Fast Wide Area Live Migration with a Low Overhead through Page Cache Teleportation

Soramichi Akiyama*, Takahiro Hirofuchi†, Ryousei Takano† and Shinichi Honiden*‡
*The University of Tokyo, Bunkyo, Tokyo, Japan 113–8656
Email: {akiyama, honiden}@nii.ac.jp
†National Institute of Advanced Industrial Science and Technology, Japan
Email: {t.hirofuchi, takano-ryousei}@aist.go.jp
‡National Institute of Informatics, Japan

***Abstract*—Live migration of virtual machines over a wide area network has many use cases such as cross-datacenter load balancing, low carbon virtual private clouds, and disaster recovery of IT systems. An efficient wide area live migration method is required because cross-datacenter connections have a narrow bandwidth. Page cache occupies a large portion of the memory of a Virtual Machine (VM) when it executes data-intensive workloads. We propose a new live migration technique, page cache teleportation, which reduces the total migration time of wide area live migration and has a low overhead. It detects the restorable page cache in the guest memory that has the same contents as the corresponding disk blocks. The restorable page cache is not transferred via the WAN but is restored from the disk image before the VM resumes. In this way, the IO performance degradation reduces after the migration. Evaluations show that page cache teleportation reduces the total migration time of wide area live migration and has a lower performance overhead than existing approaches.**

***Keywords*-virtualization; live migration; datacenter; cloud computing; wide area migration**

I. Background

A. Wide Area Live Migration

Wide area live migration has many use cases. Al-Kiswany *et al.* achieve cross-datacenter load balancing by using wide area live migration [1]. Load balancing within a datacenter with local area migration is widely adopted in research and products [2], [3], but [1] is unique in the point that it distributes the load of a datacenter to other distant datacenters. Moghaddam *et al.* minimize the carbon footprint of a virtual private cloud through a live migration of VMs to a datacenter that uses clean energy sources as much as possible [4]. Tsugawa *et al.* [5] proposes to use wide area live migration for disaster recovery of indispensable IT systems. VMs can survive a disaster by using live migration of them to safe datacenters when a disaster occurs. The work shows that when the great earthquake hit Japan in 2011, the network connectivity and the uninterruptible power supply were kept alive for dozens of minutes at Tohoku University (150 km from the epicenter).

An efficient wide area live migration method is mandatory to allow practical use of these research. The greatest issue to achieve this is the bandwidth bottleneck of the network between datacenters. Connections within a datacenter are provided with 1 Gbps or even 10 Gbps links these days. However, cross-datacenter connections often have a narrower bandwidth and the links underneath are shared resources. Therefore, reducing the amount of transferred data to reduce the total migration time is an important design criterion of wide area live migration.

B. Impact of Page Cache upon Live Migration

Page cache is on-memory cache for disk access, which is slower than memory access. Free memory pages are allocated as much as possible for page cache in Linux and Windows. Hence a large portion of a guest memory is used for page cache when data-intensive workloads such as web servers with large contents and transaction systems with databases are executed. Therefore, reducing the amount of transferred memory used for page cache is important in wide area live migration.

Deleting page cache just before a live migration to reduce the memory transfer is problematic. Page cache is *restorable* when the same data exists on disk. Deleting restorable page cache and reloading it after a migration has been suggested in existing research [6], [7], [8]. However, this approach degrades the IO performance of the guest OS after a migration in some workloads because the cache does not exist on the memory and, thus, has to be reloaded from the disk image.

C. Research Goal

Our goal is to invent low-cost live migration techniques and to achieve a more aggressive VM placement optimization than existing studies. Some studies (e.g. [9]) ignore the costs of live migration and cannot be adopted to the real world. Others (e.g. [10], [11]) avoid the costs by limiting the number of migrations or abandoning the use of them; thus, their optimization policies are conservative. We believe that reducing the cost of live migration mitigates these disadvantages.

This paper focuses on reducing the cost of wide area live migration by taking page cache into account. As described in Section I-B, page cache in a guest memory has a great impact on the performance of live migration. In [12], we have accelerated live migration by keeping a VM memory image on a host and reusing the unchanged memory pages when the VM migrates back to the host. This is useful when the page cache in the guest memory is large because it can stay unchanged for a long time. However, the utility is limited in

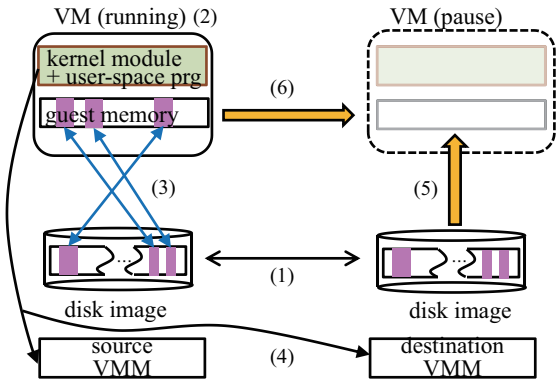

Fig. 1. Proposed live migration procedure. (1) Disk image is synchronized before a migration. (2) Write accesses to the memory are tracked during the following steps. (3) The kernel module detects where the restorable page cache is on the guest memory and the disk. (4) The module sends them to the VMMs. (5) The destination VMM copies the disk blocks to the guest memory. (6) The source VMM sends memory pages except the restorable page cache, which has not been modified during steps (3)–(5).

a WAN environment because a VM must return to a host on which it has been executed before. Thus we investigate a new technique suitable for wide area live migration in this paper.

II. Proposal: Page Cache Teleportation

A. Overview

Our proposal, *page cache teleportation*, reduces the amount of transferred memory used for page cache while introducing a lower performance overhead on the target VM, in comparison to existing research. It consists of two main features:

1) Page cache is **restored** from the disk image at the destination instead of being transferred over the WAN.
2) The restore process is executed **while the VM keeps running** on the source.

Feature (1) accelerates wide area live migration as reading the page cache from a local disk or a datacenter-local shared storage is much faster than transferring it over the WAN. Feature (2) reduces the penalty to the IO performance because page cache has been fully restored when the VM resumes.

B. How Our System Works

Wide area live migration with page cache teleportation is executed as illustrated in Figure 1. Detailed descriptions of each step are as follows:

1) The disk image is synchronized before a live migration.
2) Write access to the memory is tracked during the following steps (please refer to Section III-C for details).
3) Our kernel module inside the guest OS detects the page frame numbers (PFNs) of the restorable page cache and the corresponding disk block numbers.
4) Our user-space program inside the guest OS sends the PFNs and the disk block numbers to the source virtual machine monitor (VMM) and the destination VMM.
5) The destination VMM copies the restorable page cache to the guest memory (*page cache restore*).
6) The source VMM transfers the guest memory except the restorable page cache to the destination over the WAN. The guest OS is instantly suspended and the CPU states are transferred when the amount of remaining memory becomes sufficiently small.

III. Implementation

A. Design

We use QEMU/KVM as the VMM because it is open-source and widely used in the research field. We modularize the implementation into three components:

1) A kernel module that detects the PFNs of the restorable page cache and the corresponding disk block numbers.
2) A user-space program that sends the PFNs and disk block numbers to the VMMs.
3) A modified VMM that copies the restorable page cache to the guest memory at the destination, and skips transferring it at the source. It also tracks write access to the memory at the source to keep the memory consistency.

B. Detecting Restorable Page Cache

Our kernel module detects the PFNs of the restorable page cache in the guest memory and the corresponding disk block numbers in the disk image. This is the only part in our proposal that depends on a specific guest OS. Currently we support only Linux, but we have a plan to support other operating systems. The module has 200 lines of code and takes up 155 KB in binary format. Thus, it is sufficiently small to be installed in the guest OS. It is executed only during migrations and takes less than 1 second to scan 1 GB of memory. Compared to an approach in which the host OS analyzes the guest memory, this design reduces the cost of the implementation and adaptation to applications that has an own cache manager. The module does not need to know the binary placements of kernel data structures, though analyzing the guest memory from the host OS requires that. Restoring the own cache of an application requires a new module and a new user-space program but VMMs need not to be updated, which means datacenter operations are not interrupted.

The module easily detects the page cache using kernel functions of the guest OS. In Linux, the PFN of a memory page can be translated into the disk block number corresponding to it by using certain kernel functions (`pfn_to_page` and `bmap`), if the page is used for page cache. A kernel data structure includes a flag indicating that the memory page is not yet written back to the disk and the page is not restorable. Windows also provides similar functionalities, which can be used in the future to port the module into Windows.

The PFNs and the disk block numbers retrieved by the kernel module are sent to the VMMs by our user-space program inside the guest OS. Any communication mechanism can be used between the program and the VMMs; we simply use TCP/IP. Mechanisms such as a shared memory between the guest OS and the source VMM can be used, but could have OS dependencies and few advantages.

TABLE I
EVALUATION ENVIRONMENT

CPU	Intel Xeon X5460 (4 cores)
Memory	8 GB
Disk	250GB HDD (read: 90 MB/s, write: 25 MB/s)
Network	limited to 100 Mbps or 50 Mbps
OS	Debian 6.0.5 (Linux 2.6.32)
QEMU	0.13.0

C. Guaranteeing Memory Consistency

The most important challenge in our system is how to guarantee the consistency of the guest memory during a migration. The page cache is restored at the destination while the guest OS continues running on the source to avoid causing an IO performance overhead on the guest OS. Suppose a memory page that contains restorable page cache is updated while the page cache is being restored. In this case, the page must be transferred via the WAN to keep the consistency because the restored data is obsolete. There are two cases in which a memory page used for the page cache is updated: the cache data contained in the page might be updated or the guest OS might free the page and use it for a purpose other than page cache.

We solve this issue by using the dirty page tracking functionality of the VMM. The x86 architecture has a dirty bit for each memory page that is set when the page is updated. QEMU can read the dirty bits from a user-space context. Dirty page tracking is enabled at the source when a migration starts and all the memory writes are tracked afterwards. A memory page updated during the tracking is transferred over the WAN, even if the memory page **was** restorable when the kernel module detected restorable page cache.

D. Restoring Page Cache

The restorable page cache is copied into the guest memory by a user-space context of QEMU. The copy operations are done transparently to the guest OS. Disk blocks (4 KB each) that include the restorable page cache are copied in the ascending order of the disk block numbers. This is because read throughput of a physical disk is in general maximized with sequential reads.

Analyzing and modifying the kernel data structures is not required. Instead, simply copying the disk blocks to the guest memory is sufficient. The data contained in restorable page cache are the contents of files and their metadata on the disk image (e.g. `inode`), thus they are exactly the same in memory and on disk. Note that updates to the page cache are properly treated as describe in Section III-C. Run-time metadata, such as a radix-tree to look up page cache, is treated as normal memory pages and transferred over the WAN.

IV. EVALUATION

A. Total Migration Time

We show that page cache teleportation reduces the total migration time of wide area live migration. The total migration time refers to the time period between the start of a migration and the point when the memory, CPU and device states have been transferred to the destination. A VM running practical benchmarks was moved by live migration over a simulated WAN and the total migration time was measured. Migrations were executed with the original QEMU/KVM and with our VMM in which we implemented page cache teleportation. The executed benchmarks are as follows:

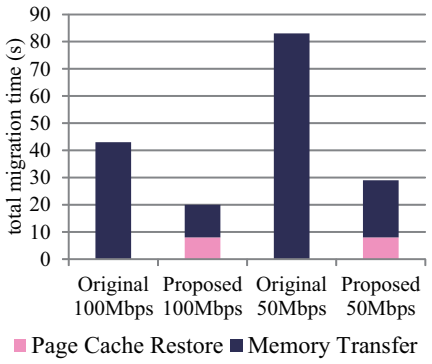

Fig. 2. Total Migration Time (seconds) of the Web Server Benchmark

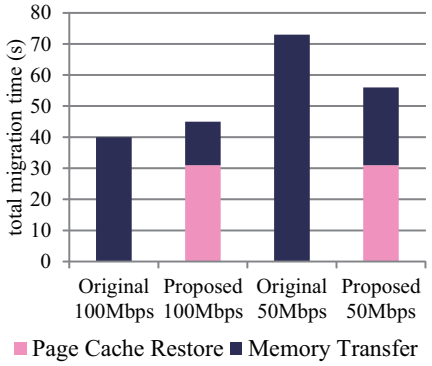

Fig. 3. Total Migration Time (seconds) of the TPC-C Benchmark

Web Server: A simulated web server benchmark without a database. Apache manages static files and a load-generator reads the files from outside of the VM. The files are read using a different network from the one used by the live migration to avoid bandwidth pressure. The number of files is 1024 and the size of each file is 300KB (imitating the typical size of a flash content or an image).

TPC-C [13]: A simulated transaction system with a database. The system generates DB access patterns of a mimic net shopping web site. The total size of the DB data is 1.9 GB.

The evaluation environment is described in Table I. The guest OS is Debian 6.0.5 with 1 CPU and 1 GB of memory. Note that the guest kernel version or OS distribution can be different from those of the host. The network bandwidth was limited to 100 Mbps or 50 Mbps by using the `tc` command at the hosts. The read/write throughput of the HDD was measured by bonnie++ [14] with a sequential access pattern. The disk image is synchronized between the hosts using DRBD [15].

TABLE II
AMOUNT OF TRANSFERRED MEMORY

	Web Server		TPC-C	
	Proposed	Original	Proposed	Original
100 Mbps	110 MB	487 MB	142 MB	440 MB
50 Mbps	110 MB	498 MB	139 MB	415 MB

Figure 2 and Figure 3 show the total migration time for each benchmark. The values are averaged over three runs. The labels "Original" and "Proposed" mean that we used the original QEMU/KVM and the one integrated with our proposal, respectively. The blue part of each bar indicates the time consumed to transfer the memory pages, and the pink part indicates the time consumed to restore the page cache. In the Web Server benchmark, page cache teleportation reduced the total migration time more than 50% in both the 100 Mbps and 50 Mbps environments. The time consumed to restore the page cache is constant regardless of the bandwidth. This is because the disk image is synchronized before the migration starts and the time depends only on read throughput of the disk image. In the TPC-C benchmark, our proposal reduced the total migration time in the 50 Mbps environment, but not in the 100 Mbps environment. The reason is that the disk read throughput was slower than 100 Mbps because the restored disk blocks are scattered over a wide range of the disk image in this benchmark.

Table II describes the amount of transferred memory in the benchmarks. The values are averaged of over three runs and do not include the amount of the restored page cache. Apache manages 1024 static files with 300 KB each, thus the total size of the files is 300 MB. The reductions are meaningfully larger than 300 MB for both bandwidths, because the page cache contains not only Apache contents but also kernel code, shared libraries, etc. We found that even when the guest OS is completely idle, 45 MB of the memory is reduced through our proposal. The results of the TPC-C benchmark show that our proposal reduced the memory transfer around 300 MB in both bandwidths. Note that the "Original" had a shorter total migration time in the 100 Mbps bandwidth case than our proposal, as explained in the end of the previous paragraph. The amount of transferred memory was less in 50 Mbps than in 100 Mbps because the TPC-C performance was penalized by the degradation of the disk synchronization performance due to the narrower bandwidth.

B. Performance Penalty on the Guest OS

This section shows that the performance penalty of our proposal is sufficiently small. Page cache conceals the gap between the access speeds of the memory and the disk, thus, deleting the page cache penalizes the IO performance of the guest OS. On the other hand, our proposal restores the page cache **before** the guest OS resumes at the destination. The page cache is fully warmed up from the view point of the guest OS, thus, no penalty on IO performance occurs.

Figure 4 shows the file read throughput of a VM, which is live migrated at the 150^{th} second. The VM reads a 1 GB file during 250 seconds, starting again from the head after it reaches the end of the file. The top figure shows the results with our proposal and the bottom one shows the ones with a naïve method, which deletes page cache just before a migration (Linux supports to delete page cache with `/proc` interfaces). The throughput is low at the beginning but increases at 15 seconds due to the page cache. In the top figure, the throughput degrades when the migration starts due to the load of memory copying and networking, but it recovers in few seconds because the page cache is restored before the VM resumes. However, the bottom figure shows 15 seconds of throughput degradation because the guest OS must reload the file from the disk image.

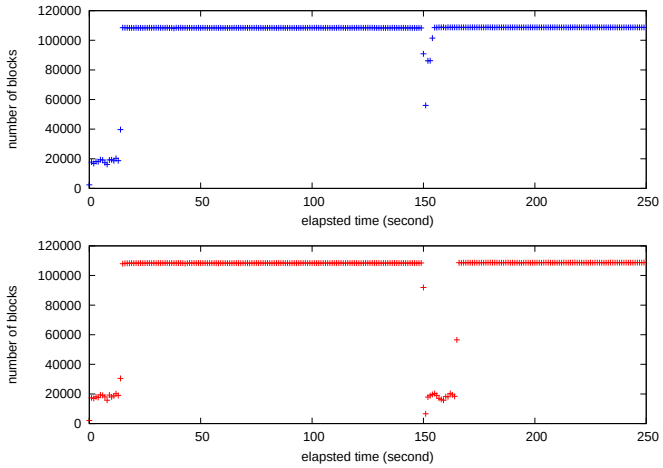

Fig. 4. File read throughput with our proposal (top) and with an approach that deletes the page cache just before a migration (bottom). The migration starts from the 150^{th} second in both cases. The figures show that our proposal incredibly reduces the IO performance penalty.

V. RELATED WORK

We compare our proposal with existing research in this section. Koto *et al.* discuss computational costs of live migration (referred as *migration noise*) in [7]. The work reduces the migration noise by skipping the transfer of restorable pages including page cache, free pages, and kernel objects that can be regenerated from other data. Hines *et al.* skip the transfer of the page cache by using the balloon driver of Xen [8]. In a paravirtualization environment with Xen, a guest OS returns unused memory pages to the Xen using the balloon driver. Hence, if the guest OS deletes the page cache then Xen can skip the transfer of the page cache in a live migration.

The biggest difference between our proposal and these works is whether the page cache is restored before the guest OS resumes or not. Our proposal restores the page cache before the guest OS resumes and minimizes the performance penalty as evaluated in Section IV-B. On the other hand, in the existing research, the guest OS must reload the page cache after a migration and its IO performance will be degraded.

Many research approaches reduce the total migration time with different techniques. Svärd *et al.* compress the delta of two versions of a memory page when the page is re-transferred in pre-copy live migration [16]. Zhang *et al.* deduplicate memory pages that have the same contents using a hash function [17]. Wood *et al.* combine above two techniques with *smart stop and copy*, which minimizes the number of iterative copies of updated memory pages [18]. These approaches can be used together with our proposal.

VI. Conclusion and Future Work

This paper proposes page cache teleportation to accelerate wide area live migration by reducing the amount of transferred memory used for page cache. We detect the location of the page cache in the guest memory and restore it from the disk image instead of transferring it over a WAN. Our proposal does not degrade the IO performance of the guest OS after a migration, in comparison to existing research. Evaluations show that page cache teleportation reduces the total migration time of wide live migration under limited bandwidth, and that the performance penalty to the guest OS is much smaller than existing research.

Page cache teleportation demands large-scale experiments and technical improvements. Large-scale experiments help to evaluate the temporal locality of page cache in real applications and network pressure to synchronize disk images. Temporal locality affects the overhead of deleting the page cache; thus, evaluating it clarifies the utility of our proposal. Possible technical improvements include concurrent execution of page cache restore and the transfer of other memory pages. They use different data channels (storage IO and network) and can be overlapped to accelerate the wide area live migration even more.

This paper is a milestone of our research goal to an achieve aggressive VM placement optimization based on fast live migration. In [12] and this paper, we have accelerated local and wide area live migration with the assumption that the page cache occupies a large portion of guest memory. Future work includes (1) combining our techniques with existing research that exploits other memory characteristics than page cache and (2) investigating a new VM placement mechanism that utilizes our fast live migration techniques aggressively.

Acknowledgments

This work is partially funded by J-RAPID and CREST/ULP of the Japan Science and Technology Agency, and KAKENHI 23700048 by the Ministry of Education, Culture, Sports, Science and Technology in Japan. The authors would like to thank the anonymous reviewers for the valuable comments, as well to their colleagues and staffs for the kind help.

References

[1] S. Al-Kiswany, D. Subhraveti, P. Sarkar, and M. Ripeanu, "Vmflock: Virtual machine co-migration for the cloud," in *International Symposium on High Performance Distributed Computing*, 2011, pp. 159–170.

[2] A. Singh, M. Korupolu, and D. Mohapatra, "Server-storage virtualization: integration and load balancing in data centers," in *ACM/IEEE Conference on Supercomputing (SC)*, 2008, pp. 1–12.

[3] "VMware vSphere Distributed Resources Scheduler & Power Management." [Online]. Available: http://www.vmware.com/products/datacenter-virtualization/vsphere/drs-dpm.html

[4] F. F. Moghaddam, M. Cheriet, and K. K. Nguyen, "Low carbon virtual private clouds," in *IEEE International Conference on Cloud Computing*, 2011, pp. 259–266.

[5] M. Tsugawa, R. Figueiredo, J. Fortes, T. Hirofuchi, H. Nakada, and R. Takano, "On the use of virtualization technologies to support uninterrupted it services," in *IEEE ICC 2012 Workshop on Re-think ICT infrastructure designs and operations*, 2012, pp. 7892–7896.

[6] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines," in *Symposium on Networked Systems Design & Implementation*, 2005, pp. 273–286.

[7] A. Koto, H. Yamada, K. Ohmura, and K. Kono, "Towards unobtrusive vm live migration for cloud computing platforms," in *Asia-Pacific Workshop on Systems (APSys)*, 2012.

[8] M. R. Hines and K. Gopalan, "Post-copy based live virtual machine migration using adaptive pre-paging and dynamic self-ballooning," in *International Conference on Virtual Execution Environments*, 2009, pp. 51–60.

[9] M. Mishra and A. Sahoo, "On theory of vm placement: Anomalies in existing methodologies and their mitigation using a novel vector based approach," in *IEEE International Conference on Cloud Computing*, 2011, pp. 275–282.

[10] T. Wood, G. Tarasuk-Levin, P. Shenoy, P. Desnoyers, E. Cecchet, and M. D. Corner, "Memory buddies: exploiting page sharing for smart colocation in virtualized data centers," in *International Conference on Virtual Execution Environments*, 2009, pp. 31–40.

[11] O. Biran, A. Corradi, M. Fanelli, L. Foschini, A. Nus, D. Raz, and E. Silvera, "A stable network-aware vm placement for cloud systems," in *International Symposium on Cluster, Cloud and Grid Computing*, 2012, pp. 498–506.

[12] S. Akiyama, T. Hirofuchi, R. Takano, and S. Honiden, "Miyakodori: A memory reusing mechanism for dynamic vm consolidation," in *IEEE International Conference on Cloud Computing*, 2012, pp. 606–613.

[13] "TPC-C." [Online]. Available: http://www.tpc.org/tpcc/

[14] "bonnie++." [Online]. Available: http://www.coker.com.au/bonnie++/

[15] "DRBD." [Online]. Available: http://www.drbd.org/

[16] P. Svärd, B. Hudzia, J. Tordsson, and E. Elmroth, "Evaluation of delta compression techniques for efficient live migration of large virtual machines," in *Internatilnal Conference on Virtual Execution Environment*, 2011, pp. 111–120.

[17] X. Zhang, Z. Huo, J. Ma, and D. Meng, "Exploiting data deduplication to accelerate live virtual machine migration," in *Internatinal Conference on Cluster Computing*, 2010, pp. 88–96.

[18] T. Wood, K. K. Ramakrishnan, P. Shenoy, and J. Van der Merwe, "Cloudnet: dynamic pooling of cloud resources by live wan migration of virtual machines," in *International Conference on Virtual Execution Environments*, 2011, pp. 121–132.