A Survey of Migration Mechanisms of Virtual Machines

VIOLETA MEDINA, Universidad Michoacana de San Nicolás de Hidalgo
JUAN MANUEL GARCÍA, Instituto Tecnológico de Morelia

In the virtualization area, replication has been considered as a mechanism to provide high availability. A high-availability system should be active most of the time, and this is the reason that its design should consider almost zero downtime and a minimal human intervention if a recovery process is demanded. Several migration and replication mechanisms have been developed to provide high availability inside virtualized environments. In this article, a survey of migration mechanisms is reported. These approaches are classified in three main classes: process migration, memory migration, and suspend/resume migration.

Categories and Subject Descriptors: C.2: Computer-Communication Networks; C.2.2 [**Network Protocols**]: Routing Protocols; C.2: Computer-Communication Networks; C.2.4 [**Distributed Systems**]: Network Operating Systems

General Terms: Design

Additional Key Words and Phrases: High availability, migration, replication, virtual machine, virtualization

ACM Reference Format:
Violeta Medina and Juan Manuel García, 2014. A survey of migration mechanisms of virtual machines. ACM Comput. Surv. 46, 3, Article 30 (January 2014), 33 pages.
DOI: http://dx.doi.org/10.1145/2492705

1. INTRODUCTION

Virtualization is a technology that started in the 1960s. The origin of virtual machines was in mainframe computers with multiprogramming operating systems, which made use of time sharing and resource sharing on expensive hardware [Susanta and Tzi-Cker 2005]. The principal architectural characteristics were the dual-state hardware organization with a privileged and a nonprivileged mode [Goldberg 1974]. The architecture of IBM's M44/44X was the first to introduce the *virtual machines* term. The main machine was an IBM 7044 (M44) scientific computer and several simulated 7044 virtual machines, or 44Xs, using both hardware and software, virtual memory, and multiprogramming, respectively. The address space of a 44X was resident in the M44's memory hierarchy, implemented via virtual memory and multiprogramming [Dittner and Rule 2007].

Nowadays, the interest in virtualization has resurged since computers have enough processing power to use virtualization as a technique of partitioning or dividing the resources of a single server into multiple separated execution environments. Each

This work is supported by Consejo Nacional de Ciencia y Tecnología (CONACYT) and Universidad Michoacana de San Nicolás de Hidalgo.
Authors' addresses: V. Medina, División de Estudios de Posgrado, Facultad de Ingeniería Elíctrica, Universidad Michoacana de San Nicolás de Hidalgo, Ciudad Universitaria, Francisco J. Múgica S/N, Col. Felicitas del Rio 58030, Morelia, Mich., México; J. M. García, Computer Systems Department, Instituto Tecnológico de Morelia, Av. Tecnológico 1500, Morelia, México.

DOI: http://dx.doi.org/10.1145/2492705

environment is isolated from others; therefore, multiple operating systems can run on the same hardware [Barham et al. 2003; Changarti 2007; Rosenblum 2004]. These logical units are called Virtual Machines (VMs).

The basic motivation for virtualization is the same for the multitask operating systems; the computers have more processing power than necessary to fulfill a task. The computers of the second generation were programmable; those computers could complete just one task at a time. Eventually, the hardware became faster and a computer could complete a job and had unused resources. Multitasking took advantage of the unused computational power [Chisnall 2007].

Virtualization introduces a software abstraction layer between the hardware and the operating system and applications running on top of it. This abstraction layer is called a Virtual Machine Monitor (VMM) or hypervisor. Because the hardware resources are controlled by the VMM, it is possible to run in parallel multiple operating systems in the same hardware [Sahoo et al. 2010].

Rosemblum and Garfinkel [2005] present an overview of VMMs. The current VMMs offer characteristics that attracted the interest of researchers, such as isolation, server consolidation [Padala et al. 2007], cloning, server migration, and replication. These characteristics are used to provide high availability in a virtualized system.

Many organizations found that they had servers doing only one task or small clusters of related jobs. The virtualization allows that a series of virtual servers can be consolidated within a unique physical machine without losing the security in the isolated environments.

A VMM provides certain characteristics, such as clonation, at a very low cost. For example, if a patch is tested, a VM can be cloned, and the patch can be applied to the clone machine to observe the way the system is affected. This procedure is easier than a test in a machine in production. Another issue that has contributed to increase the VMM popularity is the support to the legacy operating systems. System administrators can use this capacity to consolidate many underutilized servers into only one machine, thus ensuring isolation and efficient performance [Whitaker et al. 2005] by expanding these capacities to clustering environments [Kiyanclar 2005]. Another great advantage is the migration. A virtual machine can be migrated to another host if the hardware begins to experience failures or if an actualization is scheduled. The machine can be migrated to the original machine when it begins to work again. The efficient use of energy makes the virtualization attractive. An idle server consumes power, too. Consolidating many servers in VMs in a small number of hosts can contribute to reduce the cost of energy in a substantial way [Torres et al. 2008]. Finally, virtualization provides a high degree of isolation between VMs [Gupta et al. 2006]. A virtual machine failure will not affect other running virtual machines in the same hardware. Performance isolation is another important goal; the resource consumption of one virtual machine should not impact the performance of other VMs sharing the same hardware.

Several VMMs include migration as a mechanism to provide fault tolerance in a highly available system. Migration is a tool for load balancing, dealing with hardware failures, scaling systems, or reallocating resources. A migration mechanism has to provide almost zero downtime.

This migration should be transparent to the guest operating system, applications running on the operating system, and remote clients of the virtual machine. It should appear, to all involved parties, that the virtual machine did not change its location. The only perceived change should be a brief slowdown during the migration and a possible improvement in performance after the migration, since the VM was moved to a machine with more available resources.

This article is focused on migration mechanisms, but replication approaches are presented too. We consider that a migration mechanism moves a VM from one host

to another and just one copy is created at a time. A replication mechanism maintains multiple copies of an original image VM in sync. It is typically used to create a failover scheme for a highly available system. Record and replay techniques may be used to support migration, but they can also be used for replication or diagnosis/debugging. A record/replay mechanism captures and records nondeterministic events of a source VM in a log file. The stored information may be used to replay the whole state of the source VM in a different VM. In this article, different forms of migration have been reviewed.

The remainder of Section 1 shows the background of VM migration. The rest of the article is structured as follows. In Section 2, we present live migration mechanisms. In Section 3, we present suspend/resume migration mechanisms. In Section 4, we describe some live migration approaches, which migrate VMs over a WAN. In Section 5, we present some migration mechanisms to get load balancing and efficient resource consumption. In Section 6, we show the automatic management of live migration. In Section 7, we describe some migration mechanisms based on a record/replay technique and the live migration of the lightweight virtualization solution. In Section 8, we present some current trends, and in Section 9, we conclude.

1.1. Definition of Terms

In the section to follow, several migration and replication mechanisms are detailed. We consider that a migration mechanism moves, at a time, a single copy of a VM from one host to another. A replication mechanism maintains multiple copies of an original image VM in sync. Record and replay techniques may be used to support migration. A record/replay mechanism captures and records nondeterministic events of a source VM in a log file. During a replay phase, these log records guide the virtual machine as it re-executes from a checkpoint.

1.2. Background Work

In the mid-1980s, there were two important technological advances. The first one was the development of powerful microprocessors. CPUs quickly were manufactured in 64 bits. The second development was the invention of high-speed computer networks. The use of Local Area Networks (LANs) and Wide Area Networks (WANs) allows building computing systems formed by a large number of CPUs connected by a high-speed network. These systems were called distributed systems. It was necessary to develop operating systems to support distributed systems. A distributed system can be defined as a collection of independent computers that appears to users as a single computer [Tanenbaum and Steen 2001]. A system can reach this goal if it is transparent. Distributed systems consider several concepts of transparency: location transparency—users cannot know where resources are located; migration transparency—resources can move anywhere without changing their names; replication transparency—users cannot know how many copies of data exist; concurrency transparency—users can share resources automatically; and parallelism transparency—some activities can be executed in parallel without users knowing. Under this philosophy, many distributed operating systems were developed with process migration as an inherent characteristic.

Several operating systems were created to work in a distributed environment, such as Amoeba [Mullender et al. 1990], Mach [Tanenbaum 1995], and Chorus [Rozier et al. 1991]. Within the first works in migration, the migration process can be considered as a strategy to copy the complete state from one VM to another physical machine.

1.2.1. Zap. Zap [Osman et al. 2002] introduces a virtualization layer on top of the operating system called PrOcess Domain (pod). A pod includes a group of processes within a private namespace that presents the process group with the same virtualized view of the system. A pod allows relating virtual identifiers with operating system

resources like process identifiers and network addresses. This abstraction separates a pod from dependencies on the operating system and from other processes in the system.

Zap was implemented as a kernel module without kernel modifications, which intercepts system calls as needed for virtualization and saves and restores the kernel state as needed for migration. The Zap prototype provides a migration process with low overhead.

To support transparent process migration, Zap considers three requirements: resource consistency, resource conflicts, and resource dependencies. The first requirement is the need to preserve resource-naming consistency. An operating system contains numerous identifiers for its resources, including Process IDs (PIDs), file names, and socket ports. The operating system assumes that the identifier does not change during a cycle of a process life.

The second requirement is oriented to eliminate naming conflicts when processes are migrated. The third requirement is to eliminate process dependencies when a process is migrated.

Pods are self-contained units that can be suspended to secondary storage, migrated to another machine, and transparently resumed. A pod can contain any number of processes. The main difference between a pod and a traditional operating system environment is that each pod has its own private, virtual namespace.

Names within a pod are assigned in the same way that traditional operating systems assign names, but such names are localized to the pod. Because the namespace is private to a given pod, there are no resource-naming conflicts for processes in different pods.

Zap virtualization is based on a checkpoint-restart mechanism to suspend, migrate, and resume pods and their associated processes. The checkpoint-restart approach avoids leaving residual components, migrating the complete state in a checkpointed image; a pod can be migrated over a network or in local storage and the checkpointed state can be stored.

To migrate a pod, Zap first suspends the pod by stopping all processes in the pod, saving the virtualization mappings, and saving all process states, including memory, CPU registers, open file handles, and so forth. On the destination host, Zap resumes the pod by first restoring the pod virtualized environment and then restoring processes in a stopped state; for this reason, it is necessary to create the virtualization mappings of the stopped processes. Finally, Zap enables the processes to continue executing in the restored pod environment.

In the process of migrating a pod, the memory migration is considered. Zap checkpoints the memory areas allocated by processes within the pod. Zap ensures that the same view of the file system is available to a pod on whatever machine on which the pod is executed. Because of this, Zap does not need to save the code segments that belong to an executable, for example, text pages of the process or linked libraries in use by the process. Migration of text pages is carried out by saving references to the executable files and virtual memory addresses to which they are mapped.

1.2.2. AutoPod. AutoPod [Nieh 2005] is a system that determines when a patch needs to be applied in an operating system based on a system status, preserving application service availability. The migration of application services to another host is automatic.

The AutoPod model is based on a virtual machine abstraction called a pod. For migration, the same operating system version for all systems is assumed.

AutoPod uses a virtualization layer to translate the AutoPod namespace to the host operating system namespace.

AutoPod uses a checkpoint-restart mechanism and an intermediate format to represent the state that needs to be saved at the checkpoint. When a checkpoint is realized,

the intermediate format representation is saved and digitally signed; during the restart process, the integrity of the image is verified. According to performed tests, to start a desktop environment from scratch can take around 20 seconds; the checkpoint and restarting of AutoPod can take 851ms and 942ms, respectively.

It is common that operating system vendors provide their users with the ability to automatically check for system updates and download and install them when they become available. AutoPod provides a service that monitors the security repositories; it downloads the security updates and through the checkpoint/restart mechanism enables the actualizations.

Commodity systems can provide information about the state of the system and detect some irregular conditions such as DMA timeout or the need to reset the IDE bus. When AutoPod discovers possible failures, this service checkpoints the pods running on the system, migrates the pod, and restarts it on a new host.

2. LIVE MIGRATION

2.1. Important Concepts

In the virtualization area, several replication and migration approaches to achieve fault tolerance and high availability have been proposed. In the ideal copy of a virtual machine, the complete state from the original VM should be transferred to the copy including memory, disk, and network connections. The memory state migration has been widely studied and basically there are two methods, precopy and postcopy. Local disk and network interface migration are not trivial. To maintain network connectivity after migration, it is necessary to preserve open connections. Network clients should be attended with minimal disruption. If the migration is within the same LAN, a VM should retain its original IP address after migration, by generating an unsolicited ARP reply advertising the new location for the migrated VM's IP [Clark et al. 2005]. But in a WAN migration, the use of network technologies such Virtual Private Networks (VPNs), tunneling [Travostino 2006; Wood et al. 2011], and DNS servers [Bradford et al. 2007] is useful. The disconnected mode [Satyanarayanan et al. 2005; Chandra et al. 2005; Cáceres et al. 2005] is employed too. Sometimes disk state migration is not considered, because it is assumed that a SAN or NAS strategy is used among VMs, but a WAN environment should consider the migration of local disk. Migrating disk state typically represents the largest component of the overall migration time, as the disk state may be in the tens or hundreds of gigabytes. Many strategies to migrate storage are used such as record and replay, transmission by deltas, or the implementation of a Distributed Replicated Block Device (DRBD) [LINBIT 2011] storage solution.

2.1.1. Memory Migration. There are two types of live migration mechanisms, precopy and postcopy. Postcopy migration transfers a VM's memory contents after its processor state has been sent to the target host. It differs from the precopy approach, which first copies the memory state to the destination, through a repetitive process, after which its processor state is transferred to the target.

Postcopy migration [Hirofuchi et al. 2011] basically follows the next steps:

(1) Stop the VM at the source host. Processor state is sent to destination VM, and the content of virtual CPU registers and the states of devices are copied to the destination.
(2) Resume the VM at the destination without any memory content.
(3) If the VM tries to access pages that have not yet been transferred, the VM is temporarily stopped and the fault pages are demand paged over the network from source. The VM is resumed.

Precopy migration [Milojičić et al. 2000; Clark et al. 2005] follows the next phases:

(1) The source VM continues running while all memory pages are copied to the destination VM in a first iteration. Subsequent iterations copy only those pages that were modified during the previous transfer round.
(2) The source VM is stopped; the CPU state and remaining dirty pages are copied. The destination VM is started.
(3) The new VM starts operation; if it tries to access a page that has not yet transferred, this page is brought from the source VM.

Both memory migration mechanisms have some pros and cons. At the first step of precopy memory migration, all memory pages are transferred to the VM destination; therefore, the migration time increases in proportion to the memory size of the VM. Besides, dirtied pages must be iteratively copied to the destination. If the VM is executing a write-intensive workload accessing large amounts of memory, numerous dirty pages are generated and transferred continuously. In the worst case, live migration is never completed [Hirofuchi et al. 2011]. Precopy minimizes VM downtime and application degradation when the VM is executing a read-intensive workload.

In postcopy migration, after the destination VM is started, memory pages are transferred across the network on first use. This approach consumes a shorter downtime but produces a longer total migration time, and the performance during the migration is likely to be considerably degraded when a large number of memory pages have been demand-paged across the network. However, in precopy migration, the iterative checking and sending of modified pages between two hosts could consume the whole bandwidth available between them and cause degradation of active services. In both schemes, mechanisms to alleviate their drawbacks have been implemented. For example, in postcopy [Hines et al. 2009], ballooning [Waldspurger 2002] is included to improve performance. In precopy, a bandwidth limit is defined during the subsequent round of copies to avoid the service degradation [Clark et al. 2005].

In postcopy migration, faulted pages are transferred on demand over the network from the source, and then each memory page is sent at most once, avoiding the duplicate transmission overhead of precopy.

Failure of the destination node has different implications in the two memory migration types. For precopy, there is no problem if a failure of the destination node occurs, because the source node still holds an entire up-to-date copy of the VM's memory and processor state. If it is necessary, the VM can be revived from this image. However, when postcopy is used, the destination node has a more up-to-date copy of the virtual machine. Hence, a failure of the destination VM during postcopy migration becomes a critical failure of the VM.

Live migration refers to techniques in which a VM is moved from one host to another with almost zero downtime. Usually, the common environment for live migration is server VMs within a cluster/data center where tight control over the networking enables transparent migration of connections. The use of storage technologies such as SAN or NAS eliminates the disk migration.

2.1.2. Self-Migration. This work establishes the basis for the Xen migration mechanism [Clark et al. 2005]. In Hansen and Jul [2004], two prototype systems are implemented. The first implementation, NomadBIOS, is a prototype host environment for running several adapted Linux instances concurrently. It migrates such instances between hosts without disrupting service, which is called live migration. NomadBIOS runs on top of the L4 microkernel. NomadBIOS reduces downtime using precopy migration, keeping the guest OS running on the original host while migrating, tracking changes to its address space, and sending updates to the target VM over a number of iterations.

The updates' size typically decreases to 100 kilobytes or less. To migrate network connections, a gratuitous ARP packet is then broadcast to the local Ethernet, to inform local peers about the move to a new interface. Files are accessed via iSCSI or NFS; disk migration is not considered. A second approach is called self-migration; here the guest OS performs the entire migration without hypervisor involvement; the guest OS is migrated by itself. This migration is based on Xen [Clark et al. 2005].

2.1.3. VMWare. *VMware* developed a migration mechanism called *VMotion* [Nelson et al. 2005] as part of the VMware VirtualCenter product that manages the VMware ESX Server. In Waldspurger [2002], the VMware memory mechanism is shown. The ESX Server uses a ballooning technique for memory management. A small balloon module is loaded into the guest OS as a pseudo-device driver or kernel service. This module collaborates with the server to reclaim pages that are considered least valuable by the guest operating system. When the server wants to reclaim memory, it instructs the driver to "inflate" the balloon by allocating pinned physical pages within the VM. Similarly, the server may "deflate" the balloon by deallocating previously allocated pages. When memory is tight, the guest operating system decides which particular pages to reclaim and, if necessary, swaps them to its own virtual disk. The memory ballooning mechanism allows memory that was allocated for a virtual machine to be given to another virtual machine without having to shut the machine down.

Each guest operating system that executes within a virtual machine addresses a zero-based physical address space, as within real hardware. The ESX Server gives each VM this illusion, virtualizing physical memory by adding an extra level of address translation. A machine address refers to actual hardware memory, while a physical address is a software abstraction used to provide the illusion of hardware memory to a virtual machine. Each virtual machine has a fixed set of physical address ranges that map to physical memory. All direct accesses to the VM's physical memory are intercepted by the VMM. The VMM then translates these physical addresses into the actual machine addresses. A *precopy* memory scheme is used. First, the physical memory from the source VM is copied and marked as read-only, so any modification can be detected by the VMM. When this process is finished, modified pages by the VM in execution may exist. These pages are copied to the new VM. This step is repeated until the remaining number of modified pages is small (16 megabytes) or there is a reduction in changed pages of less than 1 megabyte.

VMware provides a virtual Ethernet network card, *VNIC*, as part of its virtual platform, which has a unique MAC address within the local network. A VNIC can be associated to one or more physical network cards. Because the VNIC has a MAC address independent of the physical MAC address, the VMs can be moved from one host to another without stopping services and keeping the active network connections. This is possible only if the new VM is opened in the same subnet as the original machine. In storage, VMware assumes that the computers are connected to a server *SAN* or *NAS*.

2.1.4. Xen. Xen [Barham et al. 2003] proposed a live migration mechanism [Clark et al. 2005]. This approach is focused on VM memory migration. Live migration of virtual machines considers the minimization of downtime, which is the period during which the VM's services are unavailable, and the total migration time, which is the period that begins when the original VM starts the migration and ends when the VM is shut down. The memory migration of a virtual machine can be described in three phases. In the first phase, the source VM continues running while using the *precopy* [Milojičić et al. 2000] approach; all memory pages are copied to the destination VM in a first iteration, and subsequent iterations copy only those pages that were dirtied during the previous transfer round. In the second phase, the stop-and-copy process is executed, the source VM is stopped, and remaining dirty pages are copied and the destination VM

is started. The traffic is redirected to the new VM. In the third phase, the destination VM resumes normal operation, and if an inconsistent page still remains, it is brought from the source VM. It is considered to migrate the open network connections through an unsolicited ARP reply from the migrated host, notifying that the IP has a new location. However, some routers are configured to not accept broadcast ARP replies (in order to prevent security attacks), so this idea may not work in all scenarios. Storage migration is not necessary because a shared storage form, like *NAS,* is assumed.

The Writable Working Set (WWS) is formed by pages modified so often that they are not good candidates for the precopy phase. The Xen's shadow page tables are used to track dirtying statistics on all pages used by a particular executing operating system. Through these statistics, the WWS can be determined.

Xen considers two different methods for initiating and managing state transfer. The first one, *managed migration,* is mainly executed out of the VM, which is being migrated by a migration daemon running in the management VM. The second one, *self-migration*, is almost totally run within the VM being migrated and a short part on the destination machine.

Managed migration is performed by migration daemons running in the management VMs of the source and destination hosts. These are responsible for creating a new VM on the destination machine and coordinating transfer of a live system state over the network. The control software executes rounds to transfer the memory image. In the first round, all pages are transferred to the destination machine; in subsequent rounds, this copying is restricted to pages that were dirtied during the previous round. To log pages that are dirtied, Xen inserts shadow page tables underneath the running OS. All Page-Table Entries (PTEs) are initially read-only mappings in the shadow tables. If the guest tries to modify a page of memory, the resulting page fault is trapped by Xen. If write access is allowed by the relevant guest PTE, then this permission is extended to the shadow PTE, setting the appropriate bit in the VM's dirty bitmap. When it is determined that the precopy phase is no longer beneficial, using heuristics methods, a control message is sent to the OS requesting that it suspend itself in a state suitable for migration. The dirty bitmap is scanned one last time for remaining inconsistent memory pages, and these are transferred to the destination together with the VM's checkpointed CPU-register state. Once this final information is received at the destination, the VM state on the source machine can safely be discarded. Execution is then resumed by starting the new VM at the point that the old VM checkpointed itself.

Self-migration handles a precopying scheme similar to that for managed migration. The major implementation difficulty of this scheme is to transfer a consistent OS checkpoint. This difficulty was solved with a final two-stage stop-and-copy phase. The first phase disables all OS activity except for migration and then performs a final scan of the dirty bitmap, clearing the appropriate bit as each page is transferred. Any pages that are dirtied during the final scan, and that are still marked as dirty in the bitmap, are copied to a shadow buffer. The second phase transfers the contents of the shadow buffer. Page updates are ignored during this transfer.

The live migration evaluation was tested with different workloads by Clark et al. [2005]. As a representative test, the Apache workload can be mentioned, which is presented by SPECweb99, a complex application-level benchmark for evaluating web servers and the systems that host them. A virtual machine with 800MB of memory was used; the VM was migrated with a SPECweb99 in execution. The VM suspension is done when 18.2MB of memory remains to be sent. Until this point, the transmission takes 201 ms, after which an additional 9 ms is required for the domain to resume normal execution. The total downtime of 210 ms experienced by the SPECweb clients is sufficiently brief to maintain the 350 clients.

In the article [Clark et al. 2005] cited before, a multiplayer online server game server was tested. The performance of live migration with a virtual machine with 64MB of memory running a Quake 3 server was measured. Six players joined the game and started to play within a shared arena. Xen performed the live migration with a total downtime of 60 ms.

2.1.5. KVM. The Kernel-based Virtual Machine [Machine 2012], or KVM, is a virtual machine monitor that allows full virtualization for Linux on x86 hardware. Due to the increasing impact of virtualization, hardware vendors such as Intel and AMD have added extensions to the x86 architecture that make virtualization much easier (Intel VT or AMD-V). KVM consists of a loadable kernel module, kvm.ko, and the architecture-specific functionality is provided by two architecture-specific modules, kvm-intel.ko and kvm-amd.ko. Using KVM, it is possible to run multiple virtual machines of unmodified Linux or Windows.

In Kivity et al. [2007], live migration is implemented. Live migration uses the precopy strategy, which means that if a guest page is modified after it has been copied, it has to be copied again. KVM implements a dirty page log, which is used as a bitmap of modified pages since the last call. KVM maps guest pages as read-only and only maps them for write after the first write access. Live migration is completed in three phases [Clark et al. 2005]. This is an iterative process; each iteration copies memory to the destination host. In the first phase, all memory pages are marked dirty and the modification-tracking mechanism is initialized. In the second phase, pages marked as dirtied are copied. The iterative process of copying pages continues while the maximum transfer rate to the destination machine is not exceeded. Modified pages, but not copied, are used to estimate the downtime if the migration starts the third stage. If the estimated downtime is high compared with the target value, the algorithm iterates until it predicts a value lower than the target value. When the target downtime is reached, the migration enters the third and final phase, where the source virtual machine (and applications) is stopped. Dirty pages are transmitted to the destination machine, the registers are loaded, and execution is resumed on the new host [Ibrahim et al. 2011].

2.1.6. Kemari. A synchronization approach for VMs based in Xen is proposed in a project called *Kemari* [Tamura 2008]. In this project, when Kemari detects an event from the guest domain, this domain is paused, and the dirty pages created since the last synchronization are located and sent to the image VM. When the synchronization is completed, Kemari unpauses the guest domain and the event is executed in domain 0.

2.1.7. Live and Incremental Migration of Virtual Machines. In Luo et al. [2008], a live and incremental migration of VMs has been proposed; it includes memory, CPU, and disk state migration. It uses a Three-Phase Migration (TPM) algorithm (precopy, freeze-and-copy, postcopy), which is an expansion of live migration of Xen where an Incremental Migration (IM) is used for storage. During the precopy phase, the storage data are iteratively precopied. In the first iteration, all the storage data should be copied to the destination VM. After, modified data, just during the last iteration, is sent to the backup VM.

2.1.8. Migration without Virtualization. Currently, most of the migration approaches are based on virtualization and have as a requirement the hardware compatibility between the physical source machine and physical destination machine. In Kozuch et al. [2009], a mechanism for operating systems to allow direct migration from one physical machine to another physical machine is proposed even if the hardware on the target machine is different from the source machine.

2.1.9. Postcopy Live Migration of Virtual Machines. In Hines et al. [2009], a design, implementation, and evaluation of postcopy-based live migration for virtual machines across a Gigabit LAN are presented. Postcopy migration transfers the VM's memory contents after its processor state has been sent to the destination host. The precopy approach first copies the memory state over multiple iterations and after transfers the processor state to the target host.

In this work, the postcopy strategy for live VM migration is proposed and evaluated. Efficiency of postcopy depends on the process to minimize the number of page faults (or network faults); to speed up the migration, memory pages are pushed from the source before they are needed by the VM at the target. To reduce network faults, postcopy live migration includes the active push component with adaptive prepaging.

The way in which pages are fetched creates different variants of postcopy, providing improvements. Postcopy migration combines four techniques to fetch memory pages from the source: demand paging, active push, prepaging, and Dynamic Self-Ballooning (DSB). When demand paging is used, the VM resumes at the target; when a memory fault is produced, this is serviced by a request to the source node over the network. Active push avoids transferring pages more than once. A page is transferred either by demand paging or by an active push. Active pushing reduces the duration of residual dependencies on the source host, pushing the VM's pages from the source to the target VM that continues in execution. Prepaging uses network faults as start points to predict the VM's page access locality at the target and search pages in the neighborhood of a network fault before they are accessed by the VM. The VM's pages at source are stored in an in-memory pseudo-paging device, which resides completely in memory. Prepaging applies an bubbling algorithm, which starts the active pushing from a pivot page inside the pseudo-paging device and transmits symmetrically located pages around that pivot. Pages that have already been transmitted are skipped. The bubbling method can be executed with a single or multiple pivots. Transferring free pages would be a waste of network and CPU resources and would increase the total migration time regardless of which migration algorithm is used. DSB reduces the number of free pages transferred during migration, improving the performance of precopy and postcopy migration. The VM performs ballooning [Waldspurger 2002] continuously over its execution lifetime.

The postcopy migration proposed is compared against Xen's precopy migration. Both the VM in each experiment and domain 0 are configured to use two virtual CPUs; the default VM size is 512MB. Different performance metrics are analyzed with four applications: (a) SPECWeb 2005: the system is configured within a VM and six external clients request connections; (b) BitTorrent Client: a multipeer distributed application, which is slightly CPU intensive; (c) Linux Kernel compile: considered because of consistency; (d) Netperf: the Netperf sender is configured inside the VM. They evaluate four metrics: downtime, total migration time, pages transferred, and page faults.

Postcopy is effective only when a large majority of the pages reach the target before they are faulted upon the target VM, in which case they become minor page faults rather than major network page faults. The quantity of major faults compared to minor page faults is a parameter of effectiveness of postcopy. So, for all of the applications except the SPECWeb, postcopy reduces the total pages transferred by more than half. The most significant result is where postcopy's prepaging algorithm becomes network page faults in minor faults for the largest applications, SPECWeb and Bittorrent, in 79% and 83%, respectively. Postcopy reduces the total migration time for all applications compared to precopy, in some cases by more than 50%. But the downtime is much higher for postcopy than for precopy.

2.1.10. Optimizing Live Migration of Virtual Machines. In Atif and Strazdins [2009], an optimization to the live migration mechanism of Xen is proposed. The experimental tests

were executed in a high-performance computing environment. The improvement is based on the reduction of the iterations number in the precopy phase of the live migration of Xen to the minimum of two. This optimization is able to reduce the total number of memory pages transferred during the migration up to 500% and their results can show an average of 50% of improvement, in comparison with the original Xen live migration mechanism.

3. SUSPEND/RESUME MIGRATION

3.1. Important Concepts

Chen and Noble [2001] suggested that virtual machine technology can be used to provide user mobility in a secure way. Suspend/resume technology is based on this idea. Suspend/resume migration refers to the motion of a VM from one host to another where the VM is locally inactive during the translation. Usually, this is a migration across a WAN. The network connections are typically dropped and need to be re-established. For a WAN migration, it is crucial to not just transfer the VM images but also transfer the local persistent state, its ongoing network connections, and the support for the disconnected operation. A technique to optimize the disk transfer is the use of deltas. On a source host, the process intercepts write operations and generates deltas, which are communication units that contain the written data, the location on the disk, and the size of the written data. The process examines the stored data and locates blocks or bytes that have changed since the last write. Changed data, rather than the whole information, can then be sent to the target hosts across the LAN or WAN.

Disconnected operation [Kistler and Satyanarayanan 1992] is a mode of operation that enables a client to continue accessing critical data during temporary failures of a shared data repository. While disconnected, file system service requests are relying solely on the contents of its cache. When disconnection ends, modifications are propagated to the corresponding server.

3.2. Internet Suspend/Resume

Internet Suspend/Resume (ISR) [Kozuch and Satyanarayanan 2002; Satyanarayanan et al. 2007] presents a project where the complete state of a VM can be migrated. ISR is based on the idea that a VMM encapsulates the volatile execution state of a VM and a VMM transfers the state of its VMs to files in the local file system within the host machine. When a VM is suspended, the volatile state is transferred to files until a suspension point. These files, including the operating system, are copied to the remote machine, where the VM can be reassumed. The downtime depends directly on the file size to be transferred.

ISR uses the distributed storage technology. Each VM encapsulates a guest operating system and guest applications in an execution state and a user customization state, and both are called a parcel. The distributed storage layer transports a parcel across space (from suspend site to resume site) and time (from suspend instant to resume instant). Users can own multiple parcels, just as they can own multiple machines with different operating systems or application suites.

This ISR structure remained invariant across the many different ISR versions. The ISR layer was implemented in two parts. One part was a loadable kernel module called Fauxide that served as the device driver for a pseudo-device. VMware was configured to use this pseudo-device for the VM state. Fauxide redirected VMware requests to this pseudo-device to a user-level process called Vulpes, which was the ISR layer's second component. Vulpes implemented the VM state-transfer policy, the VM state's mapping to a directory tree of 256Kb files in coda, and hoarding control for these files.

Some components implementing each layer have changed over time. For example, the virtual machine monitor was VMware in early versions of ISR but in the latest versions VMware, KVM, or Xen can be used. These VMMs support a mode in which a local disk partition holds the VM state. The ISR client software encrypts data from a parcel before handing it to the distributed storage layer. Neither servers nor persistent client caches used by the distributed storage mechanism contain any unencrypted user state. Compromised storage servers can, at worst, result in a denial of service. Compromise of a client after a user suspends can, at worst, prevent the updated VM state from being propagated to servers, also resulting in denial of service. Even in these situations, ISR preserves the privacy and integrity of user parcels.

ISR has implemented some versions: ISR-1, ISR-2, and ISR-3. In the latest version, coda has integrated a mechanism to use distributed storage.

In the first implementation of ISR, some results were reported. The prototype ISR implementation is able to suspend a VM on Client 1 and resume it on Client 2, and vice versa.

A measure of performance was reported with ISR-1. In tests, it was considered that before each resume operation on a client a reboot should be executed to ensure a cold NFS cache. Two types of suspend events were considered: warm suspend, which occurs in a VM shortly after a resume event, and cold suspend, which occurs long after a resume event. A cold resume takes 125 seconds, a warm suspend 114 seconds, and a cold suspend 146 seconds. These results are tolerable for some users but generally much longer than suspend resume in a laptop.

ISR-2 incorporates disconnected operation, where users can use the cached state even when disconnected. The client stores updates and reintegrates them when network connectivity is restored.

3.3. Capsule

A capsule [Sapuntzakis et al. 2002] is defined as a hardware state that includes the entire operating system, applications, and running processes. This state is moved over a network and includes the state in its disks, memory, CPU registers, and I/O devices. This project is based on the Collective [Chandra et al. 2005] and handles x86 architecture. Capsules can be suspended from execution, serialized, and resumed after. Capsules can be moved among machines to balance loads or for fail-over. An inactive capsule can contain gigabytes of disk storage, whereas an active capsule can include hundreds of megabytes of memory data, as well as internal machine registers and I/O device states. To copy a whole capsule to another physical location would take a long time. But this project implements some optimization to reduce storage requirements, transfer time, and start-up time over a network of a capsule. Four optimizations are presented:

(1) To reduce the memory state before serialization, the ballooning technique is employed. The balloon program asks the OS for a large number of physical pages. The program then zeroes the pages, making them easily compressible. Ballooning reduces the size of the compressed memory state and thus reduces the start-up time of capsules. This technique works well if the memory has many freed pages whose contents are not compressible. This memory is not transferred, and these pages are the first to be cleared by the ballooning process.
(2) Each time a capsule starts, all the updates made to disk are saved on a separate disk, using copy-on-write. Capsules are created in hierarchical order, so each child capsule could be viewed as inheriting from the parent capsule. The differences in disk state between parent and child are captured in a separate Copy-On-Write (COW) virtual disk. This strategy reduces the cost of saving a capsule disk because only the differences are captured.

(3) Instead of sending the entire disk, disk pages are fetched on demand as the capsule runs.
(4) Transfer time is decreased by sending a hash of data blocks instead of complete data. Collision-resistant hashes are used to avoid sending pages of memory or disk data that already exist at the destination. It is expected to find identical blocks of data between disk images and memories, even across different users' capsules. All network traffic is compressed with gzip.

3.4. The Collective

The Collective [Chandra et al. 2005] is a system that provides managed desktops to Personal Computer (PC) users. System administrators create such desktop environments called virtual appliances, which include the operating system and all installed applications. Target PC runs client software, called the Virtual Appliance Transceiver (VAT), that caches and runs the latest copies of appliances locally and continuously backs up changes to user data to a network repository. The Collective is available to machines with x86 architecture and uses the virtualization technology of VMWare GSX Server [VMware 2012]. PCs can be attached to a LAN or WAN with broadband or even disconnected from the network like a laptop. Users can access their desktops from any Collective client; they can also carry a bootable drive that converts a PC into a client. A VAT is built using a Knoppix Live CD, which automatically detects available hardware at runtime and load Linux drivers. A display client browser can be used to access a desktop running on a remote server.

The Collective presents a cache-based system management, which separates a computer state into two parts: the system state, which consists of an operating system and all installed applications, and the user state, which consists of a user's profile, preferences, and data files. In this model, appliances and user state are separately stored in network-accessible appliance repositories and data repositories. PCs are called VATs. A user can use any of these clients, log in, and get access to any appliance with authorization. The VAT executes functions like authenticate users, fetches and runs the latest copies of appliances locally, stores user state to the data repository, handles a cache to reduce the amount of the data that needs to be fetched over the network, and improves the performance. A VAT can run a prefetcher process to minimize cache misses. The VAT runs a prefetcher process, and even a client can completely prefetch an appliance into a cache to work in disconnected mode.

3.5. SoulPads

In Cáceres et al. [2005], a system called SoulPad is presented. This system allows a user to resume a personal computing session that was suspended in another machine. The SoulPad divides the user's machine into a body (display, CPU, RAM, I/O) and a soul (session state, software, data, preferences). The soul is carried in a small and light portable device, the SoulPad. The soul can resume on any x86-based personal computer with no preloaded software. The computers that resume SoulPad are denoted as EnviroPCs. Connections between the SoulPad and the EnviroPC are made through USB 2.0. The user can retrieve a suspend state in disconnected mode. EnviroPCs do not need any preloaded software. Users migrate from one machine to another by moving the SoulPad disk. A SoulPad disk contains three elements: first, the host OS Knoppix that boots on EnviroPCs and obtains hardware availability via auto-configuration; second, the VMM, VMware Workstation, which supports the suspend/resume operations on virtual machines and guest OS diversity; and third, a VM that executes the user's applications on a guest OS (Windows or Linux). The disk partition that holds the VM images is encrypted using the AES128 block cipher.

3.6. Cloudlets

The architecture where a mobile user exploits VM technology to instantiate customized service software on a nearby cloudlet and then uses that service over a wireless LAN is discussed in Satyanarayanan et al. [2009]. The mobile device typically functions as a thin client with respect to the service. A cloudlet is defined as a trusted, resource-rich computer or cluster of computers that is well connected to the Internet and available for use by nearby mobile devices. It is recognized that mobile hardware is resource poor with respect to static client and server hardware. A solution to improve this disadvantage is to use cloud computing.

3.7. CloneCloud

CloneCloud is presented in Chun et al. [2011]. This application partitioner and execution runtime makes possible that unmodified mobile applications run in an application-level virtual machine, off-loading part of their execution onto device clones [Chun and Maniatis 2009] operating in a computational cloud. The system transforms a single-machine execution of a mobile device into a distributed execution inside a cloud. An application-level VM is an abstract computing machine that allows hardware and operating system independence. Its instructions are bytecodes; an executable is a set of bytecodes. The VM runtime executes bytecodes of methods with threads. A partition mechanism creates a partition, which is a choice of execution points where the application migrates part of its execution and state between the device and a clone. During the execution, if a migration point is found, the executing thread is suspended and its state (including virtual state, program counter, registers, and stack) is packaged and sent to a synchronized clone. At the clone, the thread state is copied into a new thread with the same stack and reachable heap objects, and then resumed. If the migrated thread reaches a reintegration point, it is suspended, packaged, and then sent back to the mobile device. Finally, the returned packaged thread is merged into the state of the original process. A mathematical optimizer chooses migration points that optimize total execution time or mobile device energy consumption according to the application and the cost model. Test results have shown that some applications achieve as much as a 20 times execution speed-up and up to 20 times decrease of energy spent on the mobile device.

4. WAN LIVE MIGRATION

4.1. Important Concepts

In the VM migration across a WAN, the transference of the complete VM state including disk, open network connections and memory pages is essential. The next section presents some works that move the complete VM state from one host to another in a live migration over a WAN.

4.2. Seamless Live Migration of Virtual Machines over the MAN/WAN

Live migration over WANs is an interesting topic that has attracted attention in recent years. To achieve live migration of the virtual machine across WANs, an approach using IP tunnels to guarantee network connections was presented in the “VM turntable” demonstrator [Travostino 2006]. The IP tunnel between the source and target host is set up to forward packets to and from the client applications. Live migration is executed over dedicated lightpaths, which are circuits of 1Gbps capacity; they were created between two sites in Amsterdam and San Diego. A VM Traffic Controller (VMTC) coordinates the migration of VMs providing network resources and ensuring seamless layer 3–7 connectivity to the applications inside of the VM. The VMTC sent migration commands to the Xen environment. The system was tested over a network

with two sites, Amsterdam and San Diego. The source and destination VM ran a demo application for face detection, which continuously fetched images from storage. It was set up with 200MB memory. The live migration caused an application downtime of 0.8–1.6 seconds, which, compared to the downtime with intra-LAN configuration, was five to 10 times higher.

4.3. LiveWide-Area Migration of Virtual Machines

Virtual machine migration on local area networks has been concentrated on developing efficient methods to transfer the memory state of a VM. However, for wide-area network migration, it is important to transfer additional resources of a VM, like the local persistent state (its file system) and the ongoing network connections.

Bradford et al. [2007] combine a block-level solution with precopying and write throttling to transfer an entire running web server including its local persistent state with minimal disruption. The combination of dynDNS with tunneling transfers transparently existing connections. The new ones are redirected to the new network location. The original VM remains in operation on the source host.

During the storage migration, a user-level block device is used to record and forward any write accesses to the destination. The management network connection is made by a redirection scheme and the notification of the new addresses via dynDNS. The system is implemented as part of the XenoServer platform and uses the memory migration of Xen.

The migration mechanism is coordinated by a migration client, running at the source, in continuous communication with a migration daemon, running at the destination.

The system follows some phases. The initialization phase sets up the client and server processes, which handle the bulk transfer of the disk image to destination. It then starts the bulk transfer stage, which precopies the disk image of the VM to the destination while the VM continues in execution. Then, the system invokes the Xen's live migration, which iteratively copies dirtied pages to the VM destination and logs them. Finally, the source VM is paused, the remaining dirtied pages are copied to destination, and the VM is resumed.

As the VM continues to run at the source during both the bulk transfer and Xen live migration stages, it is necessary to ensure that any changes to its disk images are forwarded to the destination and applied to the version there. During bulk transfer and live migration phases, write operations on the source VM are intercepted. These operations are packaged as deltas, which are communication units that consist of the written data, the location of the write on the disk, and the size of the written data. Deltas are sent and enqueued on the destination VM for later application to the disk image. If the rate at which the VM is performing write accesses is too high, a write throttling is applied to minimize bandwidth congestion. This is based on a threshold and a delay; when a VM reaches the number of writes defined as threshold, a new attempt to write is delayed by the delay parameter.

After the bulk transfer phase finishes, the Xen live migration stage starts and in parallel the enqueued deltas are orderly applied at the destination disk image. Any new writes that occur during this stage are enqueued on the destination host and once again applied in order. Before the Xen live migration is completed, the original VM is paused and the redirection scheme starts if the VM is migrated across the Internet. Iproute2 creates an IP tunnel between the IP address at the source and the IP address at the destination. When migration finishes, the dynamic DNS entry is updated, and new connections are directed to the VM's new IP address.

Authors show some remarkable results. They run a web server providing a bulletin board. They evaluate the migration from a source to destination in LAN and WAN. In a LAN, the bulk transfer starts at 25 seconds; the application of deltas occurs after

62 seconds. Xen live migration begins after 125 seconds and the experiment finishes after 225 seconds. The migration of a running web server containing a database of 250 clients has a disruption time of 3.09 seconds. To emulate a wide-area network, they used the Linux traffic shaping interface to limit bandwidth to 5Mbps and added a round-trip latency of 100ms between hosts A and B. This is representative of the connectivity observed between a host in London and a host on the East Coast of the United States. In a WAN environment, the bulk transfer starts after 260 seconds, and the Xen live migration after 2250 seconds. The migration finishes after 3,600 seconds with a disruption time of 68 seconds.

4.4. A Live Storage Migration Mechanism over WAN

In Hirofuchi et al. [2009], a storage access mechanism that supports live VM migration over WAN is proposed. It rapidly relocates VM disks between source and destination sites with the minimum impact on I/O performance. The proposed mechanism works as a storage server of a block-level storage I/O protocol (e.g., iSCSI and NBD). The on-demand fetching and background copying techniques are applied to move online virtual disks among remote sites. This mechanism works for Xen and KVM without any modification to them.

The experiments emulated a WAN environment for remote data centers over the Pacific Ocean (e.g., Tokyo and San Francisco). The experiments showed that the proposed mechanism improved I/O performance from conventional remote storage access, minimizing the performance degradation during disk migration.

The live storage migration mechanism is composed mainly by a target server and a proxy server of a block-level I/O protocol. The Network Block Device (NBD) technology is used to build this prototype, called xNBD. NBD connects the source and destination host nodes to the target and proxy server, respectively, using TCP/IP. Virtual disks are accessed by a VM via block device files (e.g., /dev/nbd0) on a host operating system. Before live migration, it works in the same manner as a normal NBD target server, which redirects I/O requests from the VM to a disk image file. After live migration [Clark et al. 2005] is started, the proposed mechanism works together with memory migration by a VMM and the destination server continues updates of disk blocks of the image file. In the last part of the live migration, the VM is restarted at the destination site, and then I/O operations are performed at the destination site via the proxy server. At this moment, the proxy server starts disk migration. The proxy server continues remote block copies through an NBD connection, until all required blocks are cached at the destination site. After that, the proxy server terminates the NBD connection; the VM does not depend on the target server at the source site. The proxy server copies remote disk blocks using two methods, working in parallel on-demand fetching and background copying. The on-demand method works as follows: if the destination VM tries to read a block that is not cached at the destination VM, the proxy server retrieves the data block from source server. The background copying mechanism copies the blocks that still remain at the source site. The blocks are proactively retrieved before the VM accesses them. The netem module of the Linux kernel emulates the experimental WAN environment. The configuration parameters correspond to a network between Tokyo and the West Coast of the United States. The experimental results show that the proposed mechanism reallocates disks between source and destination sites with an I/O performance comparable to the I/O operations reproduced in the LAN environment.

4.5. CloudNet

The emerging technology of cloud computing [Marston et al. 2009; Mel 2009], which offers the use of services over a network, has changed the scope of resource management from allocating resources on a single server to handling pools of resources within a data center. CloudNet architecture [Wood et al. 2011] as a cloud framework consists of cloud

computing platforms linked with a Virtual Private Network (VPN) infrastructure to provide seamless and secure connectivity between enterprise and cloud data center sites. As a contribution, the authors present some optimizations that minimize the cost of transferring storage and virtual machine memory during migrations over low bandwidth and high-latency Internet links.

CloudNet includes a Virtual Cloud Pool (VCP) abstraction, which allows the seamless connection among servers into a WAN that appears like a single logical pool of resources connected over a LAN. CloudNet uses existing VPN technologies to build a VCP and moves memory and disk state using this structure.

CloudNet implements several WAN optimizations to enable migration over low-bandwidth links. It implements an adaptive live migration algorithm that dynamically tailors the migration of the memory state based on application behavior. It also implements mechanisms such as content-based redundancy elimination and page deltas into the hypervisor to reduce the data volume sent during the migration process. Collectively, these optimizations minimize total migration time, application downtime, and volume of data transferred.

Cloudnet is implemented using the Xen platform and a commercial layer 2 VPN implementation. Another tool employed is the Virtual Private LAN Services (VPLSs), which bridge multiple Multi-Protocol Label Switching (MPLS) endpoints onto a single LAN segment. This allows cloud resources to appear inside the enterprise's own LAN. It is assumed that there is a trusted relationship between the enterprise, the network provider, and the cloud provider.

CloudNet follows the next steps for live migration of VM: Step 1: Establish virtual connectivity between VCP endpoints. Step 2: If storage is not shared, transfer all disk states. Step 3: The transference of the VM memory state to a destination data center is executed while the source VM continues running. Step 4: After disk and memory state have been transferred, the source VM is paused for the remaining transition of memory and processor state to the destination host. This process does not disrupt any active network connections between the application and its clients.

Cloudnet uses the Distributed Replicated Block Device (DRBD) distributed storage system to migrate storage to the destination data center. To reduce the performance impact of this synchronization, CloudNet uses DRBD's asynchronous replication mode during this step. Once the remote disk has been brought to a consistent state, CloudNet switches to a synchronous replication scheme and the live migration of the VM's memory state is initiated. When the migration completes, the new host's disk becomes the primary, and the origin's disk is disabled.

CloudNet uses the Xen's code to achieve memory migration and implements a *Smart Stop and Copy* optimization to reduce the unnecessary number of iterations and minimize the pause time. This point was detected where the number of pages sent is equal to the number of pages dirtied.

The ContentBased Redundancy (CBR) elimination technique is another improvement. It is used to eliminate the redundant data while transferring VM memory and disk state.

After the first iteration, many pages are retransmitted because they have been modified. Another mechanism to reduce bandwidth consumption is to keep a cache of previously transmitted pages and then send only the difference between the cached and current modified page. This type of communication delta is combined with the CBR optimization.

The experimental evaluation was obtained using three data centers spread across the United States. The results show CloudNet's optimizations decreasing memory migration and pause time by 30% to 70% in typical link capacity scenarios; in a set of VM migrations over a distance of 1200km, CloudNet saves 20GB of bandwidth, a 50% reduction.

5. LOAD BALANCING

5.1. Important Concepts

The low utilization of servers increases costs for power consumption and cooling systems. Besides, more physical machines require more floor space and increase maintenance and administration labor. Nowadays, green computing contributes to develop technologies that reduce the energy consumption and CO_2 emissions from cooling systems. Virtualization belongs to these technologies and provides mechanisms to optimize the resource allocation and also an isolation layer that consolidates applications running on several underutilized servers to a reduced number of highly used servers. Further, it is possible to migrate resources from one physical machine to another with minimal disruption. There are many proposals that allow the reallocation of resources to obtain a load balancing and a more optimized resource distribution.

5.2. Dynamic Placement of Virtual Machines

In Bobroff et al. [2007], a dynamic server migration and consolidation algorithm is introduced. The goal of the algorithm is to minimize the cost of running the data center. The cost penalizes overcapacity (low utilization) and overloading, which causes poor application performance and violates contractual Service Level Agreements (SLAs). SLAs are typically expressed as CPU or response time guarantees. The proposed algorithm is based on measuring historical data, forecasting the future demand, and remapping VMs to Physical Machines (PMs), and it is called Measure-Forecast-Remap (MFR).

This approach considers the analysis and classification of resources' workloads. Time series of resource demands were analyzed with a forecasting technique, and it was found that servers that benefit most with a dynamic migration are those that denote a strong variability and autocorrelation in their demand resource distributions.

After finding out the candidate VMs for dynamic management and their future resource demand (for time interval T), the management algorithm tries to reallocate each candidate VM to PM such that the VM's resource requirement should be less than the target PM's capacity. The management algorithm sorts the candidate VMs in descending order of their future resource demand and then makes an attempt to allocate the first VM in order to list the PM as first fit bin packing problem algorithm. If the list of PMs is exhausted without the VM reallocation, the VM is assigned to the PM that has the smallest difference between resource demands and its capacity. The algorithm iterates for all candidate VMs. The experiments are conducted on IBM Blade servers with VMWare ESX 2.5, and migration is programmatically executed with the Software Development Kit (SDK) by VMWare. Results show a reduction of 29% minimum and 50% maximum of active PMs to support a specified rate of SLA violations for a given workload.

5.3. Harmony

In Singh et al. [2008], a load balancing algorithm called VectorDot for handling the hierarchical and multidimensional resource constraints is presented. This algorithm can be used in data centers with the necessity of load balancing across multiple resource layers such as servers, switches, and disk storage. The system called HARMONY continuously monitors the resource usages of servers, network switches, and storage nodes in the data center. It launches live migrations of VMs and virtual disks if a hotspot is presented. Overloaded nodes migrate to an underutilized node. The VectorDot algorithm is based on the Toyoda method for multidimensional knapsacks. Storage virtualization refers to managing physical storage into virtualized containers called virtual disks (Vdisks) that can be used by applications. This kind of storage allows dynamic growth or shrink of storage. HARMONY handles block-level virtualization

where each data block is mapped to one or more storage subsystems, but it appears the application resides on a single volume. A single file system can be formed by multiple file systems with a common namespace. HARMONY uses a block virtualization solution called IBM SAN Volume Controller (SVC). This SVC is configured in the I/O paths of three ESX servers. There is a storage virtualization manager, IBM SVC's CLI interface, which interacts with HARMONY to obtain virtual storage configuration and handles nondisruptive data migration. On the other hand, HARMONY interacts with a server virtualization manager to obtain configuration and performance information about VMs and physical servers and to manage live migration of VMs. The virtualization manager is VMWare Virtual Center.

If a node is overloaded, exceeding some threshold, HARMONY initiates the optimization planning component VectorDot. It generates recommendations for migrating one or more VMs or Vdisks to alleviate the hotspot. These are processed by the Virtualization Orchestrator by using appropriate server and storage virtualization managers.

The node could be a server node, storage node, or switch node. A server overload can be caused by an excessive CPU, memory, network, or disk I/O usage. A storage node overload can occur due to excessive storage space usage or disk I/O rates. A switch node (or a switch port) overload can occur if a large amount of I/O is being pushed through it. If a node exceeds the threshold along any of its resource dimensions, the node is considered to be an overloaded node or trigger node. To solve the overload, the load balancing algorithm in HARMONY moves one or more VMs or Vdisks from an overloaded node to underloaded nodes.

In an example of virtual machine and storage migration with HARMONY, the authors obtained the following results: a VM2 and its storage are migrated to a different physical server. VM2 is configured with 3GHz CPU, 1.2GB memory, and a storage volume of 20GB. The Postmark benchmark was running in the VM. During the VM migration Postmark transaction, throughput drops by around 11.9%. This happens due to the CPU congestion at source and destination servers caused by live migration. Storage live migration does not cause CPU congestion and has a smaller overhead of 7.6%, it occurs for a much more prolonged period of time, and 20GB of data is migrated to another physical machine. HARMONY is highly scalable, producing allocation and load balancing recommendations for over 5,000 VMs and Vdisks on over 1,300 nodes in less than 100 seconds.

5.4. pMapper

pMapper [Verma et al. 2008] is an application management middleware for workload placement. It takes into account the power and migration costs. pMapper handles three different managers, and an arbitrator ensures consistency between the three actions. Soft actions like VM resizing and idling are communicated by the Performance Manager. The Power Manager triggers power management at a hardware layer, while the Migration Manager interfaces with the Virtualization Manager to trigger consolidation through VM live migration.

The resource management flow of pMapper starts with the Monitoring engine, which collects the current performance and power characteristics of all the VMs and physical servers in the farm. The Performance Manager finds out the current performance and recommends a set of target VM sizes based on the SLA goals. If the target VM sizes are different from the current VM sizes, it presents an estimate of the benefit due to resizing. Similarly, the Power Manager looks at the current power consumption and may suggest throttling (by Dynamic Voltage and Frequency Scaling, DVFS, or explicit CPU throttling).

The Arbitrator implements an algorithm to compute the best placement for a VM based on the estimates received from the Performance, Power, and Migration managers.

pMapper is implemented for VMWare ESX-based platforms. The VMWare Virtual Center is used as the Virtualization Manager. The VI API provided by VMWare ESX 3.0 is used to communicate with the Virtualization Manager and execute migration actions. Throttling actions are executed by the IBM Active Energy Manager interface.

6. RELATED TOPIC: AUTOMATIC MANAGEMENT OF LIVE MIGRATION

6.1. Important Concepts

The automatic live migration may be caused by the hotspots' detection. A hotspot can be defined as an access point with a high demand of resources, such as an overloaded server; if some resources like the processor, network, or memory exceed a threshold; or if Service Level Agreement (SLA) violations during a period are launched. An important characteristic for a well-managed data center is its ability to avoid hotspots. Overloaded nodes (servers, storage, or network switches) often lead to performance degradation and are vulnerable to failures. To alleviate such hotspots, load must be migrated from the overloaded resource to an underutilized one.

6.2. Sandpiper

In Wood et al. [2007], a mechanism to automatically migrate virtual machines is presented. This system is called Sandpiper and implements a hotspot detection algorithm that determines when, what, and where to migrate virtual machines. Sandpiper currently uses Xen's migration mechanism proposed in Clark et al. [2005].

Sandpiper employs two strategies for virtual machine migration in large data centers: black-box and gray-box strategies. The black-box approach uses data generated by SO and the gray-box approach implements application-level statistics. These techniques automate the tasks of monitoring system resource usage, detecting hotspots, determining a new mapping, and initiating the necessary migrations.

Sandpiper assumes a large cluster of possibly heterogeneous servers. The hardware configuration of each server (CPU, network interface, disk, and memory) is known by Sandpiper. Each physical server runs a virtual machine monitor and one or more virtual machines. Sandpiper uses Xen to build this structure.

All storage is assumed to be on a Network File System (NFS) or a Storage Area Network (SAN), so it is not necessary to move the disk state during VM migrations. Sandpiper runs a component called the nucleus on each physical server; the nucleus runs inside a special virtual server (domain 0 in Xen) and collects the resource usage statistics on that server. It uses a monitoring engine that gathers processor, network interface, and memory swap statistics for each virtual server.

The gray-box approach implements a daemon within each virtual server to collect OS-level statistics. The nuclei periodically deliver these statistics to the Sandpiper control plane. The control plane runs on a specific node. It includes three components: a profiling engine, a hotspot detector, and a migration manager. The profiling engine uses the statistics from the nuclei to construct resource usage profiles for each virtual server and add profiles for each physical server.

The monitoring engine tracks the usage for each resource of each virtual machine (processor, network, and memory) in a measurement interval and reports these statistics to the control plane when the interval ends. Much of the required information can be determined directly from the Xen hypervisor or by monitoring events within domain 0 of Xen.

Sandpiper supports gray-box monitoring, using a lightweight monitoring daemon that is installed within each virtual server. In Linux, the monitoring daemon uses the /proc interface to collect OS-level statistics of CPU, network, and memory usage.

A profile is a compact description of the server's resource usage over a sliding time window W. Three black-box profiles are maintained by the virtual server: CPU utilization, network bandwidth utilization, and swap rate (i.e., page fault rate). If gray-box monitoring is permitted, four additional profiles are maintained: memory utilization, service time, request drop rate, and incoming request rate. Similar profiles are also maintained for each physical server, which indicate the global usage of resident VMs.

Hotspot detection is executed on each physical server based on the black-box approach, so a hotspot is flagged if the total CPU, network usage, or total swap activity exceeds a threshold. On the other hand, the SLA violations are detected on each virtual server based on the gray-box approach; a hotspot is flagged if the memory utilization of the VM exceeds a threshold or if the response time or the request drop rate exceeds the SLA-specified values.

Sandpiper works by moving load from the most overloaded servers to the least overloaded servers, minimizing data copying during migration. It considers the overload on one or more of three dimensions (CPU, network, and memory) of a VM, and the volume of a physical or virtual server is defined as the product of its CPU, network, and memory loads:

$$Vol = \frac{1}{1-cpu} * \frac{1}{1-net} * \frac{1}{1-mem},$$

where cpu, net, and mem correspond to the use of that resource for the virtual or physical server. The volume indicates the degree of (over)load on multiple dimensions.

Migration Phase: To determine which VMs to migrate, the algorithm orders physical servers in decreasing order of their volumes. VMs are considered in decreasing order of their Volume-to-Size Ratio (VSR), where VSR is defined as $\frac{Volume}{Size}$; size is the memory footprint of the VM.

The migration algorithm locates the VM with the highest VSR from the server with the highest volume and determines if the VM can be relocated on the least loaded physical server. Migration is possible just if the selected server has enough unused CPU, network, and memory resources to host the VM. If there are not enough resources, the algorithm analyzes the next least loaded server and so on, until a new server host is found. If the algorithm cannot find a physical server to host the VM with the highest VSR, then the algorithm moves on to the next highest VSR VM and attempts to move it in a similar way. The process is repeated until the use of all resources on each physical server decreases below their thresholds.

Swap Phase: When there are not sufficient free resources on less loaded servers to eliminate a hotspot, the migration algorithm considers VM swaps. A swap exchanges a high-VSR virtual machine from a loaded server with one or more low-VSR virtual machines from an underloaded server. The swap releases resources, but with less impact than a one-way migration.

The experiments with Sandpiper in Wood et al. [2007] compare the black-box approach with the gray-box approach. The scenario was configured with a 6-second measurement interval. Once a hotspot is flagged, the gray-box approach eliminates it within 40 seconds with two migrations, while the black-box approach requires 110 seconds and three migrations. As a result, the decrease of overloaded servers by the use of Sandpiper was obtained within a reasonable time.

6.3. VirtualPower

The VirtualPower [Nathuji and Schwan 2007] approach operates in virtualized platforms. It makes it possible to control and coordinate the application of several power management polices inside VMs. The hardware usage by guest is limited with software

and hardware techniques that leverage underlying hardware support like the processor frequency scaling. VirtualPower Management (VPM) is based on the Xen hypervisor and implements multiple system-level abstractions including VPM states, channels, mechanisms, and rules. VirtualPower exports a rich set of virtualized power states, called VirtualPower Management states (VPM states); then the guest takes actions according to the VM-level power management policies. Power states are changed by privileged actions, such as the modification of the Model Specific Registers (MSRs). VirtualPower intercepts the power management policies built into guest VMs, but they are "hints" rather than executable commands. So, VirtualPower intercepts attempts to execute privileged actions on the interface for power management (Advanced Configuration and Power Interface, ACPI) and maps them into a VPM channel. These channels provide hints to VirtualPower's VPM rules running on Dom0. Finally, rules are based on a set of VPM mechanisms that implement management methods across heterogeneous hardware platforms. The mechanisms supported by VirtualPower include hardware scaling, soft scaling, and consolidation. VPM rules implement a tiered policy approach: local PM-L policies perform actions corresponding to resources on local platforms, and a higher-level PM-G policy component uses rules responsible for coordinating global decisions.

The consolidation mechanism is an example of power management through policies that handle the idle, standby, and sleep states. Consolidation can be used to minimize the number of currently active resources so that standby and sleep states can be used. Underused servers can be consolidated into fewer hosts and offloaded hosts can be placed into a sleep mode. VirtualPower provides policies that combine consolidation with migration to perform efficient allocations. The employ of VPM management techniques in heterogeneous server systems provides 34% of power savings.

7. RELATED TOPIC: RECORD/REPLAY

7.1. Important Concepts

Logging and replay are widely used for the recovering state. The procedure starts from a checkpoint of a prior state, then rolls forward, replaying events from the log to reach the desired state [Dunlap et al. 2002]. Replaying a virtual machine requires logging the nondeterministic events that affect the virtual machine's computation. These log records guide the virtual machine as it re-executes (rolls forward) from a checkpoint. Deterministic events (e.g., arithmetic, memory, branch instructions) do not need to be logged; the virtual machine will re-execute these events in the same way during replay as it did during the original execution. To replay an execution, any nondeterministic event that affects the state of the system is logged and replayed. For virtual machines, this includes logging virtual interrupts; input from virtual devices such as the virtual keyboard, network, or real-time clock; and the results of nondeterministic instructions such as those that read the processor's Time-Stamp Counter (TSC). Nondeterministic events fall into two categories: time and external input. Time refers to the exact point in the execution stream at which an event takes place. External input refers to data received from a nonlogged entity, such as a human user or another computer.

Recording fidelity [Andrica and Candea 2011] quantifies recorded events, the log is relatively small and replay is close to ideal [Surie et al. 2008]. High-fidelity recording requires that all events be recorded. Replaying fidelity correctly quantifies played-back events, and high-fidelity replaying requires that all interactions be realistically simulated.

Techniques for replaying single processor systems are widely known. But with the increasing use of multicore processors, execution replay on multiprocessor systems has become more important. When replaying shared-memory systems, reads from memory

by one processor are affected by writes of another processor. Since these reads and writes may happen in any arbitrary interleaving order, this introduces fine-grained nondeterminism into any shared memory operation [Dunlap et al. 2008].

The record/replay technique has to face many challenges such as maximizing trace completeness and the detail level of an event register and minimizing distortion. The information collected should faithfully represent the program being recorded, reducing log file size and being fast, inexpensive, and easy to operate. To save log files on disk represents an overhead in performance but introduces an option to ensure the execution of deterministic events.

7.2. Live Migration of VM Based on Full System Trace and Replay

CR/TR-Motion [Liu et al. 2009] is an approach based on checkpointing/recovery and trace/replay technology. The execution trace logs are obtained from source VM and synchronically replayed on the target VM. The objective of this live migration mechanism is to minimize downtime, network bandwidth consumption, and the total migration time. Live migration phases are as follows:

(1) A target host with sufficient resources is selected. The migration requirement from the source VM to the target VM is made. (2) The source VM changes to a frozen state and saves the actual system state (virtual memory, CPU registers, memory from external devices, and virtual disk) to a file in copy-on-write mode. After checkpointing, the source VM continues running. (3) In the first round of transference, the checkpoint file is copied from host source to destination, and source VM continues on execution and saving nondeterministic system events in a log file. The following iterations copy the log file generated during the previous transfer round. At the same time, destination VM replays the received log files. (4) The iterative process is executed until the log file generated during the previous transfer round is reduced to a specified size defined as a threshold value Vthd (1KB in experiments). (5) The source VM is suspended and the remaining log file is transferred to the destination host. After the last log file is replayed, there is a consistent replica of the VM at both the source and destination. The VM at the source is still considered to be primary and may be resumed in case of failure. (6) The source VM informs the destination VM that it has successfully synchronized their running states. The source host acknowledges this message and then all its network traffic is redirected to VM destination. Now the source VM may be discarded. (7) The destination VM is activated.

The copy-on-write mechanism reduces the checkpoint time. During the iterative log transference, the execution log of the source VM is copied, but not the dirty memory pages, and this may decrease the amount of data transferred while synchronizing the two VMs' running state.

7.3. Record/Replay for Replication

7.3.1. Important Concepts. Record/replay is a technique that is able to restore the complete state of a VM even in more than one location.

7.3.2. Remus. Remus [Cully et al. 2008] migrates running VMs between physical hosts and replicates snapshots of an entire running OS instance at very high frequencies (as often as every 25ms) between a pair of physical machines.

Remus runs paired servers in an active-passive configuration. Remus utilizes three techniques in its implementation: First, virtualization is used to run a pair of systems in lock-step; external events are carefully injected into both the primary and backup VMs, which results in identical states. Second, Remus does not attempt to make a deterministic computation. The state of the replica needs to be synchronized with the primary only when the output of the primary has become externally visible. Instead

of letting the normal output stream dictates when synchronization must occur, Remus saves in a buffer output (network and disk) up to a more convenient time, performing computation speculatively ahead of synchronization points. And third, an asynchronous replication is implemented; buffering output at the primary server allows replication to be performed asynchronously. The primary host can resume execution at the moment its machine state has been captured, without waiting for acknowledgment from the remote end. The primary server remains productive while synchronization with the replicated server is asynchronously performed.

The high-availability mechanism of Remus is based on frequent checkpoints of an active VM to a backup VM. The VM image backup is resident in memory and may begin execution immediately if failure was suffered by the active system. The VM backup is not always consistent with the primary VM; the network output is saved to a buffer until the state is synchronized on the backup. The cycle that includes checkpoint, buffer, and release buffer occurs frequently (up to 40 times per second). Disk replication is asynchronously maintained through writes to the backup. Writes to the primary disk are held in a RAM buffer until the corresponding checkpoint arrives. After that, the checkpoint is acknowledged to the primary, which then releases the outgoing network traffic, and the buffered disk writes are applied to the backup disk. The backup VM is assumed as active until a failure is detected. Multiple backups can be done.

Remus implements a failure model that considers handling the failure of any host. If primary and backup hosts fail at the same time, Remus will leave the system in a consistent state. Thus, the outputs just can be visible until the associated system state has been committed to the replica.

A failure detector within the checkpoint strategy has been implemented. If a timeout is detected while the primary is waiting to commit requests, the primary will assume that the VM backup has crashed and it will disable protection. On the other hand, if a timeout is detected during a checkpoint, the backup will assume that the primary has crashed and it will resume the execution from the most recent checkpoint.

In the performance evaluation of Cully et al. [2008], it was found that in a general-purpose task, Remus incurs approximately a 50% performance penalty when checkpointed 20 times per second.

Cully et al. applied several tests to Remus; within representative tests it can be mentioned that:

—The kernel compile test measures the time required to build Linux kernel version 2.6.18 using the default configuration and the bzImage target. This is a test for CPU, memory, and disk performance. The checkpoint was configured at rates of 10, 20, 30, and 40 times per second, compared to a baseline compilation in an unprotected virtual machine. Total measured overhead at each of these frequencies was 31%, 52%, 80%, and 103%, respectively.

7.3.3. Lightweight Live Migration. High availability is a valuable feature for service clusters and cloud computing, and sometimes it is more valuable than performance. Live migration of virtual machines is a mechanism to achieve availability. But the use of checkpointing can introduce significant overhead. In the work of Jiang et al. [2010], a Lightweight Live Migration (LLM) mechanism is presented. This work is developed on Xen and compared with Remus [Cully et al. 2008]. Test cases show that LLM overcomes Remus in terms of network delay and overhead.

In LLM, the machine that provides regular services is called the primary machine, and the replica machine is called the backup machine. One challenge of live migration is to minimize the downtime.

LLM integrates two ideas: checkpointing and input replay with the objective of improving migration for applications with intensive network workload. The primary

machine migrates CPU, memory, and disk status updates to the backup machine at low frequency and the service requests from network clients at high frequency.

In this work, a hardware failure model considers the following assumptions: any correct server can detect if another server has failed and it is used for stable storage that contains the last correct state of the crashed server.

Remus uses the block/commit case where the output packets are blocked until a checkpoint is acknowledged. LLM releases the response packets immediately. During a transference with low checkpointing frequency, network clients may experience very long delays with the block/commit mechanism of Remus.

The Netfilter system in the IPv4 protocol stack makes it possible to filter and copy network packets to and from a specific guest VM.

The time sequence of migrating the checkpointed resources is described as follows:

—The guest VM is paused. During this suspension period, all the status updates of CPU/memory/disk are collected and stored in a migration buffer.
—Once the guest VM is resumed, the content stored in the migration buffer is migrated first to the replica machine.
—Then, the network buffer migration starts at high frequency until the guest VM is suspended again. At the end of each network buffer migration cycle, it is known which packets need to be replayed on the backup machine and which ones need to be responded to the clients.

Any time a failure happens to a primary machine, the backup machine will continue the execution of VM from the latest checkpointed status. Then, it will replay the requests after the first boundary to achieve consistency, and it will respond to those unresponded requests after the second boundary.

LLM was evaluated and compared with Remus in terms of its correctness, downtime, delay for clients, and overhead under various checkpointing periods in Jiang et al. [2010]. LLM tested some case studies. For example, a flood ping is launched over the primary machine. This experiment is called “HighNet” because of the intensity of the network load. With HighNet, LLM obtains shorter downtimes than Remus. This is because there are too many duplicated packets to be served by the backup machine in Remus from the client side.

In this experiment, the network delay was estimated too. Results show that most of the time, LLM has a much shorter network delay than Remus.

7.4. Lightweight Virtualization Solution

7.4.1. Important Concepts. The lightweight virtualization [Vaughan-Nichols 2006] approaches allow a single operating system to run several instances of the same OS or different OSs. They require fewer CPU and memory resources, which is why the technology is called “lightweight” virtualization. Containers do not emulate any of the underlying hardware. Instead, the virtualized OS or application talks to the host OS, which then makes the appropriate calls to the real hardware.

7.4.2. OpenVZ. Container-type virtualization allows running multiple isolated sets of processes, known as containers, under a single kernel instance. Each isolated instance has the possibility to execute a checkpoint, which saves the complete state of a container and later restarts it. In Mirkin et al. [2008], a mechanism used for live migration is presented. Checkpointing and restart are implemented as loadable kernel modules.

The container’s processes are saved in a consistent state. Dependencies such as process hierarchy, identifiers (PGID, SID, TGID, etc.), and shared resources (open files, SystemV IPC objects, etc.) are saved and restored during restart.

Networking is a resource managed by checkpointing and restart. During checkpointing and restart, the network should be disabled. All incoming packets are dropped, the processes are frozen, and they do not process incoming packets.

Live migration is implemented using the checkpointing and restart feature. The algorithm does not require any special hardware like SAN or iSCSI storage and follows the next steps:

(1) Transfer the container's file system to the destination server. This can be done using rsync utility.
(2) Freeze all the processes and disable networking.
(3) Collect all the processes and their resources and save them to a file on disk. This step can take a long time.
(4) There is a second container's file system synchronization. During the first synchronization, the source container is in operation, and some files could be modified and the destination container could hold outdated files. The dump file is transferred to the destination server.
(5) Restart the container on the destination server. At this phase, a container and its processes are created in a destination server extracting information from the dump file. Processes are in a frozen state.
(6) The destination container resumes execution on the destination server.
(7) Stop the original container.
(8) Destroy the original container.

8. CURRENT TRENDS

Live migration allows that VMs can be moved between physical hosts in response to changing workloads, so data centers can have a dynamic infrastructure, which can be built with virtualization technology pools computing resources and by allocating these resources on demand or with resource utilization. An important virtualization trend is the support for the dynamic data center. Many organizations are already using virtualization benefits through server consolidation. The next phase to server consolidation is the ability to move virtualized resources between different hosts with no downtime [Singh et al. 2008]. Computation is no longer localized within a data center; now resources can be spread across multiple data centers around the world. The development of live migration over a WAN is an open research area. This kind of migration faces challenges like the transference of disk images and memory over low bandwidth and minimizing the performance degradation of the migrating VM system.

Nowadays, virtualization and cloud computing are two linked topics [Armbrust et al. 2009, 2010]. Using virtualization to create a private cloud is an important virtualization trend. Lastly, an important research direction is to develop new technologies that enable cloud computing to support large distributed applications despite limitations in network bandwidth and latency [Sripanidkulchai et al. 2010]. Today, when cloud providers need to perform fixes or maintenance on a physical machine, they inform users to shut down and reallocate their instances to a different location. Cloud providers could use live migration to move instances while they are running. Although some providers expose an interface for users to use live migration within their own cloud, live migration to a different cloud is a research opportunity [Williams et al. 2012].

On the other hand, smartphones are a technology in continuous renovation. Smartphones provide pervasive computing, but their hardware is limited in terms of computation, memory, and energy reserves. This is a research direction that can be developed to create live migration. An engine may augment the smartphone's capabilities by off-loading some tasks to a nearby computer, where they are executed in clone image

Table I. Migration Characteristics

	Process Migration	Memory Migration	Replay	Checkpoints	Multi-backup
Zap	•	•		•	•
Autopod	•	•		•	•
VMotion		•			
Xen Live Migration		•			
KVM				•	
Post-Copy Live Mig.		•			
ISR				•	
Capsule				•	
SoulPad				•	
Live Wide-Area Mig.		•		•	
SandPiper			•	•	•
CT/TR Motion			•	•	
Remus		•		•	•
LLM			•	•	

Table II. Migration Characteristics-Continuation

	Storage Migration	Hypervisor Required	Incremental Migration	Automatic Migration
Zap	•			
Autopod	•			•
VMotion		•[5]		
Xen Live Migration		•[1]		
KVM	•	•[3]		
Post-Copy Live Mig.	•	•[1]		
ISR	•	•[1,2,3]		
Capsule	•	•[5]	•[9]	
SoulPad	•	•[6]		
Live Wide-Area Mig.	•	•[7]		
SandPiper		•[1]	•	•[8]
CT/TR Motion	•	•[4]		
Remus	•	•[1]		
LLM		•[1]		

Note: [1]Xen, [2] VirtualBox, [3]KVM, [4]UMLinux mod, [5]VMware, [6]Knoppix, [7]XenoServer, [8]Migration starts if a threshold is reached, [9]Incremental backup.

of the device, after reintegrating the results in the smartphone execution [Chun and Maniatis 2009].

9. CONCLUSIONS

In this article, a survey of selected publications related to migration mechanisms of virtual machines is presented. In Tables I and II, we summarize the main characteristics of VM migration mechanisms. Process migration indicates whether the migration mechanism is based on process migration. Memory migration indicates if the mechanism is based on memory migration. Replay denotes if migration saves in log records the state of a VM. Checkpointing determines if migration uses a checkpoint to take a snapshot of the entire state of a VM. Multibackup indicates if the replication mechanism maintains more than one VM copy at the same time. Storage migration indicates if migration considers a storage migration mechanism. Hypervisor identifies if a hypervisor is necessary for the migration mechanism. Incremental migration indicates

if a copy of the VM can be reassumed from a specific execution point. Automatic migration notes if the migration mechanism is auto-started when a specific condition is reached. The necessity of migration of processes appeared since the creation of distributed systems. With the development of the virtualization and the incorporation of this technology as a tool to achieve high-availability systems, the migration mechanisms were greatly improved. In this article, we have considered that the migration mechanisms have been divided into three main branches. The first one, process migration, which was born along with the distributed systems, is the beginning of the VM's migration. The mechanisms based on process migration have some characteristics, such as complexity, performance, transparency, fault resilience, scalability, and heterogeneity, that have a major impact on the effectiveness and deployment of process migration. The complexity of implementation and dependency on an operating system are among the obstacles to the wider use of process migration [Milojičić et al. 2000]. Process migrations can be made in the range of seconds, and this time can increase to minutes.

The migration mechanisms based on memory migration are efficient and faster than others; they migrate virtual machines from one physical machine to the other in a short time (60 ms in the best test time). However, the majority of these hypervisors do not perform storage migration, and it is necessary for a SAN or NAS platform to handle the file system, which is not always efficient for geographically separated file systems. The increasing necessity of sharing information and the new vision where the information is no longer localized within a data center but rather can be dispersed across geographical distances has created migration mechanisms to transfer VM across a WAN. These approaches transfer local disk and network connections along with the memory state within a negligible time. Diverse optimizations to efficiently transfer memory and disk [Travostino 2006; Bradford et al. 2007; Hines et al. 2009] have been developed. Besides, researches have generated live migration mechanisms to get load balancing and efficient resource consumption [Wood et al. 2007; Nathuji and Schwan 2007]. Suspend/resume migration allows the transference of a VM across a WAN. The source VM is suspended and retrieved from the network when the VM is resumed. This technology has developed the disconnected operation concept, where the users keep working with the cache content. Suspend/resume is the base for mechanisms that allow carrying an auto-configuring operating system along with a suspend VM on a small portable device [Cáceres et al. 2005] or a mobile device user executing a service software on a nearby cloudlet [Satyanarayanan et al. 2009].

Cloud computing technology is attracting new users around the world, and there are many scientific projects that are adopting this technology as their platform [NERSC 2011; Vöckler et al. 2011; Portal 2010; Gunarathne et al. 2010; Li et al. 2009; Iványi and Topping 2011]. These projects demand the use of intensive computational power, access to fault-tolerant mechanisms, and management of virtual machine replicas. They require the use of hundreds of nodes to provide the necessary computational power. Currently, the hypervisors based on memory have become the most widely used around the world, due to the generalized use of these types of hypervisors as the underlying platform in systems that provide cloud computing services. For example, Amazon's Web Services handle instances of Xen and VMWare; Futuregrid manages virtual machines in Xen, KVM, or VMWare; and NERSC uses Xen, KVM, or UML. However, the use of live migration mechanisms in cloud computing is an open research area. Cloud computing providers still do not create a live migration mechanism that does not interrupt user activities [Williams et al. 2012].

Recently, the record-and-replay strategy has been widely used in migration of virtual machines, due to the nondeterministic events, which can be recorded and reproduced

after. This strategy includes storage migration with efficient migration time. It can be noticed that record-and-replay strategies allow incremental migrations to be applied to virtual machines and specific filters to be applied to restore just the information that the user needs. This type of migration has great possibility of development since the experiments in this area have shown data consistency, efficient time performance during the log and replay processes, and security in which the whole state of a VM would be migrated, including memory, storage, and network connections.

Several migration mechanisms and their respective experimental results have been presented. It is hard to compare the performance of various migration mechanisms because the implementations were done on a number of different architectures. Tests have been executed under different circumstances; elements such as hardware, size samples, benchmarking, and report units are distinct among the migration methods. It is also hard and inaccurate to normalize the performance.

REFERENCES

2009. The NIST Definition of Cloud Computing. *National Institute of Standards and Technology* 53, 6 (2009), 50. http://csrc.nist.gov/groups/SNS/cloud-computing/cloud-def-v15.doc

Silviu Andrica and George Candea. 2011. WaRR: A tool for high-fidelity web application record and replay. In *Proceedings of the 2011 IEEE/IFIP 41st International Conference on Dependable Systems & Networks (DSN'11)*. IEEE Computer Society, Washington, DC, 403–410. DOI:http://dx.doi.org/10.1109/DSN.2011.5958253

Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. 2010. A view of cloud computing. *Commun. ACM* 53, 4 (April 2010), 50–58. DOI:http://dx.doi.org/10.1145/1721654.1721672

Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy H. Katz, Andrew Konwinski, Gunho Lee, David A. Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. 2009. *Above the Clouds: A Berkeley View of Cloud Computing*. Technical Report UCB/EECS-2009-28. EECS Department, University of California, Berkeley. Retrieved from http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.html

Muhammad Atif and Peter Strazdins. 2009. Optimizing live migration of virtual machines in SMP clusters for HPC applications. In *Proceedings of the IFIP International Conference on Network and Parallel Computing Workshops*. 51–58. DOI:http://dx.doi.org/10.1109/NPC.2009.32

Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. 2003. Xen and the art of virtualization. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP'03)*. ACM, New York, NY, 164–177. DOI:http://dx.doi.org/10.1145/945445.945462

N. Bobroff, A. Kochut, and K. Beaty. 2007. Dynamic placement of virtual machines for managing SLA violations. In *Proceedings of the 10th IFIP/IEEE International Symposium on Integrated Network Management (IM'07)*. 119–128. DOI:http://dx.doi.org/10.1109/INM.2007.374776

Robert Bradford, Evangelos Kotsovinos, Anja Feldmann, and Harald Schiöberg. 2007. Live wide-area migration of virtual machines including local persistent state. In *Proceedings of the 3rd International Conference on Virtual Execution Environments (VEE'07)*. ACM, New York, NY, 169–179. DOI:http://dx.doi.org/10.1145/1254810.1254834

Ramón Cáceres, Casey Carter, Chandra Narayanaswami, and Mandayam Raghunath. 2005. Reincarnating PCs with portable SoulPads. In *Proceedings of the 3rd International Conference on Mobile Systems, Applications, and Services (MobiSys'05)*. ACM, New York, NY, 65–78. DOI:http://dx.doi.org/10.1145/1067170.1067179

Ramesh Chandra, Nickolai Zeldovich, Constantine Sapuntzakis, and Monica S. Lam. 2005. The collective: a cache-based system management architecture. In *Proceedings of the 2nd Conference on Symposium on Networked Systems Design & Implementation - Volume 2 (NSDI'05)*. USENIX Association, Berkeley, CA, 259–272.

Prabhakar Changarti. 2007. *Xen Virtualization: A Practical Handbook* (1st ed.). Packt Publishing Ltd., Birmingham, UK.

Peter M. Chen and Brian D. Noble. 2001. When virtual is better than real. In *Proceedings of the 8th Workshop on Hot Topics in Operating Systems (HOTOS'01)*. IEEE Computer Society, Washington, DC, 133.

David Chisnall. 2007. *The Definitive Guide to the Xen Hypervisor*. Prentice Hall, Upper Saddle River, NJ.

Byung-Gon Chun, Sunghwan Ihm, Petros Maniatis, Mayur Naik, and Ashwin Patti. 2011. CloneCloud: elastic execution between mobile device and cloud. In *Proceedings of the 6th Conference on Computer systems (EuroSys'11)*. ACM, New York, NY, 301–314. DOI:http://dx.doi.org/10.1145/1966445.1966473

Byung-Gon Chun and Petros Maniatis. 2009. Augmented smartphone applications through clone cloud execution. In *Proceedings of the 12th Conference on Hot Topics in Operating Systems (HotOS'09)*. USENIX Association, Berkeley, CA, 8–8. http://dl.acm.org/citation.cfm?id=1855568.1855576

Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. 2005. Live migration of virtual machines. In *Proceedings of the 2nd Conference on Symposium on Networked Systems Design & Implementation (NSDI'05)*. USENIX Association, Berkeley, CA, 273–286.

Brendan Cully, Geoffrey Lefebvre, Dutch Meyer, Mike Feeley, Norm Hutchinson, and Andrew Warfield. 2008. Remus: High availability via asynchronous virtual machine replication. In *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation (NSDI'08)*. USENIX Association, Berkeley, CA, 161–174.

Rogier Dittner and David Rule. 2007. *The Best Damn Server Virtualization Book Period: Including VMware, Xen, and Microsoft Virtual Server*. Syngress Publishing.

George W. Dunlap, Samuel T. King, Sukru Cinar, Murtaza A. Basrai, and Peter M. Chen. 2002. ReVirt: enabling intrusion analysis through virtual-machine logging and replay. *SIGOPS Oper. Syst. Rev.* 36, SI (December 2002), 211–224. DOI:http://dx.doi.org/10.1145/844128.844148

George W. Dunlap, Dominic G. Lucchetti, Michael A. Fetterman, and Peter M. Chen. 2008. Execution replay of multiprocessor virtual machines. In *Proceedings of the 4th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE'08)*. ACM, New York, NY, 121–130. DOI:http://dx.doi.org/10.1145/1346256.1346273

R. P. Goldberg. 1974. Survey of virtual machine research. *IEEE Computer Magazine* (June 1974), 34–45.

Thilina Gunarathne, Tak-Lon Wu, Judy Qiu, and Geoffrey Fox. 2010. Cloud computing paradigms for pleasingly parallel biomedical applications. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing (HPDC'10)*. ACM, New York, NY, 460–469. DOI:http://dx.doi.org/10.1145/1851476.1851544

Diwaker Gupta, Ludmila Cherkasova, Rob Gardner, and Amin Vahdat. 2006. Enforcing performance isolation across virtual machines in Xen. In *Proceedings of the ACM/IFIP/USENIX 2006 International Conference on Middleware (Middleware'06)*. Springer-Verlag, New York, NY, 342–362.

Jacob Gorm Hansen and Eric Jul. 2004. Self-migration of operating systems. In *Proceedings of the 11th ACM SIGOPS European Workshop (EW'11)*. ACM, New York, NY, Article 23. DOI:http://dx.doi.org/10.1145/1133572.1133616

Michael R. Hines, Umesh Deshpande, and Kartik Gopalan. 2009. Post-copy live migration of virtual machines. *SIGOPS Oper. Syst. Rev.* 43, 3 (July 2009), 14–26. DOI:http://dx.doi.org/10.1145/1618525.1618528

Takahiro Hirofuchi, Hidemoto Nakada, Satoshi Itoh, and Satoshi Sekiguchi. 2011. Reactive consolidation of virtual machines enabled by postcopy live migration. In *Proceedings of the 5th International Workshop on Virtualization Technologies in Distributed Computing (VTDC'11)*. ACM, New York, NY, 11–18. DOI:http://dx.doi.org/10.1145/1996121.1996125

Takahiro Hirofuchi, Hidemoto Nakada, Hirotaka Ogawa, Satoshi Itoh, and Satoshi Sekiguchi. 2009. A live storage migration mechanism over wan and its performance evaluation. In *Proceedings of the 3rd International Workshop on Virtualization Technologies in Distributed Computing (VTDC'09)*. ACM, New York, NY, 67–74. DOI:http://dx.doi.org/10.1145/1555336.1555348

Khaled Z. Ibrahim, Steven Hofmeyr, Costin Iancu, and Eric Roman. 2011. Optimized pre-copy live migration for memory intensive applications. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC'11)*. ACM, New York, NY, Article 40, 11 pages. DOI:http://dx.doi.org/10.1145/2063384.2063437

P. Iványi and B.H.V. Topping (Eds.). 2011. *Threat Detection in an Urban Water Distribution Systems with Simulations Conducted in Grids and Clouds*. Ajaccio, Corsica, France. DOI:http://dx.doi.org/10.4203/ccp.95.93

Bo Jiang, Binoy Ravindran, and Changsoo Kim. 2010. Lightweight live migration for high availability cluster service. In *Proceedings of the 12th International Conference on Stabilization, Safety, and Security of Distributed Systems (SSS'10)*. Springer-Verlag, Berlin, 420–434. Retrieved from http://dl.acm.org/citation.cfm?id=1926829.1926865.

James J. Kistler and M. Satyanarayanan. 1992. Disconnected operation in the Coda File System. *ACM Trans. Comput. Syst.* 10, 1 (Feb. 1992), 3–25. DOI:http://dx.doi.org/10.1145/146941.146942

Avi Kivity, Yaniv Kamay, Dor Laor, Uri Lublin, and Anthony Liguori. 2007. KVM: The Linux virtual machine monitor. In *Proceedings of the Ottawa Linux Symposium*. 225–230. Retrieved from http://www.kernel.org/doc/ols/2007/ols2007v1-pages-225-230.pdf.

Nadir Kiyanclar. 2005. A survey of virtualization techniques focusing on secure on-demand cluster computing. *CoRR* abs/cs/0511010 (2005).

Michael Kozuch and M. Satyanarayanan. 2002. Internet suspend/resume. In *Proceedings of the 4th IEEE Workshop on Mobile Computing Systems and Applications (WMCSA'02)*. IEEE Computer Society, Washington, DC, 40.

Michael A. Kozuch, Michael Kaminsky, and Michael P. Ryan. 2009. Migration without virtualization. In *Proceedings of the 12th Conference on Hot Topics in Operating Systems (HotOS'09)*. USENIX Association, Berkeley, CA, 10.

Bo Li, Jianxin Li, Jinpeng Huai, Tianyu Wo, Qin Li, and Liang Zhong. 2009. EnaCloud: An energy-saving application live placement approach for cloud computing environments. In *Proceedings of the 2009 IEEE International Conference on Cloud Computing (CLOUD'09)*. IEEE Computer Society, Washington, DC, 17–24. DOI:http://dx.doi.org/10.1109/CLOUD.2009.72

LINBIT. 2011. Distributed Replicated Block Device. Retrieved from http://www.drbd.org/.

Haikun Liu, Hai Jin, Xiaofei Liao, Liting Hu, and Chen Yu. 2009. 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'09)*. ACM, New York, NY, 101–110. DOI:http://dx.doi.org/10.1145/1551609.1551630

Yingwei Luo, Binbin Zhang, Xiaolin Wang, Zhenlin Wang, Yifeng Sun, and Haogang Chen. 2008. Live and incremental whole-system migration of virtual machines using block-bitmap. In *Proceedings of the IEEE International Conference on Cluster Computing*. 99–106. DOI:http://dx.doi.org/10.1109/CLUSTR.2008.4663760

Kernel Based Virtual Machine. 2012. Main Page. Retrieved from http://www.linux-kvm.org/page/Main_Page.

Sean R. Marston, Zhi Li, Subhajyoti Bandyopadhyay, Anand Ghalsasi, and Juheng Zhang. 2009. Cloud computing: The business perspective. *SSRN eLibrary* (2009).

Dejan S. Milojičić, Fred Douglis, Yves Paindaveine, Richard Wheeler, and Songnian Zhou. 2000. Process migration. *ACM Comput. Surv.* 32, 3 (2000), 241–299. DOI:http://dx.doi.org/10.1145/367701.367728

Andrey Mirkin, Alexey Kuznetsov, and Kir Kolyshkin. 2008. Containers checkpointing and live migration. In *Proceedings of the Linux Symposium 2008*. 85–92.

Sape J. Mullender, Guido van Rossum, Andrew S. Tanenbaum, Robbert van Renesse, and Hans van Staveren. 1990. Amoeba: A distributed operating system for the 1990s. *Computer* 23, 5 (May 1990), 44–53. DOI:http://dx.doi.org/10.1109/2.53354

Ripal Nathuji and Karsten Schwan. 2007. VirtualPower: Coordinated power management in virtualized enterprise systems. *SIGOPS Oper. Syst. Rev.* 41, 6 (Oct. 2007), 265–278. DOI:http://dx.doi.org/10.1145/1323293.1294287

Michael Nelson, Beng-Hong Lim, and Greg Hutchins. 2005. Fast transparent migration for virtual machines. In *Proceedings of the Annual Conference on USENIX Annual Technical Conference (ATEC'05)*. USENIX Association, Berkeley, CA, USA, 25–25.

National Energy Research Scientific Computer Center NERSC. 2011. NERSC Mission. Retrieved from http://www.nersc.gov/about/nersc-mission/.

Jason Nieh. 2005. AutoPod: Unscheduled system updates with zero data loss. In *Proceedings of the 2nd International Conference on Automatic Computing*. IEEE Computer Society, Washington, DC, 367–368. DOI:http://dx.doi.org/10.1109/ICAC.2005.16

Steven Osman, Dinesh Subhraveti, Gong Su, and Jason Nieh. 2002. The design and implementation of Zap: a system for migrating computing environments. *SIGOPS Oper. Syst. Rev.* 36, SI (December 2002), 361–376. DOI:http://dx.doi.org/10.1145/844128.844162

P. Padala, X. Zhu, Z. Wang, S. Singhal, and K. Shin. 2007. *Performance Evaluation of Virtualization Technologies for Server Consolidation*. Technical Report. HP Labs Tech.

Future Grid Portal. 2010. Future Grid. Retrieved from https://portal.futuregrid.org/.

Mendel Rosenblum. 2004. The reincarnation of virtual machines. *Queue* 2, 5 (2004), 34–40. DOI:http://dx.doi.org/10.1145/1016998.1017000

Mendel Rosenblum and Tal Garfinkel. 2005. Virtual machine monitors: Current technology and future trends. *Computer* 38, 5 (May 2005), 39–47. DOI:http://dx.doi.org/10.1109/MC.2005.176

M. Rozier, V. Abrossimov, F. Armand, I. Boule, M. Gien, M. Guillemont, F. Herrmann, C. Kaiser, S. Langlois, P. Léonard, and W. Neuhauser. 1991. Overview of the CHORUS Distributed Operating Systems. *Computing Systems* 1 (1991), 39–69.

J. Sahoo, S. Mohapatra, and R. Lath. 2010. Virtualization: A survey on concepts, taxonomy and associated security issues. In *Proceedings of the 2nd International Conference on Computer and Network Technology (ICCNT'10)*. 222 –226. DOI:http://dx.doi.org/10.1109/ICCNT.2010.49

Constantine P. Sapuntzakis, Ramesh Chandra, Ben Pfaff, Jim Chow, Monica S. Lam, and Mendel Rosenblum. 2002. Optimizing the migration of virtual computers. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation*. 377–390.

Mahadev Satyanarayanan, Paramvir Bahl, Ramón Caceres, and Nigel Davies. 2009. The case for VM-based cloudlets in mobile computing. *IEEE Pervasive Computing* 8, 4 (Oct. 2009), 14–23. DOI:http://dx.doi.org/10.1109/MPRV.2009.82

Mahadev Satyanarayanan, B. Gilbert, M. Toups, N. Tolia, D. R. O'Hallaron, Ajay Surie, A. Wolbach, J. Harkes, A. Perrig, D. J. Farber, M. A. Kozuch, C. J. Helfrich, P. Nath, and H. A. Lagar-Cavilla. 2007. Pervasive personal computing in an Internet suspend/resume system. *IEEE Internet Computing* 11, 2 (March-April 2007), 16–25. DOI:http://dx.doi.org/10.1109/MIC.2007.46

M. Satyanarayanan, Michael A. Kozuch, Casey J. Helfrich, and David R. O'Hallaron. 2005. Towards seamless mobility on pervasive hardware. *Pervasive Mob. Comput.* 1, 2 (July 2005), 157–189. DOI:http://dx.doi.org/10.1016/j.pmcj.2005.03.005

Aameek Singh, Madhukar Korupolu, and Dushmanta Mohapatra. 2008. Server-storage virtualization: integration and load balancing in data centers. In *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing (SC'08)*. IEEE Press, Piscataway, NJ, Article 53, 12 pages. http://dl.acm.org/citation.cfm?id=1413370.1413424

Kunwadee Sripanidkulchai, Sambit Sahu, Yaoping Ruan, Anees Shaikh, and Chitra Dorai. 2010. Are clouds ready for large distributed applications? *SIGOPS Oper. Syst. Rev.* 44, 2 (April 2010), 18–23. DOI:http://dx.doi.org/10.1145/1773912.1773918

Ajay Surie, H. Andrés Lagar-Cavilla, Eyal de Lara, and M. Satyanarayanan. 2008. Low-bandwidth VM migration via opportunistic replay. In *Proceedings of the 9th Workshop on Mobile Computing Systems and Applications (HotMobile'08)*. ACM, New York, NY, 74–79. DOI:http://dx.doi.org/10.1145/1411759.1411779

Nanda Susanta and Chiueh Tzi-Cker. 2005. *A Survey on Virtualization Technologies*. Technical Report. Stony Brook University. http://www.ecsl.cs.sunysb.edu/tr/TR179.pdf

Yoshi Tamura. 2008. Kemari: Virtual machine synchronization for fault tolerance using DomT. In *Proceedings of the Xen Summit*.

Andrew S. Tanenbaum. 1995. *Distributed Operating Systems*. Prentice Hall.

Andrew S. Tanenbaum and Maarten Van Steen. 2001. *Distributed Systems: Principles and Paradigms* (1st ed.). Prentice Hall, Upper Saddle River, NJ.

J. Torres, D. Carrera, K. Hogan, R. Gavalda, V. Beltran, and N. Poggi. 2008. Reducing wasted resources to help achieve green data centers. In *Proceedings of the IEEE International Symposium on Parallel and Distributed Processing (IPDPS'08)*. 1–8. DOI:http://dx.doi.org/10.1109/IPDPS.2008.4536219

Franco Travostino. 2006. Seamless live migration of virtual machines over the MAN/WAN. In *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing (SC'06)*. ACM, New York, NY, 290. DOI:http://dx.doi.org/10.1145/1188455.1188758

Stephen J. Vaughan-Nichols. 2006. New approach to virtualization is a lightweight. *Computer* 39, 11 (Nov. 2006), 12–14. DOI:http://dx.doi.org/10.1109/MC.2006.393

Akshat Verma, Puneet Ahuja, and Anindya Neogi. 2008. pMapper: Power and migration cost aware application placement in virtualized systems. In *Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware (Middleware'08)*. Springer-Verlag, New York, NY, 243–264.

VMware. 2012. VMware GSX Server Documentation. Retrieved from http://www.vmware.com/support/pubs/gsx_pubs.html.

Jens-Sönke Vöckler, Gideon Juve, Ewa Deelman, Mats Rynge, and Bruce Berriman. 2011. Experiences using cloud computing for a scientific workflow application. In *Proceedings of the 2nd International Workshop on Scientific Cloud Computing (ScienceCloud'11)*. ACM, New York, NY, 15–24.

Carl A. Waldspurger. 2002. Memory resource management in VMware ESX server. *SIGOPS Oper. Syst. Rev.* 36, SI (December 2002), 181–194. DOI:http://dx.doi.org/10.1145/844128.844146

A. Whitaker, R. S. Cox, M. Shaw, and S. D. Gribble. 2005. Rethinking the design of virtual machine monitors. *Computer* 38, 5 (May 2005), 57–62. DOI:http://dx.doi.org/10.1109/MC.2005.169

Dan Williams, Hani Jamjoom, and Hakim Weatherspoon. 2012. The Xen-Blanket: virtualize once, run everywhere. In *Proceedings of the 7th ACM European Conference on Computer Systems (EuroSys'12)*. ACM, New York, NY, 113–126. DOI:http://dx.doi.org/10.1145/2168836.2168849

Timothy Wood, K. K. Ramakrishnan, Prashant Shenoy, and Jacobus van der Merwe. 2011. CloudNet: Dynamic pooling of cloud resources by live WAN migration of virtual machines. *SIGPLAN Not.* 46, 7 (March 2011), 121–132. DOI: http://dx.doi.org/10.1145/2007477.1952699

Timothy Wood, Prashant Shenoy, Arun Venkataramani, and Mazin Yousif. 2007. Black-box and gray-box strategies for virtual machine migration. In *Proceedings of the 4th USENIX Conference on Networked Systems Design & Implementation (NSDI'07)*. USENIX Association, Berkeley, CA, 17–17.

Received February 2007; revised March 2009; accepted June 2009