Instant Virtual Machine Live Migration

Changyeon Jo$^{[0000-0002-9707-1256]}$, Hyunik Kim$^{[0000-0001-7740-8344]}$, and Bernhard Egger$^{✉[0000-0002-6645-6161]}$

Seoul National University, Republic of Korea
{changyeon,hyunik,bernhard}@csap.snu.ac.kr

Abstract. Live migration of virtual machines (VMs) is an important tool for data center operators to achieve maintenance, power management, and load balancing. The relatively high cost of live migration makes it difficult to employ live migration for rapid load balancing or power management operations, leaving much of its promised benefits unused. The advance of fast network interconnects has led to the development of distributed shared memory (DSM) that allows a cluster of nodes to utilize remote memory with relatively low overhead. In this paper, we explore VM live migration over DSM. We present and evaluate a novel live migration algorithm on our own DSM implementation. The evaluation of live migrating various VMs executing real-life workloads shows that live migration over DSM can reduce the total migration time by 70% and the total amount of transferred data by 65% on average. The absolute average total migration time of only 1.1 seconds demonstrates the potential of live migration over DSM to lead to better load balancing, energy management, and total cost of ownership.

Keywords: Virtualization · Distributed Shared Memory · Live Virtual Machine Migration · Cloud Computing.

1 Introduction

The growing size of datasets in modern memory-intensive workloads presents unique challenges to cloud computing infrastructure and the underlying system software. To process such workloads, Amazon EC2, for example, offers instances with up to 24 TiB of RAM [2]. In this context, two challenges are of particular interest: (1) the total memory footprint of such workloads does not fit into the physical memory of a single machine, and (2) the working set size fluctuates strongly over time. While the first challenge could be addressed by paging to secondary storage, such a system would need to encompass both a large physical memory and abundant fast secondary storage - both measures that render it difficult to efficiently utilize memory resources and that increase the total cost of ownership (TCO) of a data center.

Distributed shared memory (DSM) has long been proposed as a remedy to solve the problem of limited local memory. DSM provides a unified address space across a cluster of physical machines that can be accessed by any machine, whether the accessed memory is local or remote. This allows to allocate more memory for running workloads than is physically available on the host it is running on. DSM also enables elastic use of memory across physical node boundaries which helps to raise the overall memory utilization and lower the TCO of a cluster. Despite the many advantages of deploying DSM in data centers, however, DSM has been deemed impractical in production environments due to its significantly longer latency to access remote memory. The recent advances in fast network interconnects are rapidly closing the gap between

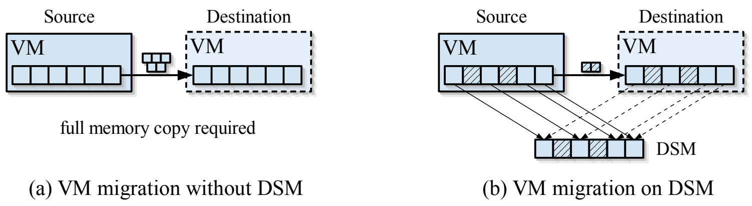

Fig. 1. DSM Live Migration

local and remote memory accesses. For example, an off-the-shelf commodity Mellanox ConnectX-6 network interface card (NIC) achieves sub-microsecond access latencies and 100 Gbit/s bandwidth over remote direct memory access (RDMA). This modern high-performance networking hardware revives the idea of DSM in the era of cloud computing, and recent research shows that DSM can provide on-par performance compared to running applications entirely in local memory [8, 9, 13, 15, 17, 20].

In addition to DSM achieving higher utilization of memory resources, it also enables novel optimizations for virtualized environments. Live migration of a virtual machine, i.e., the act of moving the execution context along with all volatile data of a VM from one physical host to another while the VM is running, not only enables data center operators to perform maintenance on physical machines, but also to balance the load between physical hosts [19]. If a physical machine is running low on resources, some VMs are live-migrated away to avoid a resource overload and provide a better service. On the other hand, consolidation of VMs from lightly-loaded hosts on fewer nodes can help lower the energy consumption and TCO of a data center.

This paper focuses on live migration of VMs in a DSM environment. While traditional approaches copy the entire volatile state of a VM - including its main memory and paged out data on secondary storage - from the source to the destination host, the novel approach presented in this work employs the local memory as a cache for DSM. When migrating a VM, only the local cache of the VM's volatile data needs to be migrated since all other memory is directly accessible through the DSM.

We first present the design and implementation of DSM that is tailored to virtual environments. The DSM uses RDMA to reduce remote memory access latency and runs VMs without a significant performance degradation even under a highly memory-constrained environment. We then discuss a novel VM migration algorithm employing DSM to reduce the total migration time by sending only the contents of the local memory to the destination. An evaluation of VMs running modern memory-intensive applications demonstrates that, on average, the presented approach is able to reduce the total migration time by 70 percent and can migrate large-memory VMs almost instantly.

The contributions of this paper are as follows:

(1) A high-performance DSM for virtualized environments that runs applications without a noticeable performance degradation on DSM.
(2) A novel DSM-based VM live migration algorithm that does not require an entire memory copy.
(3) An implementation and evaluation of the techniques in QEMU/KVM.
(4) Results that show almost instant live migration of VMs running various realistic workloads.

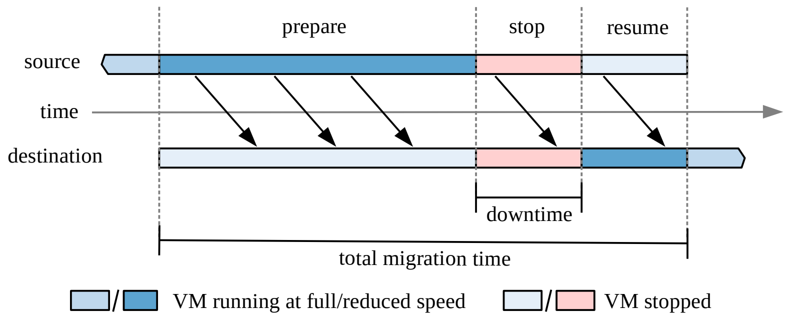

Fig. 2. VM live migration overview.

2 Background of VM Live Migration and DSM

This section briefly introduces VM live migration and distributed shared memory.

2.1 VM Live Migration

Live migration is a technique that moves a running VM from one physical host to another without stopping the execution of the VM. The main action of live migration consists of copying the entire volatile state of the VM, including its memory, from the source machine to the destination host. The memory of a VM can be copied before or after moving the execution state (VCPU register values, device driver states) to the destination host. The VM is only briefly stopped during the transfer of the execution state. Figure 2 shows the different phases of live migration.

Prepare: In this phase, the source starts to copy the volatile state of the VM (mostly the data in the memory) from the source to the destination host. Since the VM keeps running on the source, it may alter memory that has already been transferred to the host, therefore *pre-copy*-based live migration algorithms iteratively copy the modified state to the destination host. Pre-copy enters the *Stop* phase when the amount of dirtied memory falls below a certain threshold. The VM experiences a minimal performance degradation during the *Prepare* phase caused by the tracking of dirtied memory pages.

Stop: In this phase, the VM is stopped on the source, and the states of the VM on the source and the destination host are synchronized. When the synchronization completes, the VM is resumed on the destination host. The length of the *Stop* phase depends on the amount of data to be synchronized, and the minimization of this phase is typically an important optimization goal of live migration algorithms.

Resume: This phase only exists in live migration algorithms that do not transfer the entire state in the *Prepare* and *Stop* phase. During the *Resume* phase, the destination host fetches the remaining volatile state from the source host in the background. Data accessed by the VM that still resides on the source is reactively fetched from the source which typically leads to a heavy performance degradation of the VM. *Post-copy*-based live migration operate in this manner. The advantage of this technique is that it effectively reduces the total amount of data to be transferred by skipping the iterative pre-copy phase; however, accesses to data still residing on the source can severely degrade the performance of the VM.

2.2 Performance Metrics of VM Live Migration

To accommodate large in-memory workloads, cloud service providers have begun to offer VMs with tens to thousands GiBs of memory [2,6,7]. Despite the increased network bandwidth, migrating such large amounts of data not only takes a long time but also consumes a large amount of network bandwidth; optimization of several performance metrics is thus of importance. The major performance metrics of VM live migration are:

- The **total migration time** denotes the time from start to end of the migration.
- The **downtime** represents the duration of the *stop* phase, i.e., period during which the VM is unavailable to the user.
- The **total transferred data** measures the total amount of data transferred from source to destination.
- **Performance degradation** is a measure representing the performance degradation experienced by the VM during live migration such as reduced throughput or increased service latency.

2.3 Distributed Shared Memory

Distributed shared memory (DSM) provides a global shared address space for all machines in the same cluster. DSM provides an easy-to-use communication mechanism for distributed applications. It also enables the execution of applications with a large working set that does not fit into the memory of a single machine.

DSM can be implemented using hardware-based or software-based approaches. In this paper, we focus on software-based DSM. A common practice of software-based approaches is to implement DSM using virtual memory. The virtual-to-physical address translation process enables interception of accesses to locations outside the local memory of a machine and redirection to the DSM. The overhead of remote accesses is minimized by designating a portion of the local memory as a cache for DSM.

The main obstacles of deploying DSM are the comparatively slow remote memory accesses and the high cost of maintaining a coherent view of DSM in the presence of local caches. To overcome the first obstacle, we employ RDMA (Remote Direct Memory Access) to minimize the software overhead of a remote memory access. The second obstacle is avoided by the observation that VMs do not share memory and thus no explicit coherence management is necessary.

2.4 Why DSM for Virtual Machines?

The major cloud service providers now provide large memory VM instances with up to 24 TiB of memory [1,6,7]. DSM is an efficient way to implement such large memory instances. DSM allows the machines of a cluster to share memory and thus helps increasing the overall memory utilization of the cluster. In addition, modern data-intensive workloads exhibit dynamically changing working set sizes, which implies that a relatively small local memory on the server is sufficient to run the application without a significant performance degradation. The live migration cost of large-memory VMs

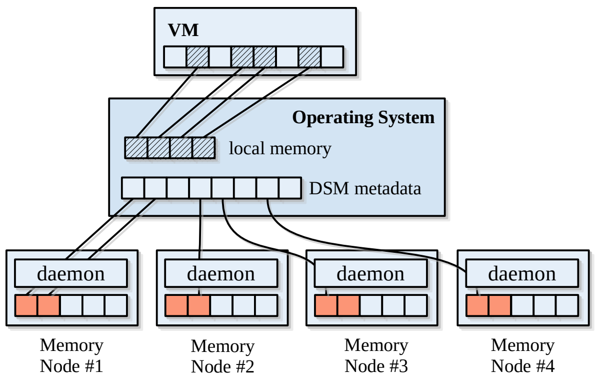

Fig. 3. DSM Design

provides another motivation for deploying DSM in virtualized environments. Large-memory VMs tend to experience significantly higher migration costs and have been reported as a reason for service disruption in production clusters [3, 19]. DSM can reduce the cost of VM migration by significanly reducing the amount of memory to be copied, which makes deploying DSM further attractive. We expect that cloud service providers will start deploying DSM in their datacenters in the near future.

3 DSM Design for Virtualized Environments

This section discusses the design choices of our DSM specialized for virtualized environments and presents a novel live migration approach aimed at exploiting the benefit of our DSM implementation.

3.1 Assumptions on DSM

The presented DSM makes the following assumptions:
- The DSM provides a cluster-wide shared memory address space.
- The DSM relies on the virtual-to-physical address translation mechanism of the host operating system to allow transparent access to remote memory.
- The working set of an application is copied into local memory for direct access and to hide the remote memory access latency through a transparent caching mechanism.
- The DSM does not provide transparent cache coherence. Coherent memory is supported but requires explicit API calls before synchronization points. This greatly simplifies the design of the DSM and achieves far better performance than software-based transparent cache coherence.

Unlike traditional DSM designs, we assume that only one application (a VM) exclusively accesses its DSM-backed memory address space and data sharing is only enabled upon request. These properties are sufficient to execute VMs on DSM and allow VMs to run with little performance degradation. Any DSM implementation that shares the same characteristics can be easily modified to support the presented live migration algorithm in this paper.

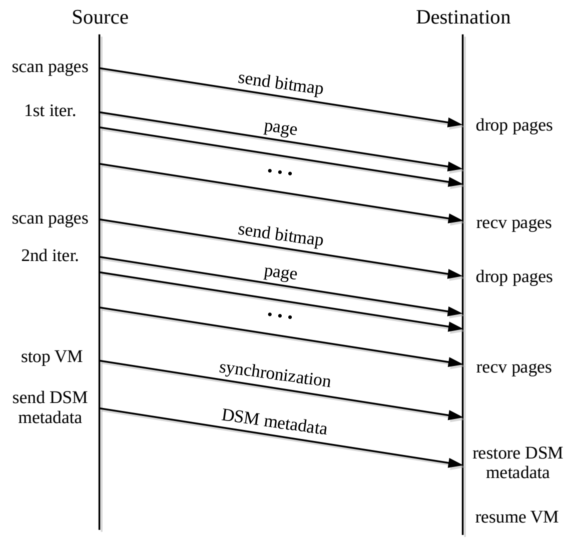

Fig. 4. DSM Live Migration Timeline

3.2 DSM Components

This section briefly introduces the design of the DSM, how to allocate memory to a VM (or an application), and the discusses the components of the DSM.

User interface. The DSM is implemented as a kernel module and exposed to user space as a character device. The virtual machine monitor (VMM) or regular applications use this device to reserve the desired amount of distributed shared memory. The memory is mapped into the virtual address space of the VM by the VMM. To the VM, the entire memory space appears as local physical memory, i.e., the VM is not aware that it is running on top of DSM.

OS service. Our DSM kernel module provides the illusion of a single address space covering both local and remote memory. It transparently manages an area of local memory as a cache for recently accessed DSM. Accesses to remote memory cause a page fault which is handled by the DSM kernel module. The accessed page is fetched from the remote node and brought into local memory. If the local cache is full, a victim page is selected and written back to remote memory to make room for the new page. Another task of the OS service is to balance memory requests across the cluster. The service communicates with the memory nodes and distributes memory allocation requests nodes with low memory pressure.

Memory nodes. All nodes in the cluster act both as providers and consumers of DSM. A consumer node may send several requests to different nodes that then reserve the requested amount in their local memories. A centralized approach with one or separate pool of memory nodes (called as memory disaggregation and gaining popularity in the recent days) is also possible and does not change the semantics of the presented DSM and live migration algorithm as long as all remote memory is accessible from both the source and destination host.

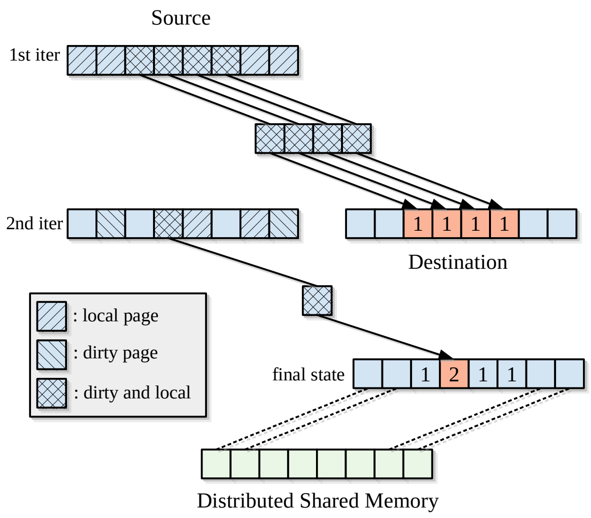

Fig. 5. DSM Live Migration Example

3.3 DSM-Optimized Live Migration Algorithm

In traditional environments, the volatile state of a VM is local to a host and thus needs to be copied to the destination in its entirety to enable live migration. In a DSM environment, however, all memory except the data cached in local nodes is visible from any node in the cluster. This allows us to design a novel live migration approach that eliminates most of the state transfer in the *Prepare* phase. Figure 4 shows a timeline of the events in DSM-optimized live migration. We first explain the necessary steps of the live migration algorithm on the source node.

Step 1: Preparation. The source node notifies the destination node of the imminent migration. The destination host performs the necessary initialization such as starting an empty process for the VM. This step takes only a negligible amount time with respect to the total migration time.

Step 2: Iterative Pre-copy. The first task of the iterative pre-copy algorithm usually encompasses sending the entire memory once to the destination. The presented DSM-optimized algorithm, however, only needs to send the pages that exist in the local cache. Memory residing in a remote node does not need to be transferred since the data can be accessed directly from the destination node. Note that in post-copy mode, this step is executed after the following step 3.

Figure 5 shows an example of iterative pre-copy under DSM. In the first iteration, the VMM marks all pages as dirty in order to transfer the entire memory to the destination. For a VM using DSM, however, only a subset of all pages exists in the local memory. The example shows that four pages with index 2 to 5 are cached in local memory. In the first iteration, all pages are marked dirty. The set of pages to be sent to the destination is given by the intersection of the dirty and the local pages. During the data transfer of the first iteration, the VM keeps running and modifies pages 2, 4, and 8. In the second iteration, again the intersection of all local pages (pages 4, 5, and 7) and dirty pages (pages 2, 4, and 8), that is, page 4, is sent to the destination. Pages 2 and 8 were modified but have already been synchronized with their remote copy, so they do not need to be transferred. In these two iterations, a total of five pages need to be transferred to the destination with our DSM-optimized algorithm. The original pre-copy algorithm would have to send 11 pages (8 in the first, 3 in the second iteration) in total.

Step 3: Synchronization. In the synchronization phase, the VM is stopped, so no pages are dirtied at the source anymore. The dirtied and local pages are sent in a last iteration along with the VM's CPU and device state. In this step, the DSM-optimized algorithm also transfers the DSM metadata to the destination host. The DSM metadata for a VM contains the location of all remote memory pages and is required on the destination node to access the remote data of the VM.

Step 4: Cleanup. In this last step, the VM is terminated on the source host.

Destination node. The tasks of the destination are simpler than those on the source node. Two additional steps are required to correctly complete the DSM-optimized VM migration. At the beginning of every iteration, the destination receives the status of DSM-local pages from the source. The destination computes the intersection of DSM-local pages from the source and DSM-local pages on the destination. Then the destination purges all DSM-local pages that do not belong to the intersection an can thus be restored from remote memory. The purged pages are reused to hold new DSM-local pages sent from the source; this allows us to limit the amount of local memory required for VM live migration on the destination node. Second, in the stop phase, the destination receives the DSM metadata from the source node and maps it into the address space of the VM on the destination host to enable access to its data.

4 Implementation

This section gives some implementation details of the DSM and the modifications to QEMU/KVM required to support the DSM-optimized live migration algorithm.

4.1 Linux Kernel DSM

Our DSM is implemented in a Linux kernel module. The kernel module maps the DSM as a character device node into user-space. DSM-backed memory is obtained by invoking the `mmap` system call on that device node. The returned virtual address range represents a distributed memory region. This virtual address range is not mapped to any memory initially. Accesses to unmapped addresses trigger a pagefault that is handled by our DSM module. The module maps the accessed page to local or remote memory.

Resolving a Pagefault. The early pagefaults are resolved by allocating memory from local memory. We request a 4KB page from the kernel, map the page to the faulted address, and let the application continue. As the VM continues to access more pages in its virtual address space, the size of the locally allocated pages will eventually hit the set limit allowed for caching DSM data. From this point on, the kernel module starts to page out pages that have not been accessed recently to remote memory. To do so, it first finds a node with sufficient free memory to back up the part of the data in the local memory, then reserves the required amount on that node, and finally copies the victim page(s) to the remote node. If a pagefault is to a paged-out address, the kernel fetches the data from the remote node, places it in a local page, updates the page tables, and then lets the VM continue.

Background processes running on all nodes of the DSM periodically exchange information about the available free memory. Donor nodes are selected on-demand using the worst-fit policy. The minimum memory request size is 64MB to reduce the overhead of DSM metadata. The donor node allocates a contiguous address range in its memory, and the kernel module maps a part of the VM's virtual address space to this area. Victim pages are selected by an approximate-LRU algorithm implementing a CLOCK policy [4].

Memory Server. Each memory node in the DSM cluster runs a daemon to serve memory requests from other nodes. Each memory server is connected to all other nodes. Memory allocation requests are handled using remote procedure calls (RPC). In response to an allocation request, the memory server locally allocates memory using `malloc` and registers the area for `RDMA`. After the registration is complete, the memory server replies to the request by sending the allocated memory's DSM metadata which is used by the requesting node for later accesses.

4.2 Userspace DSM Library

To allow the implementation of our DSM-optimized live migration technique in userspace, a userspace library is available that provides the necessary API. The API implements the following functionality: (1) set local DSM cache size, (2) read status of local DSM page, (3) make a DSM region persistent, (4) drop specific pages of a DSM region, and (5) provide access to DSM metadata. The library is used by the DSM-optimized algorithm presented in Section 3.3.

4.3 DSM Live Migration Implementation in QEMU/KVM

We implement the DSM-optimized VM live migration technique in the most popular VM hypervisor, QEMU/KVM [5]. QEMU/KVM has been widely adopted in production clouds, hence adding the new feature to QEMU/KVM will allow for an easy adoption of the new approach into existing data centers. The functions enabling live migration over DSM are implemented in a library so the technique can be easily accessed by other hypervisors such as VMware and Xen.

Starting a VM on DSM. QEMU/KVM internally allocates a contiguous address range from anonymous memory using `mmap`. Modifying QEMU/KVM to use DSM-backed memory is trivial: the only necessary change is to provide a file descriptor to the DSM device node instead of a NULL pointer to signal anonymous memory. The DSM kernel modules transparently handles all accesses to memory of a VM, local or remote, so VM itself requires no modification and can make use of DSM-backed memory transparently.

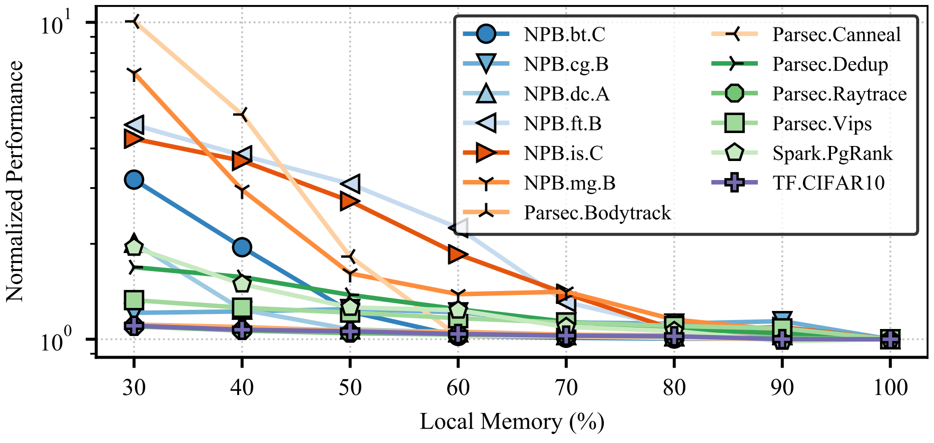

Fig. 6. Application performance on DSM

DSM live migration in QEMU. QEMU/KVM supports many live migration techniques out-of-the-box with a clean API. Adding a new live migration technique requires describing the parameters of the new live migration technique, then QEMU automatically generates the boilerplate code. The DSM-optimized live migration algorithm is implemented with around 350 lines of code. The code mainly invokes the DSM userspace API to retrieve a page's status, to obtain a list of dirtied pages and compute the pages to be transferred, and to send a few messages from the source to the destination to synchronize the live migration steps between the source and the destination.

5 Evaluation

This section evaluates the DSM-optimized live migration technique and compares it with DSM-unaware pre-copy as available in QEMU/KVM.

5.1 Test Environment

The source and destination nodes are equipped with an Intel(R) Xeon(R) Silver 4114 2.20GHz processor (10-core, 20-thread) and 64 GiB of local memory. Each machine runs Ubuntu 18.04 with Linux kernel version 5.3.7. The machines are connected by a ConnectX-4 56g Infiniband NIC with a theoretical bandwidth of 56 Gbit/s and a latency of 0.6μs. The DSM and VM live migration traffic share the same Infiniband connection and can interfere. The VM network is physically separated and uses a 1 Gbit/s Ethernet NIC. QEMU/KVM version 4.2 provides the hypervisor, and the VMs run Ubuntu 18.04 as their guest OS. The VMs use network-attached storage to access their disk images. VM disk I/O traffic is routed through the VM network. All VMs used in the experiments are equipped with 4-VCPU and 8 GB of local memory. The local memory is limited by our DSM depending on the evaluated scenario.

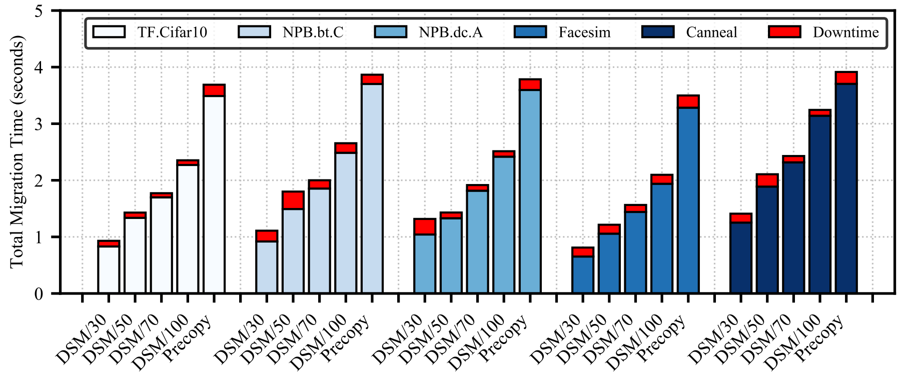

Fig. 7. Total migration time of the DSM approach and the baseline

5.2 Workloads Used in the Evaluation

The presented live migration technique over DSM is evaluated with a wide range of applications from high-performance parallel processing benchmark suites (Parsec, NPB, and Spark) and the CIFAR10 Tensorflow inference benchmark. All applications run with four threads to fully utilize all available cores. The total memory footprint of the applications is 2.5 GB on average. The average working set size (measured by scanning the access bit in the page table) over a 60s window is 1.2 GB on average with a standard deviation of 877 MB. The workloads have dynamically changing working sets and the memory footprints are large enough to stress the DSM.

5.3 Application Performance on DSM

We first show that DSM is a viable option for a virtualized environment by evaluating application performance degradation in dependence of the allocated local memory. We tested application performance in a VM by limiting the DSM local memory size against the application's total memory footprint. The baseline is obtained by running the VM entirely in local memory; sufficient memory is available such that no paging occurs. Figure 6 shows the application performance of DSM-backed-storage normalized to the baseline; note the log-scale of the Y axis.

The results in Figure 6 show that all applications run without noticeable performance degradation with 80% of local memory. Even in the extreme case of only providing 30% of the total memory footprint as local memory, more than half of all applications show a high tolerance with a slowdown of under 2. Note that the tested applications are all memory-bound, so we expect that less memory-intensive applications such as smaller databases and web service application will show even better tolerance than the evaluated applications.

5.4 Performance of DSM-Optimized Live Migration

Next, we present the results of our DSM-optimized live migration algorithm applied to VMs running the applications from Section 5.3. The DSM-optimized approach is compared to the pre-copy algorithm implemented in QEMU/KVM. QEMU/KVM supports optional data compression to reduce the amount of transferred data. However, we found that data compression is counterproductive since the compression algorithm cannot saturate the fast network links. As a consequence, data compression is not enabled for the baseline and our DSM-optimized migration technique.

Figure 7 the total migration time of the presented DSM-optimized live migration algorithm compared to QEMU/KVM's pre-copy algorithm. The y-axis represents the absolute total migration time in seconds, the x-axis five representative applications with at 30%, 50%, 70%, and 100% of the total application memory footprint in local memory. The Infiniband NIC is used in IPoIB (IP over Infiniband) mode and the `TCP` protocol for the data transfer. The result shows that the presented approach clearly outperforms the baseline. The more memory is remote, the better performs the DSM-optimized live migration. For example, with 30% local memory, the source needs to send at most 30% of the total VM's memory in one iteration. Note that our approach also outperforms QEMU's pre-copy with 100% of VM memory kept locally. This seemingly counterintuitive result stems from the way zero pages are handled. Zero pages are pages whose contents are only 0s; QEMU excludes such pages to reduce the amount of data transferred. While QEMU has to inspect every page to determine whether it is a zero page or not, this information is readily available in our DSM, which considerably speeds up the detection of zero pages and leads to overall better performance. On average, our approach reduces the total migration time by 70 and the downtime by 8 percent. The total amount of transferred data is reduced by 65% at a 30% local memory ratio setting.

Overall, these results are very promising and show the potential of DSM-backed virtualization. Live migration needs to be fast when a node experiences a shortage of resources. It is reasonable to assume that in this case, the ratio of local memory has already been reduced at least for some VMs; and migrating such VMs away from the overloaded outperforms standard live migration by a factor 2-4.

6 Related Work

VM live migration and DSM have more than a decade of history. In this section, we introduce previous work and highlight our contribution.

VM live migration. Clark [11] first presented the *pre-copy* approach and showed that a VM can be migrated seamlessly with its workload After the inception of pre-copy, many follow-up works have presented optimization techniques to make live migration more efficient. Hines [14] presented the *post-copy* approach which trades VM performance for a reduction in the transferred data and a shorter downtime. Many works have shown optimization techniques for live migration performance that can be applied to both pre-copy and post-copy. In difference to this work, most of the published research focuses on VMs with a small amount of memory of up to 10 GiB. VM workloads have changed significantly along with hardware improvements, ubiquitous parallel processing, and much larger working set sizes than a decade ago. We argue that it is the right time to think about a fundamentally different approach for VM live migration; such as, for example, the DSM approach presented in this paper.

Distributed Shared Memory. DSM also has a long history in the computer systems, and many different designs have been proposed. In this paper, we focus on executing VMs or unmodified processes on a mix of local and remote memory to improve overall utilization of memory resources. One early work on this topic is NSwap [16]. As the name implies, it implements a swap device using remote memory as secondary storage. It is a special kind of DSM that does not support cache coherence, naming, and data sharing but provides a simple way of space sharing of the cluster of memory. Many follow-up works show some promising aspects of using remote memory as an extension of local memory, however, none of them were realized in a production environment due to the slow networks and insufficient computation power of that time. Similar ideas have been revived with the significant advances in networking hardware in recent days [8, 9, 12, 13, 15, 20]. The performance gap between local and remote memory access is now quite close [10, 18], and we expect this trend to continue with optical networks.

Many recent works on this topic show promising results of running applications on a mix of local and remote memory; in difference to this work, they focus on native applications and not virtual machines. Certain types of application have been shown to perform well even in highly memory-constrained environments; a results that makes DSM for virtual environments even more promising.

7 Conclusion

This paper has presented a design for distributed shared memory tailored at virtualized environments. A novel DSM-optimized live migration algorithm makes use of the DSM's properties to improve several metrics of VM migration. The presented approach is able to significantly reduce both the total migration time and the downtime of large memory VMs and enables almost instant VM live migration for moderately-sized VMs.

Acknowledgments

This work was supported by the National Research Foundation of Korea (NRF) funded by the Korean government, in part, by grants NRF-2015K1A3A1A14021288, 2016R1-A2B4009193, by the BK21 Plus for Pioneers in Innovative Computing (Dept. of Computer Science and Engineering, SNU, grant 21A20151113068), and by the Promising-Pioneering Researcher Program of Seoul National University in 2015. ICT at Seoul National University provided research facilities for this study.

References

1. Amazon ec2 instance types - amazon web services. https://aws.amazon.com/ec2/instance-types/, (online, accessed July 2020)
2. Amazon ec2 instance types - memory optimized - high memory. https://aws.amazon.com/ec2/instance-types/#Memory_Optimized, (online, accessed July 2020)
3. Google cloud platform blog: Now shipping: ultramem machine types with up to 4tb of ram. https://cloudplatform.googleblog.com/2018/07/now-shipping-ultramem-machine-types-with-up-to-4tb-of-ram.html, (online, accessed July 2020)

4. Page replacement algorithm - wikipedia. https://en.wikipedia.org/wiki/Page_replacement_algorithm, (online, accessed July 2020)
5. Qemu. https://www.qemu.org/, (online, accessed July 2020)
6. Virtual machine series — microsoft azure. https://azure.microsoft.com/en-us/pricing/details/virtual-machines/series/, (online, accessed July 2020)
7. Vm instances pricing — compute engine documentation — google cloud. https://cloud.google.com/compute/vm-instance-pricing, (online, accessed July 2020)
8. Aguilera, M.K., Amit, N., Calciu, I., Deguillard, X., Gandhi, J., Novakovic, S., Ramanathan, A., Subrahmanyam, P., Suresh, L., Tati, K., et al.: Remote regions: a simple abstraction for remote memory. In: 2018 USENIX Annual Technical Conference (USENIX ATC 18) (2018)
9. Amaro, E., Branner-Augmon, C., Luo, Z., Ousterhout, A., Aguilera, M.K., Panda, A., Ratnasamy, S., Shenker, S.: Can far memory improve job throughput? In: Proceedings of the Fifteenth European Conference on Computer Systems. EuroSys 20, Association for Computing Machinery, New York, NY, USA (2020). https://doi.org/10.1145/3342195.3387522
10. Binnig, C., Crotty, A., Galakatos, A., Kraska, T., Zamanian, E.: The end of slow networks: Its time for a redesign. Proc. VLDB Endow. **9**(7) (Mar 2016). https://doi.org/10.14778/2904483.2904485
11. Clark, C., Fraser, K., Hand, S., Hansen, J.G., Jul, E., Limpach, C., Pratt, I., Warfield, A.: Live migration of virtual machines. In: Proceedings of the 2nd Conference on Symposium on Networked Systems Design and Implementation - Volume 2. NSDI05 (2005)
12. Gao, P.X., Narayan, A., Karandikar, S., Carreira, J., Han, S., Agarwal, R., Ratnasamy, S., Shenker, S.: Network requirements for resource disaggregation. In: 12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16) (2016)
13. Gu, J., Lee, Y., Zhang, Y., Chowdhury, M., Shin, K.G.: Efficient memory disaggregation with infiniswap. In: 14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17) (2017)
14. Hines, M.R., Gopalan, K.: 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. VEE 09, Association for Computing Machinery, New York, NY, USA (2009). https://doi.org/10.1145/1508293.1508301
15. Koh, K., Kim, K., Jeon, S., Huh, J.: Disaggregated cloud memory with elastic block management. IEEE Transactions on Computers **68**(1) (2018)
16. Newhall, T., Finney, S., Ganchev, K., Spiegel, M.: Nswap: A network swapping module for linux clusters. In: European Conference on Parallel Processing. Springer (2003)
17. Nitu, V., Teabe, B., Tchana, A., Isci, C., Hagimont, D.: Welcome to zombieland: Practical and energy-efficient memory disaggregation in a datacenter. In: Proceedings of the Thirteenth EuroSys Conference. EuroSys 18, Association for Computing Machinery, New York, NY, USA (2018). https://doi.org/10.1145/3190508.3190537
18. Rumble, S.M., Ongaro, D., Stutsman, R., Rosenblum, M., Ousterhout, J.K.: It's time for low latency. In: HotOS. vol. 13 (2011)
19. Ruprecht, A., Jones, D., Shiraev, D., Harmon, G., Spivak, M., Krebs, M., Baker-Harvey, M., Sanderson, T.: Vm live migration at scale. In: Proceedings of the 14th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments. VEE 18, Association for Computing Machinery, New York, NY, USA (2018). https://doi.org/10.1145/3186411.3186415
20. Shan, Y., Huang, Y., Chen, Y., Zhang, Y.: Legoos: A disseminated, distributed {OS} for hardware resource disaggregation. In: 13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18) (2018)