Efficient Storage Synchronization for Live Migration in Cloud Infrastructures

Katharina Haselhorst, Matthias Schmidt, Roland Schwarzkopf, Niels Fallenbeck, Bernd Freisleben
Department of Mathematics and Computer Science, University of Marburg
Hans-Meerwein-Str. 3, D-35032 Marburg, Germany
{haselhorst, schmidtm, rschwarzkopf, fallenbe, freisleb}@informatik.uni-marburg.de

***Abstract*— Live migration of virtual machines is an important issue in Cloud computing environments: when physical hosts are overloaded, some or all virtual machines can be moved to a less loaded host. Live migration poses additional challenges when virtual machines use local persistent storage, since the complete disk state needs to be transferred to the destination host while the virtual machines are running and hence are altering the disk state. In this paper, several approaches for implementing and synchronizing persistent storage during live migration of virtual machines in Cloud infrastructures are presented. Furthermore, the approaches also enable users to migrate swap space, which is currently not possible on most virtual machine hypervisors. Finally, measurements regarding disk synchronization, migration time and possible overheads are presented.**

I. Introduction

In the last few years, a shift from virtual machines running on privately owned and operated hardware to Cloud computing setups where machines and computing power are rented from remote providers on demand can be observed. In fact, the notion of Cloud computing is tightly coupled to the technology of virtualization: computing resources are provided to customers in the form of virtual machines. This is called Infrastructure as a Service (IaaS), as implemented by Amazon's Elastic Compute Cloud [1] or Eucalyptus [9].

Load balancing is an important issue in Cloud computing installations, and the basic enabling technology is the migration of virtual machines: when physical hosts are overloaded, some or all virtual machines can be moved to a less loaded host. A particular type of migration is live migration where virtual machines continue running during the migration process, such that no downtimes can be observed by clients (e.g. network connections are kept open). Thus, live migration is completely transparent from the customer's point of view. Live migration poses additional challenges when the virtual machines use local persistent storage: if no residual dependencies to the source host should remain after migration, the complete disk state needs to be transferred to the destination host while the virtual machines keep running and hence are altering the disk state.

Usually, each virtual machine (VM) needs persistent storage to work. This includes the base operating system (which in case of Amazon, Eucalyptus and the XGE [14], [12] is Linux), the user's own applications and the user's data on which the applications operate. However, unlike traditional machines, all data written during the lifetime of a VM in a Cloud environment is temporary in the sense that it is disregarded after shutdown of the VM. Thus, the disk space of a VM is naturally divided into two parts: a basic part containing the base operating system and a writable part containing all new data written during VM operation. Live migration is simplified if the basic part is never altered: in this case, it can be copied to the destination host without losing updates issued by the VM during the migration process; only the writable part has to be taken care of.

In this paper, several approaches for implementing and synchronizing persistent storage during VM live migration in Cloud infrastructures are presented. This includes a temporary in-memory file system, a real disk image and a hybrid approach. A disk synchronization mechanism originally designed for high availability (HA) clusters is leveraged to ensure seamless and transparent storage synchronization prior to and during VM migration. Furthermore, the approach also enables swap space migration. Most current VM hypervisors (e.g. the Xen Virtual Machine Monitor) do not support live migration of swap space, i.e. all VMs have to run without swap space, which is a major drawback, especially for memory intensive applications. The presented approaches are implemented in the XGE [14], [12] that provides a basic infrastructure for managing a cluster of virtual machines: machines can be started and stopped per job, and user specific disk images can be deployed to the corresponding physical locations before the machines are started. The XGE offers a live migration facility to migrate VMs during runtime between physical machines. The novel concept of VM image synchronization introduced in this paper enhances the current migration mechanism and allows cheaper and faster VM migrations to ease load balancing. Measurements regarding disk synchronization, migration time and possible overheads are presented.

The paper is organized as follows. Section II states the problems involved in virtual machine distribution. Section III discusses related work. Section IV presents the proposed storage synchronization approaches. Section V discusses implementation issues. Section VI presents results of several performance measurements. Section VII concludes the paper and outlines areas for future work.

II. Problem Statement

In Cloud computing environments where many VMs dynamically (i.e. they are created and destroyed over time) run

on clusters of physical hosts, the possibility of migrating VMs between differed hosts is an essential feature: it enables dynamic load balancing, energy efficient machine utilization, and ease of maintenance. An important property of the employed migration mechanism is transparency for the VMs: a user should not notice that his or her VM is being migrated, and the VM's operation should continue seamlessly during the migration process.

Most current live migration implementations (e.g., the Xen migration facility) do not take disk storage (including swap space on disk) into account. Disk storage is assumed to be located on a shared medium that can be accessed from both the source and the destination host involved in a live migration.

This approach leads to several problems, especially in a Cloud environment:

- A disk access over a network always introduces a performance penalty compared to a local disk access.
- The VM depends on a shared storage facility and a functioning network connection to work properly.
- Each VM instance needs its own working copy of a disk image due to local modifications, although in a Cloud environment many VMs might share the same basic image.

A solution that is better suited to the requirements of a Cloud environment should allow a VM to access its disk storage locally during normal operation. This implies that a live storage migration that only affects the VM performance during a migration process in a negligible way is needed in addition to the migration of main memory and CPU state. Furthermore, it is desirable to exploit the sharing of basic disk images between VMs to reduce the number of copies.

III. Related Work

A. VM Migration

One of the first works considering migration of VMs has been published by Kozuch et al. [6]. The basic motivation of the proposal is mobility of users: a VM is suspended on the source hardware and resumed on the destination host. Local state, memory pages and persistent storage are copied using distributed filesystems like NFS or Coda [11]. While this approach targets users who want to continue their work on different locations without the need for carrying the physical hardware (like a laptop) with them, it does not fit into a high-performance computing environment where the migration should have a minimal impact on the migrating VM. Nevertheless, a slightly modified version of the approach that copies directly to the target machine instead of using shared storage would be a very simple and efficient solution for scenarios tolerating longer downtimes of VMs (e.g. VMs running batch jobs).

B. Live Migration in Local Networks

Fundamental research on VM live migration was done by Clark et al. [3]. Their approach is based on the observation that write accesses to main memory are concentrated to a small number of pages (the 'writable working set') for a certain time period. Hence, they pre-copy the memory pages to the destination host while the VM keeps running on the source host. Dirty memory pages are re-transferred in several iterations until the remaining set of dirty pages is small enough or a maximum number of iterations is reached. Then, the VM is suspended, all dirty pages and the internal state are copied, and the VM is resumed on the target host. This solution has been designed for live migration in local networks assuming shared storage between the VMs, and it has been adopted by Xen.

An alternative approach is taken by Hines et al. [5]: instead of pre-copying the memory pages before migration, only the internal state is transmitted, and the VM is resumed immediately at the destination. Memory pages are copied on demand as well as by prepaging techniques, reducing the number of costly page faults over the network. The authors showed that the post-copy approach can improve some metrics, such as the total number of transferred pages, the total migration time and the network overhead compared to the pre-copy approach. An interesting mechanism is the 'dynamic self-ballooning' method introduced in their work: unused memory pages can be reclaimed from the VM, which reduces the total number of pages that must be copied during a migration process and thus yielding a migration speed up.

Voorsluys et al. [16] have extensively analyzed and evaluated the cost of VM live migration in a Cloud environment. A number of case studies with representative workloads were executed to measure the performance impact on the applications running inside a VM during live migration. The results show that the impact of live migration on a heavy loaded VM cannot be neglected especially when service level agreements need to be met. Live migration on VMs with a slightly reduced load are relatively uncritical, and the most stringent service level agreements (99th percentile) can still be met.

C. Storage Migration

Bradford et al. [2] and Luo et al. [8] propose mechanisms for entire system migration including local persistent state. Both use special disk drivers intercepting disk writes and reads to make the transfer of the local state possible during live migration.

Bradford et al. [2] use a two-phase transfer: first, the complete disk image is copied from source to destination while all new write operations are recorded and sent as 'deltas' to the destination in parallel. The second phase consists of migrating the main memory and the CPU state via the Xen migration mechanism and to apply all deltas to the disk image on the destination. The VM is resumed once Xen migration has finished, but all I/O operations must be blocked until the application of the deltas is complete. The authors showed that in the majority of cases the application of deltas is finished before the Xen migration and hence no additional downtime is observed. Nevertheless, there are workloads (especially write intensive workloads) that can cause a significant increase of downtime. In addition, write intensive workloads need to be throttled during migration if the write rate exceeds the network bandwidth, which would otherwise prevent the migration process from progressing.

Luo et al. [8] propose a three-phase migration approach. In the pre-copy phase, the disk image is copied incrementally to the destination host (similar to the mechanism used for transferring the memory pages during Xen live migration). Dirty blocks are tracked using a block bitmap and retransferred in subsequent iterations. In the next phase, the VM is suspended, the block bitmap is copied to the destination and the VM is resumed immediately without transferring the remaining dirty blocks. In the third phase - the post-copy phase - all disk accesses are intercepted by a special disk driver and the missing blocks are copied from the source host on demand. The main drawback of this solution is its dependence on the source host. If the source host (or the network connection between source and destination) crashes before all remaining dirty blocks are transferred, the VM cannot continue to run since the disk image is not consistent. Such a situation can not occur in Bradford et al.'s approach [2].

Sapuntzakis et al. [10] present a whole-machine migration process including local storage. They use so called 'capsules' that encapsulate a machine's state and can be moved between different machines. The main memory and CPU state is transferred offline (the machine is suspended, the state is copied to the destination and the machine is resumed), which takes several minutes even over a fast link and thus renders this approach useless for live migration scenarios in Cloud environments. Disk images are copied only on demand, reducing the total migration time at the price of a possibly infinite dependence on the source host.

D. Image Deployment

Most research on disk image deployment was conducted in the context of VM migration. During the migration process, the deployment usually consists of a one-to-one relation: the disk images need to be transferred from the source to the destination. However, this area represents only a small fraction of use cases for image deployment. Schmidt et al. [12] analyzed and compared different approaches for distributing images to a large number of destinations in a Cloud environment.

IV. Design

The proposed approach to satisfy the the requirements stated in Section II is divided into several parts:

A. Multilayer Disk Images

Usually, each compute node needs persistent storage to operate, containing user data, a custom software stack, etc. All data written during the lifetime of a VM in a Cloud environment is temporary, since it is disregarded after shutdown. Thus, the disk space of a VM is divided into two parts: a basic part containing the base operating system and a writable part containing all new data written during VM operation. Live migration is simplified if the base part is never altered, i.e. it can be copied to the destination host without losing updates issued by the VM during migration; only the writable part has to be considered.

However, a write operation does not only write new data to disk but might as well modify or delete existing data from the basic part. For example, the file `/etc/mtab` will be altered on each mount call. Hence, the two parts need to be merged in some way. These requirements are perfectly met by a union file system, as implemented, for example, by aufs or unionfs (which has been deprecated). Our previous work [13] showed that a union filesystem can be used as a root file system.

There are several possibilities to create a layered root filesystem:

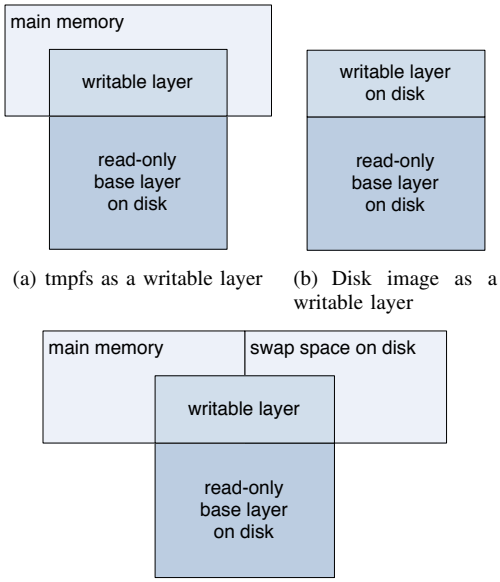
- The most simple way is to use a temporary in-memory file system (tmpfs) as a writable layer (see Figure 1(a)). The main advantage of this approach is that the setup is transparent to live migration: since all data resides in main memory, it is transferred to the destination host during the normal memory copy process without taking any special measures. A positive side effect from the VM's point of view is the increased I/O performance for disk accesses. For this setup to work properly, a sufficiently large amount of RAM needs to be allocated to the VM. However, there are two kinds of workloads that are not well suited for this approach: memory intensive workloads requiring a very large amount of RAM and workloads writing much data to disk (and possibly never - or at least in rare occasions - accessing it again).
- Another possibility is to use a real disk image as a writable layer (see Figure 1(b)). It is better suited to workloads producing lots of disk output. The disk writes do not clutter up the main memory.
- A hybrid approach is a further possibility: instead of using a disk image as a writable layer, a large tmpfs is provided in conjunction with a large swap partition (residing on disk). Figure 1(c) illustrates the resulting setup. This has the advantage of supporting memory intensive as well as disk write intensive workloads. All data written to the writable layer and not accessed again for some time will be swapped out to disk and thus the main memory does not get filled up with unused file system content.

The challenge with the last two approaches is that they are not transparent to live migration. Current VM hypervisor implementations like Xen do not support live migration of VMs with local persistent storage but assume shared storage (including possibly swap space) that is accessible from both the source and the destination host. However, for performance reasons it is desirable that a VM has local access to its disk images. Hence, a mechanism is needed for seamlessly transferring the disk images to a new host, as presented below.

B. Disk Image Synchronization

Using local persistent storage poses some challenges when performing live migration of VMs. The main problem is to transfer a consistent state of the disk to the destination host while the VM keeps running and thus is altering the disk state. Hence, the task is divided into two parts: copying the data and tracking changes (and somehow send them to the destination host).

The synchronization mechanism proposed in this paper works as follows: at the beginning of the migration process, the source host starts to copy the disk image to the destination.



(a) tmpfs as a writable layer (b) Disk image as a writable layer

(c) tmpfs as a writable layer with swap space on disk

Fig. 1. Multilayer Disk Images

In parallel, all subsequent disk writes from the VM to be migrated are trapped and issued synchronously to the local and the remote disk image. Synchronously means that the acknowledgment of the disk write is delayed until the remote host has confirmed it. Once the background copy operation is finished, the normal live migration process is started, and during the entire live migration process, the two disk images operate in the synchronized mode. After the VM is resumed on the destination host, the disk images are decoupled, and no dependence on the source host remains.

To perform the actual synchronization, the DRBD [7] kernel module is used; it is integrated into the mainline Linux kernel since release 2.6.33. DRBD is designed for high availability clusters mirroring a disk from the primary host to a secondary backup host and thus acting as a network based RAID-1. Figure 2 shows the design of the module. It presents itself to the kernel as a disk driver and hence allows a maximum of flexibility: it does neither pose restrictions on the file system used above nor the underlying disk driver managing the actual disk accesses. And it is transparent to the kernel block device facilities, which means that buffering and disk scheduling are left to the kernel as usual. The module can operate in two modes: standalone and synchronized. In standalone mode, all disk accesses are simply passed to the underlying disk driver. In synchronized mode, disk writes are both passed to the underlying disk driver and sent to the backup machine via a TCP connection. Disk reads are served locally.

Although designed for a high availability setup where disks are mirrored across the network during normal operation, the DRBD module can be integrated into the live migration process for migrating local persistent storage. The source host plays the role of the primary server, and the destination host plays the role of the secondary server. During normal operation, the source host runs in standalone mode and hence

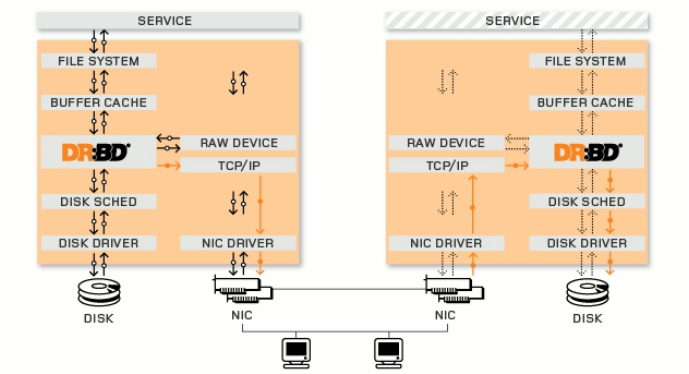


Fig. 2. DRBD module overview. Source: [7]

writes are performed only on its local disk, and there is no dependence on other hosts. During live migration, the DRBD driver is put into synchronized mode which causes all disk writes to be performed synchronously on both hosts while the entire disk is synchronized in the background. Once the migration is finished and the VM is resumed on the destination host, the DRBD driver on the destination host is put into standalone mode, and the source host is disconnected, removing all dependencies between the two physical hosts.

The properties of this approach are as follows:

- There is nearly no performance overhead during normal operation of a VM, because all disk writes are performed locally.
- The solution is reliable: if a migration fails, the VM can keep running on the source host, and due to the synchronous writes on both hosts, the VM has a consistent disk state on the destination host after a successful migration.
- There are no residual dependencies: once the VM is resumed on the destination host, no dependency on the source hosts remains. In particular, no disk writes are ever issued on an inconsistent disk (such as, for example, in the approach of Luo et al. [8]).
- The total migration time is increased compared to a memory-only migration. The additional time grows linearly with the disk size. The total migration time can be reduced as follows:
  - If a layered filesystem is used (as described above), the size of the writable layer can be kept small, reducing the amount of data to synchronize. The read-only layer can be fetched from a separate image pool in parallel or might even be already cached at the destination host so that no extra copy process is needed.
  - DRBD allows for checksum based synchronization, which means that only blocks that differ on source and destination are transferred. If sparse image files are used on both sides, all unused blocks are implicitly zero-filled and are hence identical on both sides, reducing the total amount of data to copy.
- Background disk synchronization and the transfer of the main memory are performed sequentially, so that they do

not affect each other in a counterproductive way: since both tasks generate network traffic, the memory dirtying rate would exceed the transfer rate (due to the parallel disk synchronization) much faster, resulting in the abort of the iterative copy phase and thus a longer downtime of the VM. Furthermore, disk synchronization usually takes much longer than the memory copy process and hence an exact timing would be difficult.

- No additional downtime of the VM is introduced, because the VM can be resumed without any further delay once the live migration process of the virtualization system in use has finished. In contrast, the approach of Bradford et al. [2] delays all disk I/O of the VM until the remaining writes are applied to the disk on the destination host. This can cause an additional delay for write-intensive workloads.
- Write intensive workloads are implicitly throttled by the synchronous nature of the disk writes, such that the disk write rate never exceeds the network transfer rate, which would render any disk synchronization mechanism useless. Bradford et al. [2] employ explicit write throttling whenever the write rate exceeds a predefined threshold. Luo et al. [8] stop their pre-copy phase proactively if the disk dirty rate is higher than the transfer rate, resulting in a much longer post-copy phase where the destination host still depends on the source host and the VM runs with decreased performance.
- The amount of bandwidth consumed by background synchronization can be dynamically configured in the DRBD driver. This enables the administrator to find an appropriate trade-off between total migration time and performance degradation of the VM due to high network consumption.
- Synchronization is completely transparent to the running VM.
- The approach is independent of a special virtualization environment, thus it can be used with any hypervisor that supports live migration.

V. Implementation

The synchronization mechanism using DRBD devices has been implemented in the XGE, an open source Virtual Machine Manager. The XGE is written in the programming language Python, and thus the controller that handles the synchronization mechanism has also been written in Python. The current implementation works with Xen as the backend hypervisor, although most of the code is not depending on Xen.

A. DRBD Device Configuration

A two node setup consists of a pair of DRBD devices that are identified by their path in the file system. Since the DRBD endpoints communicate via two separate TCP connections, they have to agree on port numbers on both sides. Hence, a DRBD endpoint is identified by a hostname, port number and path. To reduce the global configuration overhead, the nodes locally manage their resources (see next section). The actual device names are abstracted by symbolic links.

In our context, DRBD devices work in different modes throughout their lifetime: usually, a DRBD device runs in standalone mode as a pure bypass to the backing block device. In this mode, disk I/O performs nearly with native speed. In the pre-migration phase, the DRBD device on the source host is connected to the endpoint on the destination host, and the pair of devices runs in primary/secondary mode (only the source node is allowed to write to the device) during the initial synchronization. Just before the actual migration starts, the devices are put into primary/primary mode. This is necessary because Xen checks for write access on all associated block devices before initiating a live migration. From the devices' point of view, this mode allows both ends to issue write requests simultaneously. However, this will never happen in our setup due to the nature of a VM migration: a VM is always running on a single host and thus will always access the device only through one endpoint at a time.

B. Node Setup

Apart from kernel and RAM disk, a VM has one or more associated VM images: a read-only base layer (the user image), a writable disk layer (if it does not reside in memory) and possibly a separate VM image used as swap partition. VM images may be physical devices, logical volume manager (LVM) partitions or disk images, and the choice is left to the XGE configuration.

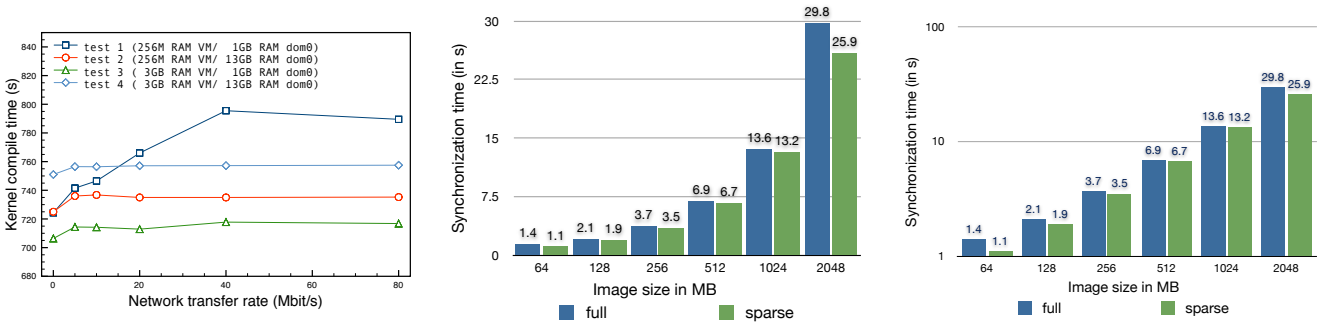
The base layer is never altered by the VM and hence all instances on one physical host using the same base image can share a single copy. When needed for the first time, the image is downloaded once (e.g., via BitTorrent) and cached in a local image pool. All writable VM images are attached to DRBD devices so that all I/O is intercepted by the DRBD driver. Each VM has a folder that contains (symbolic links to) all its VM images. Due to the symbolic links, the VM image names can easily be kept consistent within the cluster (which is important for live migration) without the need for all involved nodes to use the same actual device names (which would impose a large administrative overhead). On VM start, the writable VM images are empty.

Each physical node in the XGE cluster runs an image daemon (imaged). The image daemon is responsible for downloading and caching base images (the read-only layers) and for managing the DRBD devices.

C. Pre-Migration Process

When the XGE headnode wants to initiate the live migration of a VM from a source to a destination host, the following tasks are performed.

- The headnode contacts the image daemons on both hosts and instructs them to prepare the migration. The source hosts responds with the port numbers used by the DRBD devices of the corresponding VM images. The destination host reserves DRBD devices, attaches LVM partitions or disk images, chooses free ports to use for the synchronization and sends them back to the headnode as well.



(a) Impact of background synchronization limits on VM performance

(b) Disk synchronization

(c) Disk synchronization (logarithmic scale)

Fig. 3. Compile time (a) and disk synchronization measurements (b) + (c)

- The headnode communicates the configuration information to both hosts so that they can update their DRBD configuration accordingly and connect the corresponding DRBD endpoints. When the endpoints are connected, the synchronization starts in primary/secondary mode.
- Once the synchronization is finished, the DRBD devices on the destination host are put into primary mode, and the usual Xen live migration process is started.
- When the live migration has successfully terminated, the DRBD devices on the source host are disconnected so that the DRBD devices on the destination host run in standalone mode (with nearly native I/O speed). The corresponding resources (devices, ports, disk images) on the source host are freed.

Although the Xen hypervisor provides DRBD specific hooks that manage the mode transitions (primary/secondary to primary/primary and back) of DRBD based disk storage automatically during a live migration, we did not use them in our implementation. There are two reasons for this decision. First, the solution should be independent of the hypervisor and should not rely on Xen specific features. Second, in our implementation the association between the source and the target of a migration is dynamically established for the live migration. Hence, several configuration tasks have to be executed before actually being able to set the devices into primary/primary mode, which require a much finer grained control over the devices. The mode transitions are simply integrated into our implementation as a final step before migration.

VI. Experimental Results

In this section, the impact of disk synchronization on the total migration time and on the performance of the VM is evaluated. All participating physical nodes are AMD Dual-Core machines with 2.2 GHz, 16 GB RAM running Debian GNU/Linux. All nodes are interconnected with Gigabit Ethernet.

Figures 3(b) and 3(c) show the synchronization time of an idle disk image for several disk image sizes via the DRBD driver. The left bars represent the tests with *full disk images* (randomly filled from `/dev/urandom`), and the right bars show the corresponding tests with *sparse images* (which are observed as zero filled images by the synchronization partner). It is evident (especially from Figure 3(c) that draws the results over a logarithmic scale) that the synchronization time grows linearly with the disk size. The results also show that the synchronization time can be reduced by up to 13% when working with sparse images where only a small fraction of the size is actually in use.

As a core test, the kernel compile benchmark was used. This test measures the total time required to build a recent Linux kernel with the default configuration. The test was chosen because it represents a balanced workload stressing the virtual memory system, doing moderate disk I/O as well as being relatively CPU intensive. The kernel compile was done inside a VM while being live migrated to another host. Six tests with slightly different setups were performed:

1) 256 MB RAM in the VM with a 2 GB disk image attached to a DRBD driver and 1 G RAM in the dom0
2) 256 MB RAM in the VM with a 2 GB disk image attached to a DRBD driver and 13 G RAM in the dom0
3) 3 GB RAM in the VM with a 2 GB disk image attached to a DRBD driver and 1 GB RAM in the dom0
4) 3 GB RAM in the VM with a 2 GB disk image attached to a DRBD driver and 13 GB RAM in the dom0
5) 3 GB RAM in the VM with a tmpfs writable layer and 1 GB RAM in the dom0
6) 3 GB RAM in the VM with a tmpfs writable layer and 13 GB RAM in the dom0

The different RAM sizes in both the dom0 and the VM were chosen to evaluate the relationship between RAM size, performance and synchronization time.

In tests 1-4, the output of the compile was written to the disk image being synchronized via the DRBD module, in tests 5 and 6 the output went to tmpfs. The duration of the kernel compile, the synchronization and the Xen migration were measured, and all tests were repeated at least 50 times to get a robust mean. The kernel compile tests were also done without migration as a reference. The tests involving disk synchronization were performed with transfer rates of 5, 10, 20, 40 and 80 MByte/s for the background synchronization (a transfer rate configuration of more than 80 MByte/s did not result in a further synchronization speed up due to saturation

of hardware resources). The third proposed possibility for layering the read-only and the writable layer using a large swap space on disk (see Section IV) was not tested separately, because from the point of view of the synchronization there is no difference between synchronizing a disk image or a disk containing a swap partition.

The results of tests 1-4 are shown in Table I and Figure 3(a), Table II shows the results of tests 5 and 6, and in Table III, the reference values for all tests without migration are summarized.

Several observations can be made:

- The total performance degradation in the tests involving disk synchronization compared to the reference values ranges from 0.7% (in test 4) to 9% (in test 1 with maximum bandwidth utilization for background synchronization). These values are relative to one migration process per kernel compile. If the values are extrapolated to one migration per hour, the performance degradation ranges from 0.2% to 1.9%, which are acceptable values.
- The Xen live migration process itself has little impact on the performance of the migrating VM, as indicated when comparing the test results of tests 5 and 6 (see Table II) to their reference values. Thus, most of the observed overhead in the other tests can be assumed to be caused by disk synchronization.
- The migration time does not increase when using a tmpfs as a writable layer. Hence, the total migration time is reduced by about 90% compared to using a local disk image (when using a medium value of 20 Mbit/s for background synchronization). Thus, the tmpfs solution is suitable for workloads that do not produce large amounts of data.
- The amount of RAM allocated to the dom0 dictates an upper bound to the transfer rate for background synchronization. In the tests with only 1 GB of RAM (see Table I, column one and two), the synchronization time does not decrease between 20 and 80 Mbit/s, which means that the network bandwidth is higher than the rate at which the incoming data can be processed at the destination host. In general, the synchronization time is inversely proportional to the transfer rate in a linear fashion.
- Increasing the transfer rate for background synchronization has an observable impact only in test 1, where both the dom0 and the VM have a small amount of RAM. This test case represents the most write intensive workload in the sense that fewer writes can be cached in main memory by the kernel (both in the dom0 and the VM). In all other cases, the overhead compared to the non-migrating reference values is mostly caused by the the synchronous disk writes. Thus, especially for write intensive workloads, choosing the transfer rate will always be a tradeoff between the total migration time and the performance impact on the VM.
- Strangely enough, in all tests with 3 GB of RAM in the VM, the duration of the kernel compile increases considerably when allocating more memory to the dom0 (e.g from 714 s to 757 s between tests 3 and 4). At the time of writing, no reasonable explanation for this phenomenon could be found, but since these observations do not affect the area under test (disk synchronization), they are simply stated as observed. This topic has to be investigated in more detail in the future.

dom0 RAM	migration time	compile time
1 GB	32.40 s	721.00 s
13 GB	32.90 s	765.10 s

TABLE II
RESULTS FOR LIVE MIGRATION WITH TMPFS AS A WRITABLE LAYER (TEST 5 AND 6)

Test no.	dom0 RAM	VM RAM	compile time
1	1 GB	256 MB	724.38 s
2	13 GB	256 MB	725.00 s
3	1 GB	3 GB	706.38 s
4	13 GB	3 GB	751.00 s
5	1 GB	3 GB	722.00 s
6	13 GB	3 GB	762.88 s

TABLE III
REFERENCE VALUES FOR THE KERNEL COMPILE BENCHMARK WITHOUT MIGRATION

In normal operation, all disk I/O of the VM goes through the DRBD driver running in standalone mode. The bonnie++[1] benchmark was used to measure the overhead of the driver. The bonnie++ test was performed on a disk image file mounted through a loopback device and on a LVM partition.

Setup	writes in KB/s	reads in KB/s
File image	44003	59081
File image standalone	41070	59717
LVM	43781	57027
LVM standalone	43239	55634

TABLE IV
OVERHEAD OF THE DRBD DRIVER IN STANDALONE MODE

Table IV shows the results. The highest performance impact can be observed on the write throughput on the disk image file (decreased by 6.7%) followed by the read throughput on the LVM volume (decreased by 2.4%). The other two values only differ by around 1% in both directions. Designed for disk performance tests, the bonnie++ benchmark represents an unusual workload stressing the I/O facilities to a maximum. Hence, average applications running in a VM will usually have more moderate disk I/O throughput and hence the observable overhead due to the DRBD driver is expected to be much lower. Especially when comparing these values to shared storage solutions[2] (which are commonly required for enabling live migration), the results indicate that the proposed solution is suitable for Cloud environments like the XGE. Additionally, the performance of shared storage access is expected to decrease with the number of clients using it simultaneously

[1] http://www.coker.com.au/bonnie++/

[2] write throughput on an NFS filesystem is decreased by approximately 48% in synchronous mode and 9% in asynchronous mode according to [15]

	Test 1			Test 2			Test 3			Test 4		
speed	**sync**	**migration**	**compile**	**sync**	**migration**	**compile**	**sync**	**migration**	**compile**	**sync**	**migration**	**compile**
5 Mbit/s	400.2 s	8.2 s	741.5 s	398.2 s	7.0 s	736.0 s	397.5 s	34.3 s	714.4 s	401.7 s	32.7 s	756.5 s
10 Mbit/s	203.0 s	8.7 s	746.5 s	201.7 s	7.2 s	736.7 s	202.0 s	34.4 s	714.2 s	202.6 s	33.0 s	756.4 s
20 Mbit/s	119.7 s	8.2 s	766.0 s	103.0 s	6.7 s	735.0 s	119.3 s	34.1 s	712.9 s	102.7 s	32.7 s	757.1 s
40 Mbit/s	104.7 s	8.0 s	795.5 s	52.7 s	6.5 s	735.0 s	102.4 s	34.0 s	717.8 s	52.7 s	32.8 s	757.2 s
80 Mbit/s	108.5 s	7.5 s	789.5 s	27.0 s	5.7 s	735.2 s	100.3 s	34.3 s	716.8 s	27.3 s	32.7 s	757.5 s

TABLE I

RESULTS FOR LIVE MIGRATION WITH STORAGE SYNCHRONIZATION (TEST 1-4).

and with a higher network utilization. The DRBD approach presented here allows the VMs to do their I/O locally and thus only introduces performance penalties during a live migration process.

VII. CONCLUSIONS

In this paper, we have presented several approaches for synchronizing virtual machine storage for live migration. Two approaches based on the idea of a multi-layered root file system deal with this issue: the first approach uses an in-memory file system as a writable layer, such that no disk image changes need to be considered during a live migration. The second approach uses a local disk image of moderate size on top of the read-only base layer. The writable part needs to be synchronized during a live migration process. For this purpose, a novel mechanism based on distributed replicated block devices (DRBD) has been presented that allows for synchronous writes on the source and the destination host. Our evaluation has shown satisfactory performance for this approach.

There are several areas for future work. For example, the combination of high availability cluster technologies with the possibility of live migration (enabling load balancing and simplifying server maintenance) is an interesting topic for further investigation. Most high availability solutions such as Heartbeat[3] or Remus [4] (which has been integrated into the Xen 4.0 release) depend on a fixed relationship between primary and backup server, such that a live migration of the primary server to another physical host cannot be achieved in a straightforward way. Furthermore, the approach should be adapted to support different hypervisors (such as e.g. KVM) in the future. Finally, the work presented in this paper could be used in an energy-aware high-performance computing environment. Live migration is a well-known technique to consolidate virtual machines on physical machines and thus it is also desirable to be able to migrate storage and swap space in an efficient manner.

VIII. ACKNOWLEDGEMENTS

This work is partly supported by the German Ministry of Education and Research (BMBF) (TIMaCS, HPC Initiative).

REFERENCES

[1] Amazon.com. Amazon WebServices. http://aws.amazon.com/, April 2010.

[2] R. Bradford, E. Kotsovinos, A. Feldmann, and H. Schöberg. Live Wide-area Migration of Virtual Machines Including Local Persistent State. In *VEE '07: Proceedings of the 3rd International Conference on Virtual Execution Environments*, pages 169–179, New York, NY, USA, 2007. ACM.

[3] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live Migration of Virtual Machines. In *NSDI'05: Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation*, pages 273–286, Berkeley, CA, USA, 2005. USENIX Association.

[4] B. Cully, G. Lefebvre, D. Meyer, M. Feeley, N. Hutchinson, and A. Warfield. Remus: High Availability via Asynchronous Virtual Machine Replication. In *NSDI'08: Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, pages 161–174, Berkeley, CA, USA, 2008. USENIX Association.

[5] M. R. Hines, U. Deshpande, and K. Gopalan. Post-copy Live Migration of Virtual Machines. *SIGOPS Operating Systems Review*, 43(3):14–26, 2009.

[6] M. Kozuch and M. Satyanarayanan. Internet Suspend/Resume. In *Proceedings of the Fourth IEEE Workshop on Mobile Computing Systems and Applications*, page 40, Washington, DC, USA, 2002. IEEE Computer Society.

[7] LINBIT HA-Solutions GmbH. DRBD - Software Development for High Availability Clusters. http://www.drbd.org/, 2010.

[8] Y. Luo, B. Zhang, X. Wang, Z. Wang, Y. Sun, and H. Chen. Live and Incremental Whole-system Migration of Virtual Machines Using Block-Bitmap. In *2008 IEEE International Conference on Cluster Computing*, pages 99–106, 2008.

[9] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov. The Eucalyptus Open-Source Cloud-Computing System. *CCGRID '09: Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 124–131, 2009.

[10] C. P. Sapuntzakis, R. Chandra, B. Pfaff, J. Chow, M. S. Lam, and M. Rosenblum. Optimizing the Migration of Virtual Computers. *In Proceedings of the 5th Symposium on Operating Systems Design and Implementation*, 36(SI):377–390, 2002.

[11] M. Satyanarayanan, J. J. Kistler, P. Kumar, M. E. Okasaki, E. H. Siegel, David, and C. Steere. Coda: A Highly available File System for a Distributed Workstation Environment. *IEEE Transactions on Computers*, 39:447–459, 1990.

[12] M. Schmidt, N. Fallenbeck, M. Smith, and B. Freisleben. Efficient Distribution of Virtual Machines for Cloud Computing. In *Euromicro Conference on Parallel, Distributed, and Network-Based Processing*, pages 567–574. IEEE Computer Society, 2010.

[13] R. Schwarzkopf, M. Schmidt, N. Fallenbeck, and B. Freisleben. Multi-layered Virtual Machines for Security Updates in Grid Environments. In *Euromicro Conference on Software Engineering and Advanced Applications*, pages 563–570. IEEE Computer Society, 2009.

[14] M. Smith, M. Schmidt, N. Fallenbeck, T. Doernemann, C. Schridde, and B. Freisleben. Secure On-demand Grid Computing. In *Journal of Future Generation Computer Systems*, pages 315–325. Elsevier, 2008.

[15] Softpanorama - Open Source Software Educational Society. NFS Performance Tuning. http://softpanorama.org/Net/Application_layer/NFS/nfs_performance_tuning.shtml, August 2009.

[16] W. Voorsluys, J. Broberg, S. Venugopal, and R. Buyya. Cost of Virtual Machine Live Migration in Clouds: A Performance Evaluation. In *CloudCom '09: Proceedings of the 1st International Conference on Cloud Computing*, pages 254–265, Berlin, Heidelberg, 2009. Springer-Verlag.

[3] http://www.linux-ha.org/wiki/Main_Page