Shrinker: Improving Live Migration of Virtual Clusters over WANs with Distributed Data Deduplication and Content-Based Addressing

Pierre Riteau[1,2], Christine Morin[2], and Thierry Priol[2]

[1] Université de Rennes 1, IRISA, Rennes, France
[2] INRIA, Centre INRIA Rennes - Bretagne Atlantique, Rennes, France
Pierre.Riteau@irisa.fr, {Christine.Morin,Thierry.Priol}@inria.fr

Abstract. Live virtual machine migration is a powerful feature of virtualization technologies. It enables efficient load balancing, reduces energy consumption through dynamic consolidation, and makes infrastructure maintenance transparent to users. While live migration is available across wide area networks with state of the art systems, it remains expensive to use because of the large amounts of data to transfer, especially when migrating virtual clusters rather than single virtual machine instances. As evidenced by previous research, virtual machines running identical or similar operating systems have significant portions of their memory and storage containing identical data. We propose Shrinker, a live virtual machine migration system leveraging this common data to improve live virtual cluster migration between data centers interconnected by wide area networks. Shrinker detects memory pages and disk blocks duplicated in a virtual cluster to avoid sending multiple times the same content over WAN links. Virtual machine data is retrieved in the destination site with distributed content-based addressing. We implemented a prototype of Shrinker in the KVM hypervisor and present a performance evaluation in a distributed environment. Experiments show that it reduces both total data transferred and total migration time.

Keywords: Virtualization, Live Migration, Wide Area Networks, Cloud Computing.

1 Introduction

The complete encapsulation of execution environments (applications combined together with their underlying OS) in virtual machines (VMs) allowed the development of live virtual machine migration [4,20]. This mechanism relocates a virtual machine from one host to another with minimal downtime, usually not noticeable by users. This offers numerous advantages for data center management, including:

Load balancing: VMs can be dynamically migrated depending on workload, offering more efficient usage of computing resources.

E. Jeannot, R. Namyst, and J. Roman (Eds.): Euro-Par 2011, LNCS 6852, Part I, pp. 431–442, 2011.

Reduced energy consumption: VMs with low workloads can be consolidated to fewer physical machines, making it possible to power off nodes to reduce energy consumption.

Transparent infrastructure maintenance: Before physical machines are shut down for maintenance, administrators can relocate VMs to other nodes without noticeable interruption of service for users.

Mainstream hypervisors usually do not support live migration between different data centers connected with wide area networks (WANs). To avoid transferring persistent state of VMs, which can be of large size (from hundreds of megabytes to dozens of gigabytes and beyond), they depend on shared storage, usually not accessible across different data centers. They also require that migrated VMs stay in the same local network, to keep network connections uninterrupted after migration to new host machines.

These restrictions prevents users from getting the benefits of live virtual machine migration over WANs. It would allow administrators to load balance workload between several data centers, and offload VMs to other sites whenever a site-wide maintenance is required. Users with access to private clouds (private computing infrastructures managed with cloud computing stacks) could seamlessly migrate VMs between private and public clouds depending on resource availability. It could also allow them to leverage variable price between competing cloud providers.

Fortunately, state of the art systems allow to use live migration over WANs by migrating storage [3,16,10,32] and network connections [3,7,32]. However, the large amounts of data to migrate make live migration over WANs expensive to use, especially when considering migrations of virtual clusters rather than single VM instances.

Previous research showed that VMs running identical or similar operating systems have significant portions of their memory and storage containing identical data [30,5]. In this paper, we propose Shrinker, a live virtual machine migration system leveraging this common data to improve live virtual cluster migration between data centers interconnected by wide area networks. Shrinker detects memory pages and disk blocks duplicated in a virtual cluster to avoid sending multiple times the same content over WAN links. Virtual machine data is retrieved in the destination site with distributed content-based addressing. We implemented a prototype of Shrinker in the KVM hypervisor [12] and present a performance evaluation in a distributed environment. Our experiments show that it reduces both total data transferred and total migration time.

This paper is organized as follows. Section 2 presents related work. Section 3 covers the architecture of Shrinker. Section 4 describes our prototype implementation in the KVM hypervisor, presents our experiments and analyzes the results. Finally, Section 5 concludes and discusses future work.

2 Background and Related Work

Live virtual machine migration [4,20] has traditionally been implemented with pre-copy [27] algorithms. Memory pages are transferred to the destination host

while the virtual machine is still executing. Live migration continues sending modified memory pages until it enters a phase where the virtual machine is paused, the remaining pages are copied, and the virtual machine is resumed on the destination host. This phase is responsible for the downtime experienced during live migration, which should be kept minimal. This downtime can range from a few milliseconds to seconds or minutes, depending on page dirtying rate and network bandwidth [1].

Numerous works have been proposed to improve live migration and optimize its performance metrics: total data transferred, total migration time, and downtime. They can be classified in two types: optimizations of the pre-copy approach, and alternatives to pre-copy. Optimizations include compression, delta page transfer, and data deduplication. Compression is an obvious method to reduce the amount of data transferred during live migration. Jin et al. [11] use adaptive compression to reduce the size of migrated data. Their system chooses different compression algorithms depending on memory page characteristics. Delta page transfer [6,32,26] optimizes the transmission of dirtied pages by sending difference between old pages and new pages, instead of sending full copies of new pages. Data deduplication [34,32] detects identical data inside the memory and disk of a single virtual machine and transfers this data only once.

An alternative to pre-copy is live migration based on post-copy [8,9], which first transfers CPU state and resumes the VM on the destination host. Memory pages are then fetched from the source host on demand, in addition to a background copying process to decrease the total migration time and quickly remove the residual dependency on the source host. Similarly, SnowFlock [13] uses demand-paging and multicast distribution of data to quickly instantiate VM clones on multiple hosts. Another alternative was proposed by Liu et al. [15], based on checkpointing/recovery and trace/replay. By recording nondeterministic events and replaying them at the destination, live migration efficiency is greatly improved. However, their approach is not adapted to migrate SMP guests.

All these systems were designed in the context of live migration of single VMs, and do not take advantage of data duplicated across multiple VMs. However, previous research [30,5,33,17] has shown that multiple VMs have significant portions of their memory containing identical data. This identical data can be caused by having the same versions of programs, shared libraries and kernels used in multiple VMs, or common file system data loaded in buffer cache. This identical data can be leveraged to decrease memory consumption of colocated VMs by sharing memory pages between multiple VMs [30,5,33,17]. The same observation has been made for VM disks [21,23,14], which is exploited to decrease storage consumption. To our knowledge, Sapuntzakis et al. [25] were the first to leverage identical data between multiple VMs to improve migration. However, their work predates live migration and supported only suspend/resume migration, where the VM is paused before being migrated. Additionally, they only took advantage of data available on the destination node, limiting the possibility of finding identical data. A similar approach was also proposed by Tolia et al. [28].

To allow live migration of VMs over WANs, two issues need to be solved: lack of shared storage and relocation to a different IP network. Storage can be migrated with algorithms similar to memory migration [3,16,10,32]. Network connections can be kept uninterrupted after migration using tunneling [29,3], VPN reconfiguration [32] or Mobile IPv6 [7]. The architecture of Shrinker supports storage migration. Additionally, since Shrinker focuses solely on optimizing data transmission during live virtual cluster migrations over WAN, it is compatible with all solutions for keeping network connections uninterrupted. Therefore, we do not discuss further this issue, as it is out of scope of this paper.

3 Architecture of Shrinker

We propose Shrinker, a system that improves live migration of virtual clusters over WANs by decreasing total data transferred and total migration time. During live migration of a virtual cluster between two data centers separated by a WAN, Shrinker detects memory pages and disk blocks duplicated among multiple virtual machines and transfers identical data only once. In the destination site, virtual machine disks and memory are reconstructed using distributed content-based addressing. In order to detect identical data efficiently, Shrinker leverages cryptographic hash functions. These functions map blocks of data, in our case memory pages or disk blocks, to hash values of fixed size, also called digests. These functions differ from ordinary hash functions because they are designed to render practically infeasible to find the original block of data from a hash value, modify a block of data while keeping the same hash value, and find two different blocks of data with the same hash value. Using hash values to identify memory pages and disk blocks by content is interesting because hash values are much smaller than the data they represent. For instance, a 4 kB memory page or disk block is mapped to a 20 bytes hash value using the SHA-1 [19] cryptographic hash function, a size reduction of more than 200 times.

We first present the architecture of Shrinker, and then discuss security considerations caused by our use of cryptographic hash functions.

3.1 Architecture Overview

In a standard live migration of a virtual cluster composed of multiple VMs running on different hypervisors, each live VM migration is independent. VM content is transferred from each source host to the corresponding destination host, with no interaction whatsoever between the source hypervisors or between the destination hypervisors. As a consequence, when migrating a virtual cluster over a WAN, data duplicated across VMs is sent multiple times over the WAN link separating the source and destination hypervisors.

To avoid this duplicated data transmission, Shrinker introduces coordination between the source hypervisors during the live migration process. This coordination is implemented by a service, running in the source site, that keeps track of which memory pages and disk blocks have been sent to the destination site.

Before sending a memory page or a disk block, a source hypervisor computes the hash value of this data and queries the service with this hash value. If no memory page or disk block with the same hash value has previously been sent to the destination site, the service informs the hypervisor that it should transfer the data. The service also registers the hash value in its database. Later, when the service receives a subsequent query for another memory page or disk block with the same hash value, it informs the querying hypervisor that the data has already been sent to the destination site. Based on this information, the hypervisor sends the hash value of the data to the destination host instead of the real content. This mechanism essentially performs data deduplication by replacing duplicated transmissions of memory pages and disk blocks by transmissions of much smaller hash values. Figure 1 illustrates a live migration between two pairs of hypervisors, with a cryptographic hash function creating 16-bit hash values.[1]

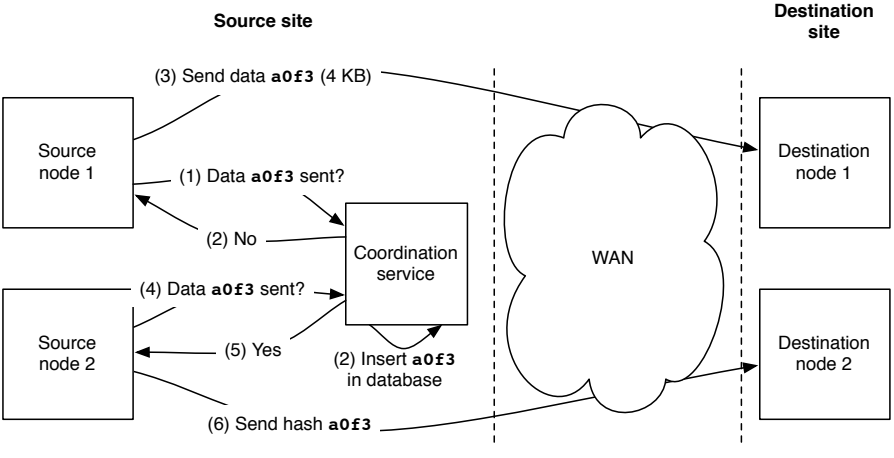


Fig. 1. Distributed data deduplication used in Shrinker to avoid sending duplicate data on a WAN link

Source node 1 has to send a memory page or disk block identified by the hash value `a0f3`. Instead of sending it directly like in a standard live migration, it first queries the coordination service (step 1). Since this is the first time that this data has to be sent to the destination site, the service informs source node 1 that it should send the data. It also updates its internal database to keep track of this hash value (step 2). After receiving the answer from the coordination service, source node 1 sends the data to destination node 1 (step 3). Afterwards, source node 2 queries the service for the same data (step 4). The service informs

[1] Note that this small hash value size is chosen to improve readability of the figure. In a real scenario, Shrinker can not use such hash function, since 65,536 different hash values would likely create collisions even for virtual machines of moderate sizes.

source node 2 that this data has already been sent to a node in the destination site (step 5), which prompts source node 2 to send the hash value `a0f3` (2 bytes) instead of the full content (4 kB) to destination node 2 (step 6).

Since destination nodes receive a mix of VM data (memory and disk content) and hash values from source nodes, they need to reconstruct the full VM content before the VM can be resumed. This is where distributed content-based addressing is used. When a full memory page or disk block is received by a destination node, its hash value and the IP of the node are registered into an indexing service, running in the destination site. When destination nodes receive hash values from source nodes, they query the indexing service to discover a node that has a copy of the content they are requesting. After contacting a node and receiving a copy of the content, they register themselves in the indexing service for this data, which increases the number of hosts capable of sending this data to another hypervisor. This process is illustrated in Fig. 2.

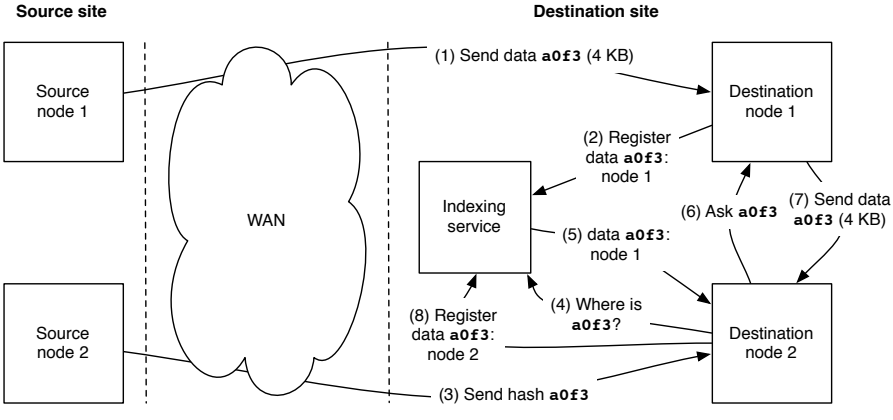


Fig. 2. Distributed content-based addressing used in Shrinker to reconstruct VM content on destination nodes

As in Fig. 1, destination node 1 receives a memory page or disk block identified by hash `a0f3` (step 1). It registers itself as having a copy of this data in the indexing service (step 2). Afterwards, as in Fig. 1, source node 2 sends hash value `a0f3` to destination node 2 (step 3). In order to get the data identified by this hash value, destination node 2 queries the indexing service to discover a host which has a copy of the corresponding content (step 4). The indexing service informs it that destination node 1 has a copy of this data (step 5). Destination node 2 asks destination node 1 for this content (step 6), receives it (step 7), and registers itself in the indexing server as having a copy (step 8).

Since the live migration is performed while the virtual machine is still executing, it is possible that data that was sent to the destination site has since been overwritten and is not accessible in any destination hypervisor. All

destination hypervisors open a communication channel with their corresponding source hypervisor. This channel can be used to directly request data to the source node.

3.2 Security Considerations

The possible number of different 4 kB memory pages and disk blocks (2^{4096}) is bigger than the number of possible hash values (2^{160} for SHA-1). As a consequence, the use of cryptographic hash functions opens the door to collisions: it is theoretically possible that memory pages and disk blocks with different content map to an identical hash value. However, the properties offered by cryptographic hash functions allow us to use these hash values with a high confidence. The probability p of one or more collisions occurring is bounded by (1), where n is the number of objects in the system and b the number of bits of hash values [22]:

$$p \leq \frac{n(n-1)}{2} \times \frac{1}{2^b} \ . \tag{1}$$

If we consider a very large virtual cluster consisting of 1 exabyte (2^{60} bytes) of 4 kB memory pages and disk blocks migrated by Shrinker using the SHA-1 hash function, the collision probability is around 10^{-20}. This is considered to be much less than other possible faults in a computing system, such as data corruption undetectable by ECC memory. However, (1) gives the probability of an accidental collision. Although the theoretical number of operations to find a collision is approximately $2^{\frac{n}{2}}$ (birthday attack), attackers can exploit weaknesses of the hash function algorithm to find collisions more easily. For example, researchers have shown attacks against SHA-1 that decrease the number of operations to find collisions from 2^{80} to 2^{69} [31]. Assuming that finding collisions is possible, an attacker capable of storing arbitrary data in VMs (for instance, by acting as a client interacting with web servers) could inject colliding data in these VMs. After migrating them with Shrinker, memory content would be corrupted because two different pages would have been replaced by the same data. If such attacks become practically feasible in the future, Shrinker can use stronger cryptographic hash functions, such as those from the SHA-2 family. Even though these hash functions are more computationally expensive, Shrinker remains interesting as long as the hash computation bandwidth is larger than the network bandwidth available to each hypervisor.

4 Implementation and Performance Evaluation

In this section, we first describe the implementation of our Shrinker prototype. Then, we present and analyze the results of our experiments performed on the Grid'5000 testbed.

4.1 Implementation

We implemented a prototype of Shrinker in the KVM hypervisor [12]. KVM is divided in two main parts. The first part is composed of two loadable kernel

modules providing the core virtualization infrastructure. The second part is a modified version of QEMU [2] running in user space to provide higher level features, including virtual device emulation and live migration. This prototype is fully implemented in the user space component of KVM version 0.14.0-rc0 and is about 2,000 lines of C code. We use Redis [24] version 2.2.0-rc4, a high performance key-value store, to implement the coordination and indexing service. Additional dependencies are OpenSSL, used to compute hash values, Hiredis, a C client library for Redis, and libev, a high-performance event loop library.

Support for data deduplication of storage migration in our prototype is not yet fully finalized. As such, in the following experiments, we use the storage migration mechanism of KVM, with copy-on-write images. The use of copy-on-write images allows KVM to send only storage data that has been modified since the boot of the VM, minimizing the amount of storage data to migrate. We expect our finalized prototype to offer even greater amounts of data deduplication for storage than for memory, since VM disks typically present large amounts of identical data, as shown in previous work (c.f. Sec. 2).

As explained in Sec. 3, before sending a memory page, source hypervisors need to contact the coordination service to know if the same page as already been sent to the destination site. Performing this query in a sequential manner would drastically reduce the live migration throughput (because of the round-trip time between source hypervisors and the coordination service). To overcome this problem, our implementation performs these queries in a pipelined manner. Queries for multiple memory pages are sent in parallel, and the decision of sending the full content or the hash is made when the answer is received. The same method is used on the destination site for hypervisors to get memory content.

The coordination service is implemented using the Redis `SETNX` command, with a memory page hash as key. If the hash is not already known by the service, it is registered and the return value notifies the source hypervisor that it was the first to register it. Otherwise, no change is performed in the service and the source hypervisor is notified that the hash was already registered.

The indexing service is implemented using the set data structure of Redis. For each registered memory page, there is one corresponding set which holds information about hypervisors having a copy of the page. When destination hypervisors register a memory page, they send a `SADD` command with the page hash as key and an IP/port pair as value. When other destination hypervisors need to get a page, they send a `SRANDMEMBER` command, which selects a random hypervisor in the set corresponding to the queried page and returns its IP/port information. After connecting on this IP and port, they query the content of the page. Finally, when a destination hypervisor doesn't hold any more copy of a page, it unregisters it with a `SREM` command.

4.2 Evaluation Methodology

All the experiments presented in this paper are run on the paradent cluster of the Grid'5000 site of Rennes. We use Carri System CS-5393B nodes supplied with 2 Intel Xeon L5420 processors (each with 4 cores at 2.5 GHz), 32 GB of

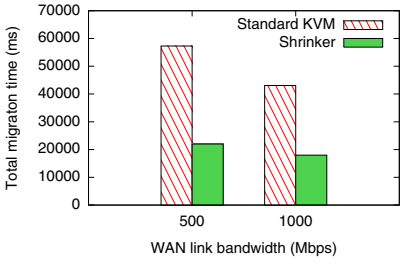
memory, and Gigabit Ethernet network interfaces. Physical nodes and VMs use the AMD64 port of Debian 5.0 (Lenny) as their operating system, with a 2.6.32 Linux kernel from the Lenny backports repository. VMs are configured to use 1 GB of memory. Two instances of Redis are running, each on a dedicated node.

To study the performance of our prototype, we configured a dedicated node to act as a router between source and destination hypervisors. Using the `netem` Linux kernel module, this router emulates wide area networks with different bandwidth rates. The emulated latency is set to a round trip time of 20 ms.

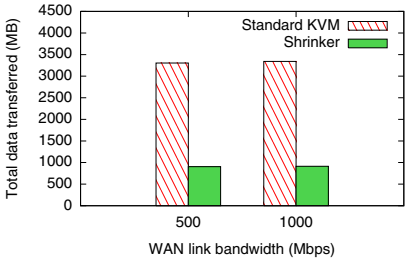
For our performance evaluation, we use a program from the NAS Parallel Benchmarks (NPB) [18], which are derived from computational fluid dynamics applications. We evaluate performance metrics of live migration during the execution of this workload. We measure total migration and total data transmitted by live migration. During our experiments, we discovered that the downtime caused by a live migration in KVM was overly important because of a miscalculation of the available bandwidth. This is why we do not report numbers for downtime, as they may be not representative of a correct behavior. We are investigating the issue and will report it to the KVM community.

4.3 Performance Results

Figure 3(a) shows the total migration time of a 16 VMs cluster executing the ep.D.16 program, for two different bandwidth rates. Figure 3(b) shows the total amount of transmitted data during the live migration of these 16 VMs, for the same two bandwidth rates.



(a) Average total migration time of 16 VMs running ep.D.16



(b) Total data transmitted over the WAN link during the live migration of 16 VMs running ep.D.16

Fig. 3. Performance evaluation of Shrinker

First, we observe that, whatever the bandwidth, the same amount of data is transmitted for both bandwidth rates. However, we can see that Shrinker sends much less data over the WAN link than the standard KVM migration. This allows Shrinker to reduce the total migration time, for instance from one minute to 20 seconds for the 500 Mbps link. Note that in the regular KVM live migration

protocol, memory pages containing only identical bytes are already compressed and sent efficiently. As such, the improvement showed by Shrinker come from real data deduplication, and not from deduplication of zero pages.

5 Conclusion

In this paper, we presented the design and prototype implementation of Shrinker, a system that improves live migration of virtual clusters over WANs by decreasing total data transferred and total migration time. Shrinker detects memory pages and disk blocks duplicated among multiple virtual machines to transfer identical data only once. Virtual machine data is retrieved in the destination site with distributed content-based addressing. We implemented a prototype of Shrinker in the KVM hypervisor. Our experiments show that it reduces both total data transferred and total migration time.

In the future, we will finalize our Shrinker prototype to perform data deduplication on storage migration and evaluate its performance. We also plan to study how Shrinker can allow destination hypervisors to fetch data from local VM image repositories found in most cloud computing infrastructures to further decrease the amount of data sent over WAN links.

Acknowledgments. Experiments presented in this paper were carried out using the Grid'5000 experimental testbed, being developed under the INRIA ALADDIN development action with support from CNRS, RENATER and several Universities as well as other funding bodies (see https://www.grid5000.fr).

References

1. Akoush, S., Sohan, R., Rice, A., Moore, A.W., Hopper, A.: Predicting the Performance of Virtual Machine Migration. In: International Symposium on Modeling, Analysis, and Simulation of Computer Systems (MASCOTS 2010), pp. 37–46 (2010)
2. Bellard, F.: QEMU, a Fast and Portable Dynamic Translator. In: Proceedings of the 2005 USENIX Annual Technical Conference (USENIX 2005), pp. 41–46 (2005)
3. Robert, B., Evangelos, K., Anja, F., Harald, S.: Live wide-area migration of virtual machines including local persistent state. In: Proceedings of the 3rd International Conference on Virtual Execution Environments (VEE 2007), pp. 169–179 (2007)
4. 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 Symposium on Networked Systems Design & Implementation (NSDI 2005), pp. 273–286 (2005)
5. Gupta, D., Lee, S., Vrable, M., Savage, S., Snoeren, A.C., Varghese, G., Voelker, G.M., Vahdat, A.: Difference Engine: Harnessing Memory Redundancy in Virtual Machines. In: 8th USENIX Symposium on Operating Systems Design and Implementation (OSDI 2008), pp. 309–322 (2008)
6. Hacking, S., Hudzia, B.: Improving the live migration process of large enterprise applications. In: Proceedings of the 3rd international Workshop on Virtualization Technologies in Distributed Computing (VTDC 2009), pp. 51–58 (2009)

7. Harney, E., Goasguen, S., Martin, J., Murphy, M., Westall, M.: The Efficacy of Live Virtual Machine Migrations Over the Internet. In: Proceedings of the 3rd International Workshop on Virtualization Technology in Distributed Computing (VTDC 2007), pp. 1–7 (2007)
8. 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 2009), pp. 51–60 (2009)
9. Hirofuchi, T., Nakada, H., Itoh, S., Sekiguchi, S.: Enabling Instantaneous Relocation of Virtual Machines with a Lightweight VMM Extension. In: Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid 2010), pp. 73–83 (2010)
10. Hirofuchi, T., Ogawa, H., Nakada, H., Itoh, S., Sekiguchi, S.: A Live Storage Migration Mechanism over WAN for Relocatable Virtual Machine Services over Clouds. In: Proceedings of the 9th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2009), pp. 460–465 (2009)
11. Jin, H., Deng, L., Wu, S., Shi, X., Pan, X.: Live Virtual Machine Migration with Adaptive Memory Compression. In: Proceedings of the 2009 IEEE International Conference on Cluster Computing, Cluster 2009 (2009)
12. Kivity, A., Kamay, Y., Laor, D., Lublin, U., Liguori, A.: kvm: the Linux Virtual Machine Monitor. In: Proceedings of the 2007 Linux Symposium, vol. 1, pp. 225–230 (June 2007)
13. Lagar-Cavilla, H.A., Whitney, J.A., Scannell, A.M., Patchin, P., Rumble, S.M., Lara, E.d., Brudno, M., Satyanarayanan, M.: SnowFlock: rapid virtual machine cloning for cloud computing. In: Proceedings of the 4th ACM European Conference on Computer Systems (EuroSys 2009), pp. 1–12 (2009)
14. Liguori, A., Hensbergen, E.V.: Experiences with Content Addressable Storage and Virtual Disks. In: Proceedings of the First Workshop on I/O Virtualization, WIOV 2008 (2008)
15. Liu, H., Jin, H., Liao, X., Hu, L., Yu, C.: Live migration of virtual machine based on full system trace and replay. In: Proceedings of the 18th ACM International Symposium on High Performance Distributed Computing (HPDC 2009), Garching, Germany, pp. 101–110 (2009)
16. Luo, Y., Zhang, B., Wang, X., Wang, Z., Sun, Y., Chen, H.: Live and incremental whole-system migration of virtual machines using block-bitmap. In: 2008 IEEE International Conference on Cluster Computing (Cluster 2008), pp. 99–106 (2008)
17. Milos, G., Murray, D.G., Hand, S., Fetterman, M.: Satori: Enlightened Page Sharing. In: Proceedings of the 2009 USENIX Annual Technical Conference, USENIX 2009 (2009)
18. NASA Advanced Supercomputing Division: NAS Parallel Benchmarks, `http://www.nas.nasa.gov/Software/NPB/`
19. National Institute of Standards and Technology: Secure Hash Standard (April 1995)
20. Nelson, M., Lim, B.-H., Hutchins, G.: Fast Transparent Migration for Virtual Machines. In: Proceedings of the 2005 USENIX Annual Technical Conference (USENIX 2005), pp. 391–394 (2005)
21. Partho, N., Kozuch, M.A., O'Hallaron, D.R., Harkes, J., Satyanarayanan, M., Tolia, N., Toups, M.: Design tradeoffs in applying content addressable storage to enterprise-scale systems based on virtual machines. In: Proceedings of the 2006 USENIX Annual Technical Conference (USENIX 2006), pp. 1–6 (2006)

22. Quinlan, S., Dorward, S.: Venti: A New Approach to Archival Storage. In: Proceedings of the Conference on File and Storage Technologies (FAST 2002), pp. 89–101 (2002)
23. Rhea, S., Cox, R., Pesterev, A.: Fast, inexpensive content-addressed storage in foundation. In: Proceedings of the 2008 USENIX Annual Technical Conference (USENIX 2008), pp. 143–156 (2008)
24. Sanfilippo, S.: Redis, `http://redis.io`
25. Sapuntzakis, C.P., Chandra, R., Pfaff, B., Chow, J., Lam, M.S., Rosenblum, M.: Optimizing the migration of virtual computers. In: Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI 2002), pp. 377–390 (2002)
26. Svärd, P., Hudzia, B., Tordsson, J., Elmroth, E.: Evaluation of Delta Compression Techniques for Efficient Live Migration of Large Virtual Machines. In: Proceedings of the 7th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE 2011), pp. 111–120 (2011)
27. Theimer, M.M., Lantz, K.A., Cheriton, D.R.: Preemptable remote execution facilities for the V-system. In: Proceedings of the Tenth ACM Symposium on Operating Systems Principles (SOSP 1985), pp. 2–12 (1985)
28. Tolia, N., Bressoud, T., Kozuch, M., Satyanarayanan, M.: Using Content Addressing to Transfer Virtual Machine State. Tech. rep., Intel Corporation (2002)
29. Travostino, F., Daspit, P., Gommans, L., Jog, C., de Laat, C., Mambretti, J., Monga, I., van Oudenaarde, B., Raghunath, S., Wang, P.Y.: Seamless live migration of virtual machines over the MAN/WAN. Future Gener. Comput. Syst. 22(8), 901–907 (2006)
30. Waldspurger, C.A.: Memory resource management in VMware ESX server. In: Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI 2002), pp. 181–194 (2002)
31. Wang, X., Yin, Y.L., Yu, H.: Finding Collisions in the Full SHA-1. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 17–36. Springer, Heidelberg (2005)
32. Wood, T., Ramakrishnan, K., Shenoy, P., van der Merwe, J.: CloudNet: Dynamic Pooling of Cloud Resources by Live WAN Migration of Virtual Machines. In: Proceedings of the 7th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, VEE 2011 (2011)
33. Wood, T., Tarasuk-Levin, G., Shenoy, P., Desnoyers, P., Cecchet, E., Corner, M.: Memory Buddies: Exploiting Page Sharing for Smart Colocation in Virtualized Data Centers. In: Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE 2009), pp. 31–40 (2009)
34. Zhang, X., Huo, Z., Ma, J., Meng, D.: Exploiting Data Deduplication to Accelerate Live Virtual Machine Migration. In: IEEE International Conference on Cluster Computing (Cluster 2010), pp. 88–96 (2010)