sparse file can contain blocks that have no actual data, which are called *holes*. Using this swap file, it can make one-to-one relation between memory areas of the VM and blocks of the swap file. Memory data of the VM is stored in either physical memory or the swap file (Fig. 3). When it is paged in to physical memory, VMemDirect makes the corresponding block of the swap file a hole to reduce storage usage. To create a hole, it writes a metadata standing for an empty block to storage.

VMemDirect accesses the swap file using direct I/O so that the page cache is not allocated for the file blocks. Direct I/O enables data to be directly read from and written to storage without storing it in the page cache managed by the operating system. In VMemDirect, most of the physical memory is used to store memory

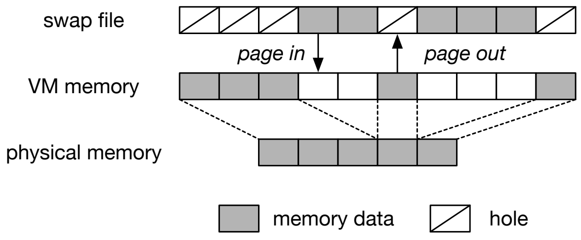


Fig. 3 The memory management of a VM using private swap space.

data of a VM. If the page cache were created for paged-out data, traditional virtual memory would have to page out data in physical memory. To use the bandwidth of NVMe SSDs as much as possible, VMemDirect accesses the swap file by the chunk larger than the 4-KB page. In the current implementation, VMemDirect uses 256-page chunks.

4.2 Direct Memory Transfer

To manage the locations where memory data is transferred on VM migration, VMemDirect creates a *location bitmap*. This bitmap allocates one bit for each 4-KB page and the value is determined according to the memory access history of a VM. The memory access history is obtained as described in the previous work [9, 8]. When memory data is transferred to the swap file, VMemDirect sets 1 to the corresponding bit in the bitmap. Otherwise, it sets 0 to the bitmap. VMemDirect transfers memory data of each page with the value of the corresponding bit. At the destination host, VMemDirect stores received memory data in either physical memory or the swap file according to the specified location.

To write memory data to the swap file by the chunk larger than the page, VMemDirect temporarily stores received memory data in a buffer on physical memory. When the buffer becomes full, VMemDirect writes data to storage at once. Since memory data is transferred sequentially in the first phase of VM migration, VMemDirect can write buffered data to contiguous blocks in the swap file. In contrast, received memory data is not contiguous when memory data is retransferred. For such data, VMemDirect writes data to the swap file by the page.

4.3 Paging for Private Virtual Memory

VMemDirect achieves paging for private virtual memory using the userfaultfd mechanism in Linux. After VM migration, it registers the entire memory of the

VM to userfaultfd. When the VM accesses a non-existent memory page and a page fault occurs, that event is notified to QEMU-KVM by userfaultfd. Next, VMemDirect reads memory data from the block corresponding to the faulting address in the swap file. Then, it writes the data to the corresponding memory page of the VM. In addition, it performs the same operations for the other pages in the same chunk. At the same time, it removes these blocks in the swap file and makes the blocks holes to complete page-ins.

At the same time, VMemDirect selects a chunk including the most unlikely accessed memory pages to perform page-outs. It writes the data of the selected pages to the corresponding blocks in the swap file and removes the mapping of the pages. To obtain the memory data and remove the mapping atomically, we used the extension of userfaultfd, which is developed for remote paging [8]. To manage the locations of memory data for these page-outs, VMemDirect creates a location bitmap in the destination host as well as the source host of VM migration. The bits are set when VMemDirect receives memory data from the source host. When a page-in occurs, the corresponding bit is set to 0. For a page-out, that is set to 1.

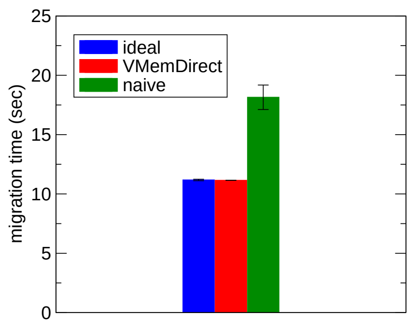
5 Experiments

We conducted several experiments to examine the performance of VM migration and a migrated VM in VMemDirect. For comparison, we examined the performance for *ideal* VM migration with sufficient memory and *naive* VM migration with traditional virtual memory. We used 1 TB of Samsung NVMe SSD 970 PRO as swap space. For the source and destination hosts, we used two PCs with an Intel Xeon E3-1226 v3 processor and 16 GB of memory. These PCs were connected using 10 Gigabit Ethernet. We ran Linux 4.11 as the host operating system and QEMU-KVM 2.4.1 as virtualization software. We used a VM with one virtual CPU and 12 GB of memory. When using traditional virtual memory, we adjusted the size of free memory to 6 GB, which was the half of the memory size of the VM.

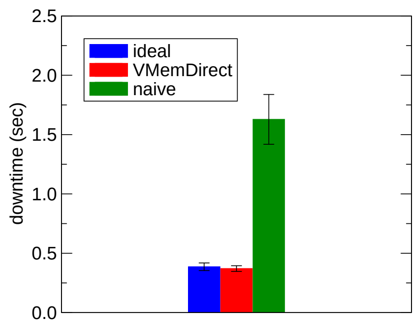
5.1 Migration Performance

To examine migration performance, we first measured the time needed for VM migration. Fig. 4(a) shows the migration time. Compared with the ideal migration, the migration time increased by 1.6 times in the naive migration. For VMemDirect, in contrast, the migration time was almost the same as the ideal migration.

Next, we measured the downtime during VM migration. As shown in Fig. 4(b), the naive migration suffered from 1.2 seconds longer downtime than the ideal migration. The root cause was that many page-ins occurred when virtual devices were resumed in the final phase. In contrast, VMemDirect reduced the downtime by 6 ms. This is because QEMU-KVM could not accurately estimate the downtime in



(a) Migration time



(b) Downtime

Fig. 4 The comparisons of migration performance.

VMemDirect. QEMU-KVM enters the final phase when it is estimated that the rest of the memory data will be transferred in 300 ms. In VMemDirect, the final memory transfers were completed earlier than the estimation because only a smaller amount of memory data was transferred to the slow swap space than during the estimation.

5.2 *Performance of Private Virtual Memory*

To examine the performance of private virtual memory, we ran in-memory database called memcached [4] in a VM and measured the performance using the memaslap benchmark. We allocated 5 GB of memory to memcached and ran the benchmark before and after VM migration. Fig. 5 shows the comparison of the transactions per second after VM migration. VMemDirect achieved 39 times higher performance on average than after the naive migration. Compared with after the ideal migration, the performance in VMemDirect was 11% lower on average in a stable state and largely fluctuated. This is because paging sometimes occurred and the performance degraded by that paging. The reason why the performance was much lower during the first 45 seconds is that paging occurs frequently. This is probably due to the implementation issue of the memory access history.

6 Related Work

vMotion provides two different migration methods in terms of swap space [2]. Unshared-swap vMotion uses different swap spaces between the source and destination hosts and transfers memory data stored in swap space to the destination host. In contrast, shared-swap vMotion stores a swap file in shared storage and transfers no memory data in swap space. To support paging during VM migration, the destination host uses temporary swap space and integrates that swap space into shared one

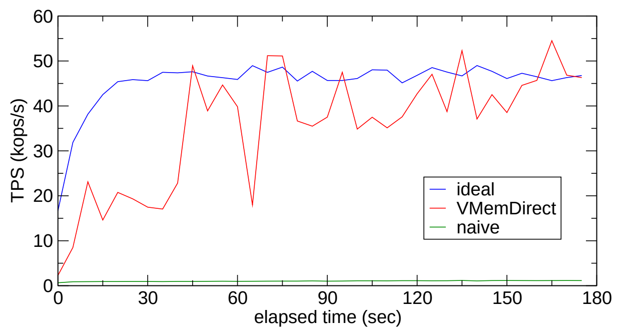


Fig. 5 The memcached performance in a migrated VM.

after the migration. This migration method can be more efficient than VMemDirect, but network paging is always necessary.

Like shared-swap vMotion, Agile live migration [3] locates swap space for each VM in the network. It pages out memory data except for the current working set aggressively. Upon VM migration, it transfers no memory data in the swap space but only data in the working-set memory. This method can further improve migration performance, but the paging overhead is much larger because most of the data is paged out.

FlashVM [7] is virtual memory using paging based on SSDs. It pages out more memory pages at once than when using HDDs. Since random reads of SSDs are fast, FlashVM prefetches more useful pages to reduce page faults. In addition, it adjusts the rate of writeback to SSDs to reduce the latency of page faults. This prefetching technique can improve the performance of private virtual memory in VMemDirect.

Swap space using SSDs and ExpEther has been proposed [10]. ExpEther extends PCI Express using Ethernet and enables SSDs to be used without the limitation of physical locations. High-speed communication using DMA is performed between local memory and remote SSDs. Paging with this swap space can extend computer memory. Using this system, VMemDirect can use remote SSDs as private swap space flexibly.

VSwapper [1] improves the performance of VMs using virtual memory. It monitors storage I/O and prevents unmodified pages from being written to swap space on page-outs. Also, it stores data written to paged-out pages in a temporal buffer and prevents data from being read from swap space if the entire page is written. These optimizations achieve 10 times performance improvement. They can be also applied to private virtual memory in VMemDirect.

7 Conclusion

This paper proposed VMemDirect for achieving efficient VM migration by cooperating with private virtual memory. VMemDirect creates private swap space for each VM on fast NVMe SSDs. It directly transfers likely accessed memory data to physical memory and the other data to the private swap space. After VM migration, VMemDirect performs paging between physical memory and the private swap space for the memory of each VM. Our experimental results show that VMemDirect could improve the performance of VM migration and a migrated VM dramatically, compared with VM migration using traditional virtual memory.

One of our future work is to evaluate the performance of VMemDirect using VMs with various numbers of virtual CPUs, various amounts of memory, and various applications. Another direction is to clarify trade-offs between using high-speed network and NVMe SSDs for VM migration. We will compare various performance of VMemDirect with that of S-memV with split migration and remote paging [9, 8] when we run various applications in VMs. In addition, we are planning to support N-to-one migration [9] in VMemDirect. The original N-to-one migration integrates memory data across multiple hosts into one large host. Similarly, VMemDirect needs to efficiently transfer memory data in both physical memory and private swap space to one host.

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. Amit, N., Tsafrir, D., Schuster, A.: VSwapper: A Memory Swapper for Virtualized Environments. In: Proc. ACM Int. Conf. Architectural Support for Programming Languages and Operating Systems, pp. 349–366 (2014)
2. Banerjee, I., Moltmann, P., Tati, K., Venkatasubramanian, R.: VMware ESX Memory Resource Management: Swap. VMware Technical Journal **3**(1), 48–56 (2014)
3. Deshpande, U., Chan, D., Guh, T., Edouard, J., Gopalan, K., Bila, N.: Agile Live Migration of Virtual Machines. In: Proc. IEEE Int. Parallel and Distributed Processing Symp. (2016)
4. Fitzpatrick, B.: memcached – A Distributed Memory Object Caching System. http://memcached.org/
5. Gu, J., Lee, Y., Zhang, Y., Chowdhury, M., Shin, K.: Efficient Memory Disaggregation with Infiniswap. In: Proc. USENIX Symp. Networked Systems Design and Implementation (2017)
6. Liang, S., Noronha, R., Panda, D.: Swapping to Remote Memory over InfiniBand: An Approach using a High Performance Network Block Device. In: Proc. IEEE Cluster Computing (2005)
7. Saxena, M., Swift, M.: FlashVM: Virtual Memory Management on Flash. In: Proc. USENIX Annual Technical Conf. (2010)
8. Suetake, M., Kashiwagi, T., Kizu, H., Kourai, K.: S-memV: Split Migration of Large-memory Virtual Machines in IaaS Clouds. In: Proc. IEEE Int. Conf. Cloud Computing, pp. 285–293 (2018)
9. Suetake, M., Kizu, H., Kourai, K.: Split Migration of Large Memory Virtual Machines. In: Proc. ACM SIGOPS Asia-Pacific Workshop of Systems (2016)
10. Suzuki, J., Baba, T., Hidaka, Y., Higuchi, J., Kami, N., Uchida, S., Takahashi, M., Sugawara, T., Yoshikawa, T.: Adaptive Memory System over Ethernet. In: Proc. USENIX Workshop on Hot Topics in Storage and File Systems (2010)