This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2016.2567381, IEEE Transactions on Cloud Computing

Optimizing Live Migration of Multiple Virtual Machines

Walter Cerroni, *Member, IEEE,* and Flavio Esposito, *Member, IEEE,*

Abstract—The Cloud computing paradigm is enabling innovative and disruptive services by allowing enterprises to lease computing, storage and network resources from physical infrastructure owners. This shift in infrastructure management responsibility has brought new revenue models and new challenges to Cloud providers. One of those challenges is to efficiently migrate multiple virtual machines (VMs) within the hosting infrastructure with minimum service interruptions. In this paper we first present a live-migration performance testing, captured on a production-level Linux-based virtualization platform, that motivates the need for a better multi-VM migration strategy. We then propose a geometric programming model whose goal is to optimize the bit rate allocation for the live-migration of multiple VMs and minimize the total migration time, defined as a tradeoff cost function between user-perceived downtime and resource utilization time. By solving our geometric program we gained qualitative and quantitative insights on the design of more efficient solutions for multi-VM live migrations. We found that merely few transferring rounds of dirty memory pages are enough to significantly lower the total migration time. We also demonstrated that, under realistic settings, the proposed method converges sharply to an optimal bit rate assignment, making our approach a viable solution for improving current live-migration implementations.

Index Terms—Cloud Computing, Virtualization, Live Migration, Bandwidth Allocation, Geometric Programming.

1 Introduction

Cloud computing is completely reshaping the way users approach to information technology (IT) services, both for business and leisure purposes. The highly distributed nature of such utility paradigm allows for fast, cheap, easy, on-demand, ubiquitous, and transparent access to personal or shared data and computing services [1]. Cloud computing enables users to store data, run applications, develop software tools, and customize entire virtual IT infrastructures without worrying about the investment and maintenance costs of the required hardware resources. On the other hand, Cloud providers can take advantage of massive sharing of their data center infrastructure costs and offer "pay-per-use" IT services at competitive prices. This virtuous cycle of demand and offer has caused the public Cloud computing market to skyrocket in the last few years, with forecasts of continuing revenue growth [2].

Resource virtualization is one of the key technologies for an efficient deployment of Cloud services. Decoupling service instances from the underlying processing, storage and communication hardware allows to flexibly deploy any application on any server within any data center, independently of the specific operating system and platform being used [3]. In particular, the use of virtual machines (VMs) to implement end-user services is now a well-established practice. A VM can be instantiated, cloned, migrated, rolled-back to a previous state without expensive hardware interventions, enabling flexible workload management operations such as server consolidation and multi-tenant isolation [4]. This is particularly useful in a distributed Cloud system, where VMs can be easily moved to a remote location as long as hypervisor compatibility is guaranteed. Live migration is an additional feature that allows to move VMs from one host to another, with minimal disruption to end-user service availability [5].

Most of the (virtual) services hosted today within a VM are based on multi-tier applications [6]. Typical examples include front-end, business logic and back-end tiers of e-commerce services, or clustered MapReduce computing environments [7]. Joint deployment of multiple correlated VMs is also envisioned by the emerging Network Function Virtualization paradigm, which is radically changing the way network operators plan to develop and build future network infrastructures, adopting a Cloud-oriented approach [8]. In general, a Cloud customer should be considered as a tenant running multiple VMs, that could be either strictly or loosely correlated, and exchange significant amount of traffic among each other. Therefore, moving a given tenant's workload often means migrating a group of VMs, as well as the virtual networks used to interconnect them, to guarantee that the communication between the VMs is not best-effort, but has a dedicated channel with reserved bandwidth.

Such scenario opens a number of challenges related to multiple VMs live migration, especially when considering transfers across multiple Clouds, *e.g.*, due to Cloud bursting, where communication resources play a critical role [9]. For example, in high-performance computing or delay-sensitive applications such as online gaming, augmented reality, video streaming or high-frequency trading, using a sub-optimal bit rate on the management channel dedicated for VM migrations may lead to Service Level Agreement (SLA) violations, bad user experience or even financial losses.

- *W. Cerroni is with the Department of Electrical, Electronic, and Information Engineering "G. Marconi," University of Bologna, Bologna, Italy. E-mail: walter.cerroni@unibo.it*
- *F. Esposito is a member of the Advanced Technology Group at Exegy Inc., St. Louis, MO and a Visiting Research Assistant Professor at University of Missouri, Columbia, MO. E-mail: fesposito@exegy.com*

Manuscript received xxxxxxx; revised xxxxxxx.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2016.2567381, IEEE Transactions on Cloud Computing

How many VMs should be migrated? How much bandwidth should be dedicated to each VM transfer and why? What is the amount of performance degradation, service interruption, and resource consumption that a migration can cause? These are only few of the challenges that may arise in managing a seamless (elastic and scalable) Cloud service. Although some existing Cloud products do offer a range of solutions for joint management of multiple VMs running multi-tier applications [10], they do not allow simultaneous VM live migration and, to the best of our knowledge, there are no readily available, state-of-the-art solutions that enable the necessary optimized allocation of migration bandwidth.

Our Contributions: In this paper, we give an answer to the above questions by dissecting the impact of live migration for an enterprise relying on a Cloud service forced to migrate a set of VMs without SLA violations. In particular:

- We characterize the role of VM live migration with a measurement campaign on production-level, Linux-based virtualization environments running the QEMU-KVM hypervisor [11] and being managed through the *libvirt* library [12]. In particular, we measure the available bit rate and the live migration latency under different conditions.
- Leveraging the insight obtained from the measurement campaign, we propose a novel geometric program [13] to optimize multi-VM migration bandwidth allocation, by minimizing a cost function that represents a tradeoff between two conflicting performance metrics: the user-perceived service interruption (downtime) and the Cloud resource consumption due to the VM transfer (pre-copy time). Our approach is able to provide a generalized quantitative time analysis of the memory transfer phase and the resulting service downtime for multiple VM live migration, along with an effective control of the iterative migration algorithm. We then use our model to quantify the tussle between the downtime and the pre-copy time in Section 6.
- We are able to answer both qualitative design questions, such as *live migration should always be used when transferring multiple VMs*, and quantitative questions, such as *very few dirty page transferring rounds are often enough to minimize the total migration time*. We also empirically demonstrate that, under realistic settings, the proposed geometric program converges sharply to an optimal bit rate assignment solution, making it a viable and useful contribution to develop advanced Cloud management tools.

The rest of the paper is organized as follows. After a brief overview of related work in Section 2, we discuss the live VM migration problem and the current existing solutions in Section 3. In Section 4 we describe our live migration measurement campaign obtained on a *libvirt*-managed QEMU-KVM environment. We then introduce our optimization model and tradeoff cost function in Section 5, and discuss numerical results obtained by solving the geometric program in Section 6. Finally, we conclude our work in Section 7.

2 Related Work

Although the idea of virtualized computing dates back to more than 40 years ago [3], the widespread use of VMs to efficiently run multiple commodity operating systems on top of conventional hardware has become a reality only in the last decade [14], [15]. The highly flexible workload management enabled by server virtualization allows to instantiate, clone and migrate VMs without expensive hardware interventions. Since the original idea of performing live-migration to move VMs with minimal service interruption came up [5], [16], several implementations and research efforts have been carried out on the topic [4], [17], [18], [19]. Only a few state-of-the-art solutions, however, deal with the issue of migrating multiple VMs.

To the best of our knowledge, VMFlockMS [20] is the first migration service specifically optimized for cross-Cloud transfer of groups of correlated VMs, executing multi-tier applications. VMFlockMS' focus is on the migration of large VM disk images, by taking advantage of the similarities typically shown within VMs in the same "flock", which can reach ratios as high as 96%. With the help of a distributed, high-throughput data de-duplication algorithm, VMFlockMS limits the volume of data transferred over the network, thus reducing the migration time. In addition, VMFlockMS adopts prioritized data transfers and early VM boot techniques to accelerate application start-up at the destination host. However, differently from our work, this approach is intended mainly for non-live VM migrations and, in the best case, it requires to pause the VM execution for as long as needed to transfer the whole memory snapshot along with the disk image. Furthermore, the optimal allocation of inter-Cloud link bandwidth is not considered.

Ye *et al.* [21] performed an extensive experimental assessment of multiple VMs live-migration, investigating the role of different resource reservation techniques and migration strategies. In particular, the authors show that a proactive allocation of CPU and memory resources on both source and destination hosts helps improving the migration efficiency and avoiding failures, especially if parallel and workload-aware migration strategies are adopted. Kikuchi *et al*. [22] investigate the performance of concurrent live-migrations in both consolidation and dispersion experiments, and then propose and verify a tandem queuing model. Differently from our approach that considers an optimal bit rate allocation, the cited solutions do not consider the significant impact of network resources on live migration performance, and do not provide a formal definition of optimal bandwidth allocation.

Other implementation-based studies have been carried out, under different assumptions and pursuing different objectives, to investigate simultaneous live-migration of VMs. Some of them apply distributed memory page de-duplication strategies to VMs that are co-located in the same host [23] or rack [24], while others aim at understanding the implications of live-migrating virtual clusters [25] or groups of VMs together, as well as the virtual network interconnecting them [26]. Although the cited work offer significant insights on implementation aspects of the state-of-the-art techniques for simultaneous VM migrations, they do not provide a bandwidth optimization framework targeted to

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2016.2567381, IEEE Transactions on Cloud Computing

this purpose.

Among the existing approaches to schedule multiple VM migrations within a data center, the heuristic algorithm proposed by Sarker *et al.* is aimed at minimizing the total migration time and downtime while taking into account computing, memory and mutual bandwidth requirements of the VMs to be migrated [27]. However, the migration bandwidth is assumed constant during the whole transfer, whereas in our model we set the link capacities for each round as variables of a geometric program and capture the effect of memory dirtying rate during the migration phase.

Solutions that optimize bandwidth allocation for VM migrations have been also floated before [28], [29]. In [28], for example, the authors consider a linear optimization problem introducing routing constraints, but they limit the analysis to the single VM case. In addition, they optimize rate allocation without dissecting the iterative pre-copy algorithm, which in our study resulted in a geometric program. In [29], the authors propose instead a heterogeneous bandwidth provisioning scheme, showing the advantages of assigning different bit rate values to different migration phases. Even in this case the authors focus on the single VM, and do not formalize an optimization problem, but rely on bandwidth allocation heuristics. Our objective function captures instead, in the migration of multiple VMs, the tussle between the two dominant components of the total migration time, the pre-copy time and the downtime.

Performance indicators such as total migration time and downtime that are suitable for multiple VM migrations have been defined in previous work [30], [31]. We use these indicators to devise the objective functions in our optimization model, which determines the best set of parameters to control the iterative transfer algorithm used in live migration. Therefore, by learning from our optimization problem and simulation campaign, one could implement a system able to (automatically) allocate migration bandwidth and number of rounds based on the dirty rate history and the application's goal.

3 Virtual Machine Live Migration: Background and Challenges

The migration of VMs from one hosting server to another while they are still running, referred to as *live-migration*, is a feature of high interest to Cloud managers [4]. In fact, the VM does not need to be shut down at the source and restarted at the destination, allowing the maintenance of both the kernel states, and the states of all processes running within the VM. The migration procedure should have minimum impact on the availability of the services provided by the VM, given that the whole VM environment is maintained consistent across the transfer. In particular, the consistency of the following three sets of states must be guaranteed:

- **network**: the network states are to be maintained, in order to avoid dropping the ongoing TCP connections and disrupt the current services;
- **storage**: once migrated, the VM has to synchronize its file system with that used at the source host;
- **memory**: all changes made by applications and guest operating system to the VM's system memory during the migration process must be detected and replicated at the destination host.

The network state consistency requirement is easily satisfied in a local Cloud environment, *i.e.*, when the VM is connected via a virtual bridge to the same physical LAN at both source and destination hosts. In this case, the VM can keep the same IP address and, when its execution is resumed at the destination, a *gratuitous ARP* packet is sufficient to make all switches and neighbors aware of the new VM location.[1] More complex is the case of a VM migration to a remote data center connected within a different network/domain: in principle, the same IP address cannot be reused and all ongoing connections are dropped. However, recent IP mobility solutions can be applied to overcome this issue, such as those based on naming the application and not the interface [32], or those based on the so-called identifier/locator split principle, including the Host Identity Protocol (HIP) [33], the Identifier-Locator Network Protocol (ILNP) [34], and the Locator/ID Separation Protocol (LISP) [35]. All these solutions can be applied to the VM mobility (and multihoming) problem [36], [37], [38]. Another possibility is to extend the VM's LAN segment across multiple Cloud locations by taking advantage of the Virtual Private LAN Service (VPLS) technology provided by the MPLS protocol [39]. Another emerging approach is to adopt the programmable network reconfiguration capabilities offered by Software Defined Networking [40], which allows migrating entire virtual networks from one data center to another [26], or dynamically rerouting external traffic after a VM has been migrated [41].

The file system state consistency requirement during live-migrations within a local Cloud environment is typically ensured by adopting well-established shared storage solutions: the migrating VM is attached to the same file system, available at both source and destination hosts, so that there is no need to copy disk images. However, when migrating a VM to a remote destination host, the storage needs to be synchronized first. This synchronization may require a large data transfer, that in the worst case can be up to the entire VM disk image. An efficient solution for such synchronization consists in: *(i)* executing the bulk storage data transfer before launching the actual VM live migration, *(ii)* recording all write operations happening during the transfer, and *(iii)* applying the changes at the destination host when the VM is being migrated [39], [42]. The use of template disk images and write throttling mechanisms allows to reduce the amount of bulk storage data to be transferred, and the number of changes to be applied.

The most typical live-migration technique for migrating VM memory state is the so called *pre-copy* [5]. As illustrated in Figure 1, the migration starts with an iterative *push phase* while the VM is still running: in the first round, which we call *round* 0, all memory pages allotted to the VM are copied; then, at each following round, the "dirty" memory pages—the pages modified during the previous round—are transferred. The push phase continues until either the

1. Gratuitous ARP requests are Address Resolution Protocol request packets sent to detect duplicate IP addresses within the same network scope, in which the source and destination IP are both set to the IP of the machine issuing the packet, and the destination MAC is the broadcast address `ff:ff:ff:ff:ff:ff`.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2016.2567381, IEEE Transactions on Cloud Computing

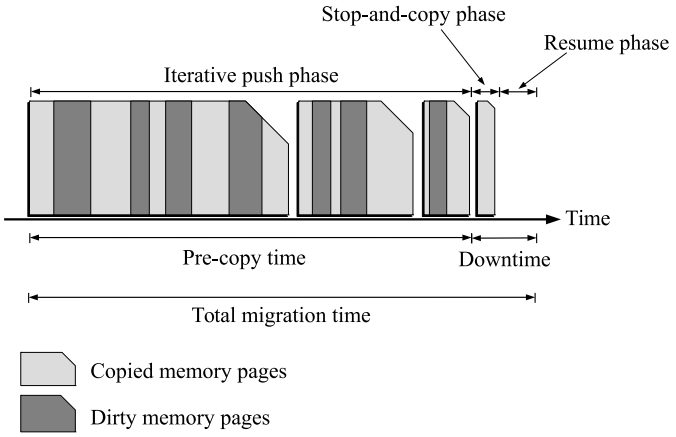


Fig. 1. The three phases of the pre-copy live memory migration technique and related timings (single VM case).

remaining size of dirty memory is small enough to be transmitted in less than a target interval, or the hypervisor concludes that the memory migration process is not converging, *e.g.*, because the page dirtying rate appears to be higher than the transfer rate. At this point, the *stop-and-copy phase* starts: the VM is suspended at the source host and the remaining dirty pages are copied to the destination. Finally, during the *resume phase* the VM is brought back on-line at the destination host with consistent memory, storage and network states.

The opposite approach is adopted by the *post-copy* migration technique [43]. The VM is immediately paused, a minimal processor state is copied, and the VM is resumed at the destination. Then, any memory page needed by the running applications is pulled from the source. In this case, pre-paging and dynamic self-ballooning techniques are used to reduce both the number of page faults, and the amount of data to be fetched. Compared to the pre-copy scheme, a migration by post-copy could reduce the migration time, as each memory page is transferred only once. However, in case of a failure of the destination host, the VM states become unrecoverable, making this approach undesirable.

The most widely adopted virtualization systems, such as VMware, Xen, VirtualBox, and QEMU-KVM, implement live-migrations according to the pre-copy technique, choosing different approaches for maintaining storage state consistency [4]. In this paper, we assume the QEMU-KVM hypervisor as a reference implementation for deriving our live-migration model. Our results are valid however for any general VM live-migration solution based on the pre-copy technique.

Kernel-based Virtual Machine (KVM) implements core virtualization functions at the Linux kernel level, and is able to take advantage of the virtualization extensions supported by modern x86 processors [11]. It requires a user-space hardware emulation platform, such as Quick EMUlator (QEMU), to build a fully-fledged VM hypervisor [44]. QEMU-KVM can provide a full virtualization system as well as paravirtualized drivers to achieve high performance in I/O operations on network devices, disks and memory. Hosting servers based on QEMU-KVM are typically managed through the *libvirt* APIs, an open-source software toolkit that allows to "provision, create, modify, monitor, control, migrate and stop the domains," *i.e.*, the VMs running on multiple hosts [12]. Since *libvirt* supports different kinds of hypervisors, including it in our model does not hinder its generality.

The pre-copy memory migration process implemented by QEMU-KVM is based on an open-loop control mechanism [45]. The administrator is allowed to configure two parameters for any given migration thread: the *maximum transfer rate* and the *maximum allowed downtime*. The maximum transfer rate is useful to limit the amount of network bandwidth consumed by the migration during the iterative push phase so that, when the same physical connection is shared with the applications running in the VM, these do not suffer significant network performance degradation. Such rate limitation is not applied during the final stop-and-copy phase, because the applications are suspended and this last transfer should be performed as fast as possible. The maximum allowed downtime specifies an upper bound to the duration of the stop-and-copy phase, and can be used to control the service interruption time when the VM is paused. An additional control parameter provided by the *libvirt* API is the *migration timeout*, which allows to set a limit to the duration of the whole VM transfer, in case the pre-copy algorithm does not terminate quicky.

When also storage migration is required, QEMU-KVM provides two methods: full disk image replication and incremental replication of changes made to a previously shared template disk image [46]. Both methods use the same migration channel as the memory migration thread, and adopt a sort of pre-copy algorithm to transfer "dirtied" disk blocks during the iterative push phase, trying to not exceed the maximum transfer rate. Network state migration is currently supported by QEMU-KVM only if the VM is migrated within the same LAN, by sending gratuitous ARP packets when the VM is resumed at the destination.

4 Characterizing Live-Migration in Linux QEMU-KVM Environments

In order to define a realistic live migration model, we analyze in details how such task is implemented in the Linux QEMU-KVM hypervisor. We also carry out an experimental characterization of the live-migration procedure, to assess the role of the main system parameters, *i.e.*, available bit rate, memory dirtying rate, and maximum allowed downtime.

The VM live-migration procedure in a libvirt-managed QEMU-KVM environment can be summarized in the following steps:

1) a secure channel (typically using SSH) is established between source and destination hosts;
2) the *maximum transfer rate* and *maximum allowed downtime* parameters, if specified, are passed to the QEMU-KVM hypervisor, and the *migration timeout* counter is started by *libvirt*;
3) a paused instance of the VM to be migrated is created at the destination;
4) if needed, the image disk replication (either full or incremental) begins, while dirty blocks are being monitored;

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2016.2567381, IEEE Transactions on Cloud Computing

5) the iterative memory push phase begins, while dirty pages are being monitored;
6) memory pages and/or disk blocks are continuously sent over the secure channel until the maximum transfer rate is reached;
7) the size of the remaining dirty pages/blocks is used to estimate their total transfer time at the highest possible channel bit rate, *i.e.*, not considering the configured rate limitation;
8) if the estimated remaining transfer time is larger than the maximum allowed downtime *and* the migration timeout has not expired, steps 6 and 7 are iterated;
9) the VM is paused and the remaining dirty pages/blocks are copied to the destination (stop-and-copy phase);
10) the VM is resumed at the destination and destroyed at the source.

To measure the actual behavior of VM live-migration in a libvirt-managed QEMU-KVM environment, we perform some tests on an experimental setup, consisting of two servers running Linux CentOS 6.3 with kernel 2.6.32, QEMU-KVM 0.12.1, and *libvirt* 0.9.10 on a quad-core Intel Core i5-3470 processor, with 4 GB RAM and two Gigabit Ethernet network interfaces. One interface is connected to the management LAN and is used by the hosts for VM migration, whereas the other is connected to the data LAN, bridged with the VM, and used by VM applications to exchange traffic (Figure 2.) Both hosts are attached via the management LAN to a common NFS storage facility, where the VM image disk is saved: this way, no storage synchronization is needed during the migration. The VM to be migrated is configured with one logical CPU and 1 GB of memory. It is running Linux CentOS 6.3 and a simple test application that, after allocating the whole VM memory space, is either idle or periodically rewriting a random set of memory pages at a specified rate. The VM is also executing *iperf* [47] to generate a UDP data traffic stream at 10 Mbps directed to an external receiver. We monitor the bit rates of both migration data transfer and UDP flow. We launch the migration command with a migration timeout of 5 minutes and the default value of the maximum transfer rate set by QEMU-KVM, *i.e.*, 32 MB per second (corresponding to about 270 Mbps).

We execute a first set of experiments with the default value of the maximum allowed downtime, *i.e.* 30 ms, changing the bit rate R available on the management LAN and the VM execution state. The measured bit rates are reported in Figure 3. The first test is performed with full bandwidth available ($R = 1$ Gbps) and an idle VM state, meaning that the only memory pages being dirtied during the migration were those modified by the background operations of the operating system. As shown in Figure 3*a*, the iterative push phase lasts for about 30 seconds and, complying with the configured value, the average migration bit rate is around 270 Mbps. Full bandwidth is used instead in the stop-and-copy phase, which takes about 0.3 second, as reported in the close-up of Figure 3*b*. Here the gap in the UDP stream bit rate gives useful information: due to the negligible network latency in the data LAN, the size of the interval during which the iperf receiver does not detect any packet can be considered as a reasonable measure of the VM downtime, *i.e.*, the time when the VM is suspended. The downtime measured in Figure 3*b* is about 0.5 second, meaning that after the stop-and-copy phase, the resume phase needs additional 0.2 second to bring the VM back to full connectivity.

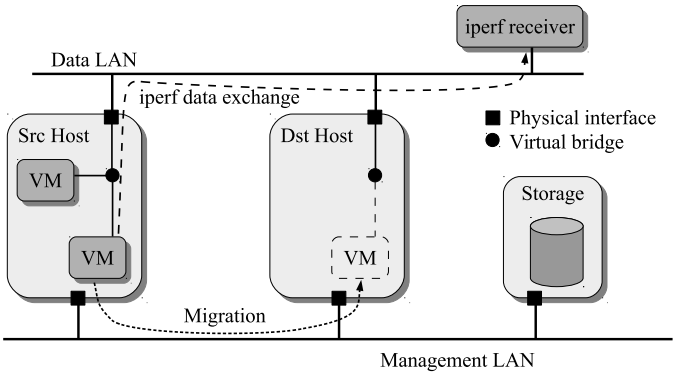


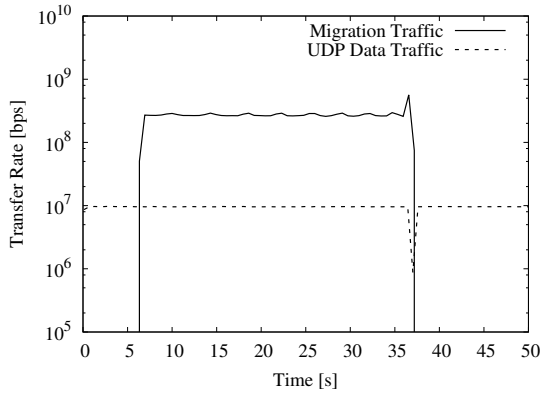
Fig. 2. Experimental setup used for single VM live migration in a *libvirt*-managed QEMU-KVM environment.

The second test is performed again with full bandwidth available, but this time the VM was running the test application at 2500 pages per second (pps), corresponding to 81.92 Mbps. As reported in Figures 3*c* and 3*d*, the effect of such a low memory dirtying rate is a slightly longer push phase (about 35 seconds) and an almost doubled downtime (about 0.9 second). We carry out two additional tests with the maximum bit rate R available on the migration channel limited to 100 Mbps, using the Linux kernel traffic shaping tool. The transfer rates measured when the VM is idle (Figure 3*e*) show that the migration rate limitation results in a significant increase of the overall migration time (about 80 seconds) as well as the downtime (about 21 seconds.) This effect is amplified when the test application is running at 2500 pps (Figure 3*f*.) The previous experiments demonstrate *the key role of memory dirtying rate and migration rate to determine the time performance of live migration*.

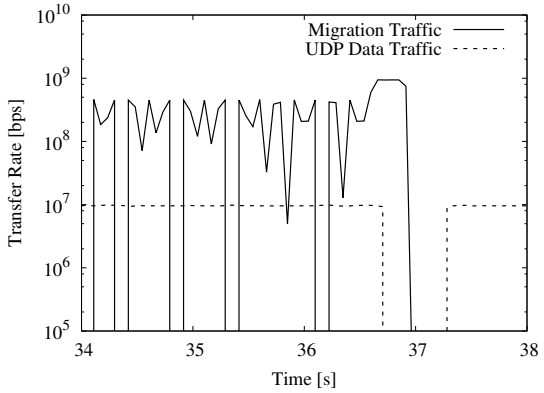
We perform a second set of experiments with different values of the maximum allowed downtime parameter, *i.e.*, 1 ms, 10 ms, 100 ms, 1 s, and 10 s, respectively. The bit rate available for migration is $R = 1$ Gbps and the VM is this time running the test application at 10000 pps. The measured values of the actual downtime, migration time, resume time and average push phase transfer rate are reported for all five tests in Table 1. In all cases the transfer rate is very close to the value of the corresponding control parameter (270 Mbps.) However, when the maximum allowed downtime is set to 1 ms, the dirtying rate is so high that the migration timeout expires before the target downtime is ever reached. After that, *libvirt* suspends the VM at the source host and the remaining dirty pages are finally copied to the destination, resulting in an actual downtime of 3.62 seconds. It is worth noting that the actual VM downtime fails to meet the target value in almost all cases. This is probably caused by a large overestimation of the channel bit rate, which is recomputed by QEMU-KVM at each iteration without using a proper averaging procedure [45].

The reported experiments show how the maximum al-

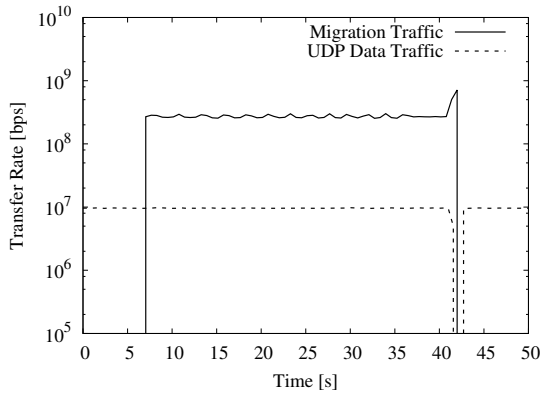
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2016.2567381, IEEE Transactions on Cloud Computing



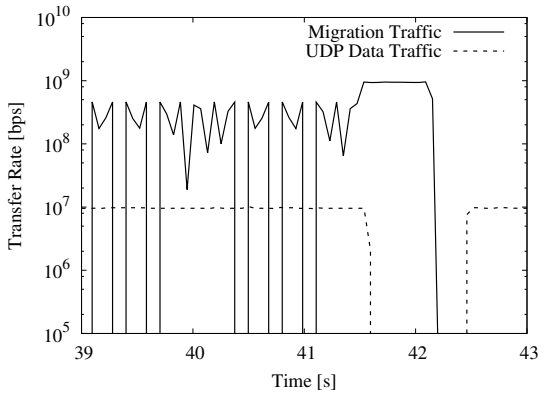
(a) $R = 1$ Gbps, idle VM.



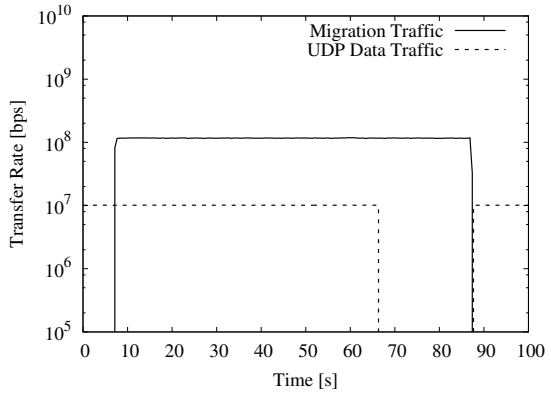
(b) $R = 1$ Gbps, idle VM. Close-up of the stop-and-copy phase in (a).



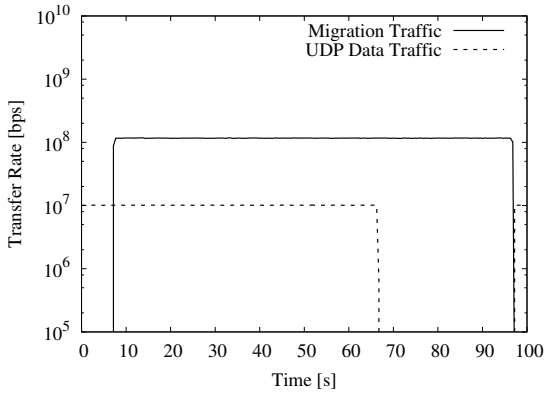
(c) $R = 1$ Gbps, test application writing 2500 pps.



(d) $R = 1$ Gbps, test application writing 2500 pps. Close-up of the stop-and-copy phase in (c).



(e) $R = 100$ Mbps, idle VM.



(f) $R = 100$ Mbps, test application writing 2500 pps.

Fig. 3. Measured transfer rate of migration traffic and UDP data flow generated by the VM, for two values of the bit rate R available for migration and different VM execution states. (a) The full bandwidth is used in the stop-and-copy phase only (note the peak at circa 37 s), with corresponding UDP flow interruption. (b) Close-up of (a) near the stop-and-copy: bit rate throttling in the push phase to avoid maximum transfer rate violations. (c) Prolonged total migration time due to a non-idle VM transfer: a test application is writing 2500 pages per second. (d) Close-up of (c): downtime doubled due to increased dirtying rate. (e) The limited bandwidth available on the migration channel forced the hypervisor to use only $R = 100$ Mbps in both push and stop-and-copy phases, prolonging the total (idle VM) migration time from about 30 s (a) to about 80 s. (f) Both migration time and downtime increased w.r.t. (e) due to the test application writing 2500 pages per second.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2016.2567381, IEEE Transactions on Cloud Computing

TABLE 1
QEMU-KVM migration timings measured in the experiments.

Max downtime	Actual downtime	Migration time	Resume time	Average push phase bit rate
1 ms	3.62 s	303.26 s	0.21 s	272.00 Mbps
10 ms	0.35 s	147.30 s	0.22 s	271.99 Mbps
100 ms	2.35 s	62.70 s	0.14 s	272.03 Mbps
1 s	8.82 s	9.97 s	0.08 s	283.79 Mbps
10 s	8.76 s	9.94 s	0.07 s	286.92 Mbps

lowed downtime parameter impacts the VM migration timing, demonstrating *the opposite trend of the downtime versus the migration time (and, consequently, the pre-copy time.)* The last increase of the target downtime from 1 second to 10 seconds has negligible influence on the migration, meaning that the maximum downtime value of a pure stop-and-copy migration has been reached.[2]

5 Optimal Migration of Multiple VMs

In this section we present our original optimization model for multiple VM migrations. We start with a quantitative analysis of the live migration process, then we introduce the optimal formulation of migration rate allocation, and finally we discuss the geometric programming approach used to solve the problem.

5.1 Optimization Model

As shown by the experiments reported in Section 4, the two key parameters which quantify the performance of VM live-migration are the *downtime* and the *total migration time*. These two quantities tend to have opposite behaviors, and should be carefully balanced. In fact, the downtime measures the impact of the migration on the end-user's perceived quality of service, whereas the total migration time measures the impact on the Cloud network infrastructure: computing resources, as well as transmission resources between them, are consumed during the whole migration phase at both source and destination hosts. The effect of a generic pre-copy algorithm on VM migration timings has already been modeled, starting with the simple case of a single VM [48]. Then the same model has been generalized, extending it to multiple VMs and evaluating how the performance parameters depend on VMs migration scheduling and mutual interactions when providing services to the end-user [30], [31]. In this section we leverage the aforementioned multiple VMs migration performance model to formulate a geometric programming optimization methodology, whose goal is to find a tradeoff between downtime and migration time.

We adopt the following parameter definitions:

- M is the number of VMs to be migrated in a batch;
- $V_{\mathrm{mem},j}$ is the memory size of the j-th VM to be migrated, $\forall j \in \mathcal{J} = \{1, \ldots, M\}$;
- $D_{i,j}$ is the memory dirtying rate during round i in the push phase of the j-th VM;
- $V_{i,j}$ is the amount of dirty memory of VM j copied during round i;
- $T_{i,j}$ is the time needed to transfer $V_{i,j}$;
- n_j is the number of rounds in the push phase of VM j;
- R is the total bit rate of the migration channel;
- $r_{i,j}$ is the channel bit rate reserved in round i to migrate VM j;
- τ_{down} is the VM maximum allowed downtime.

Then the equations that rule the live migration process of VM j are:

$$V_{0,j} = V_{\mathrm{mem},j} = r_{0,j}\, T_{0,j} \tag{1}$$

$$V_{i,j} = D_{i-1,j}\, T_{i-1,j} = r_{i,j}\, T_{i,j} \quad i \in \mathcal{I}_j = \{1, \ldots, n_j\} \tag{2}$$

The last transfer round, *i.e.* the stop-and-copy phase, starts as soon as i reaches the smallest value n_j such that:

$$V_{n_j,j} = D_{n_j-1,j}\, T_{n_j-1,j} \leq r_{n_j,j}\, \tau_{\mathrm{down}} \tag{3}$$

Writing equations (1) and (2) recursively results in:

$$T_{0,j} = \frac{V_{\mathrm{mem},j}}{r_{0,j}} \tag{4}$$

$$T_{i,j} = \frac{V_{i,j}}{r_{i,j}} = \frac{V_{\mathrm{mem},j}}{r_{0,j}} \prod_{h=1}^{i} \frac{D_{h-1,j}}{r_{h,j}} \quad i \in \mathcal{I}_j \tag{5}$$

To simplify the formulation of the optimization model, we fix the size of the problem to be solved, *i.e.*, we assume a fixed number of rounds $\bar{n}$ common to all VM migrations:

$$n_j = \bar{n}, \quad \mathcal{I}_j = \mathcal{I} = \{1, \ldots, \bar{n}\} \qquad \forall j \in \mathcal{J} \tag{6}$$

Depending on the choice of $\bar{n}$, the assumed number of rounds can be sufficient or not to reach the stop condition. This means that, for a given set of input parameters (namely $V_{\mathrm{mem},j}$, $D_{i,j}$, and R), the chosen value of $\bar{n}$ could be different from the smallest one that satisfies inequality (3). Under this assumption we are able to quantify the behavior of the pre-copy algorithm using $\bar{n}$ as a control parameter and understand the role of the number of rounds on live migration performance.

From equations (4) and (5) we can compute two objective functions when $\bar{n}$ rounds are executed in the push phase: the *total pre-copy time*, computed as the sum of the duration of the push phase of each VM:

$$T_{\mathrm{pre}}(\bar{n}, r) = \sum_{j=1}^{M} \frac{V_{\mathrm{mem},j}}{r_{0,j}} \left(1 + \sum_{i=1}^{\bar{n}-1} \prod_{h=1}^{i} \frac{D_{h-1,j}}{r_{h,j}}\right) \tag{7}$$

and the *total downtime*, computed as the sum of the duration of the stop-and-copy phase of each VM:

$$T_{\mathrm{down}}(\bar{n}, r) = \sum_{j=1}^{M} \frac{V_{\mathrm{mem},j}}{r_{0,j}} \prod_{h=1}^{\bar{n}} \frac{D_{h-1,j}}{r_{h,j}} \tag{8}$$

The expression in (7), which increases with the number of rounds $\bar{n}$, quantifies the amount of time required to bring all migrating VMs to the stop-and-copy phase. On the other hand, the second objective function measures the period during which any VM is inactive, so it is suitable to quantify the downtime of the services provided by the migrating VMs. This expression does not include the duration of the resume phase, which typically has a fixed value determined by technology. If the allocated migration bit rate $r_{i,j}$ is

2. The time needed to copy 1 GB plus overhead at 1 Gbps is around 9 seconds.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2016.2567381, IEEE Transactions on Cloud Computing

always greater than the dirtying rate $D_{i-1,j}$, the expression in (8) decreases when $\bar{n}$ increases. Therefore, with these two objective functions we intend to capture the opposite trend of migration time and downtime observed in the experiments reported in section 4.

At each round i of the migration of each VM j, given $V_{\mathrm{mem},j}$ and $D_{i-1,j}$, we wish to allocate the bit rates $r_{i,j}$ so that a combination of the two objective functions

$$T_{\mathrm{mig}}(\bar{n}, r) = C_{\mathrm{pre}} T_{\mathrm{pre}}(\bar{n}, r) + C_{\mathrm{down}} T_{\mathrm{down}}(\bar{n}, r) \quad (9)$$

which we call the *total migration time*, is minimized.[3] Formally, we have the following optimization problem:

$$\min_{r_{i,j}} \quad T_{\mathrm{mig}}(\bar{n}, r) \quad (10a)$$

$$\text{subject to} \quad D_{i-1,j} < r_{i,j} \qquad \forall i \in \mathcal{I} \quad \forall j \in \mathcal{J} \quad (10b)$$

$$\sum_{j=1}^{M} r_{i,j} \leq R \qquad \forall i \in \mathcal{I} \quad (10c)$$

$$D_{i,j} \geq 0 \qquad \forall i \in \mathcal{I} \quad \forall j \in \mathcal{J}$$

$$r_{i,j} > 0 \qquad \forall i \in \mathcal{I} \quad \forall j \in \mathcal{J}$$

The page dirtying rate constraints ($10b$) ensure that the VM migration is sustainable, *i.e.*, during the entire migration process, we can transfer (consume) memory faster than the memory to be transferred is produced. The set of constraints ($10c$) ensure that at each memory transfer round we do not allocate more bit rate than it is at any time available (R).

5.2 Minimization via Geometric Programming

Problem (10) is a *geometric program* [13]. Geometric programs are not convex in their natural form; however, they can be transformed to convex optimization problems by a change of variables, and a transformation of the objective and constraint functions. In particular, in a geometric program the objective is a *posynomial* function, and the constraints are equalities or inequalities between *monomial* functions.

Let $r_1, \ldots, r_n$ be our n real positive bit rate variables,[4] and $r = (r_1, \ldots, r_n)$ the vector of bit rates with components r_i. A real-valued function f of r, with the form:

$$f(r) = c\, r_1^{a_1} r_2^{a_2} \cdots r_n^{a_n}, \quad (11)$$

where $c > 0$, and $a_i \in \mathbb{R}$, is called a *monomial function*, or *monomial*. The definition of monomial used here is standard in geometric programming, but differs from the traditional definition used in algebra, where the exponents must be non-negative integers, *i.e.*, $a_i \in \mathbb{N}$. A sum of monomials, *i.e.*, a function of the form

$$f(r) = \sum_{k=1}^{K} c_k\, r_1^{a_{1k}} r_2^{a_{2k}} \cdots r_n^{a_{nk}}, \quad (12)$$

where $c_k > 0$ and $a_{ik} \in \mathbb{R}$, is called a *posynomial function* with K terms, or simply a *posynomial*.[5] It is easy to see that objective functions (7) and (8) as well as the total migration time (9) are all posynomials.

The main trick to solving a geometric program efficiently is to convert it to a non-linear but convex optimization problem, *i.e.*, a problem with convex objective and inequality constraints, and linear equality constraints. Efficient solution methods for convex optimization problems are well studied and can be found in [49]. The conversion from a geometric program to a convex problem is based on a logarithmic change of variables, and a logarithmic transformation of the objective and constraint functions. In particular, we substitute our original variables r_i with $y_i = \log(r_i)$, and so $r_i = e^{y_i}$. Instead of minimizing the objective $T_{\mathrm{mig}}(\bar{n}, r)$, we then minimize $\log(T_{\mathrm{mig}}(\bar{n}, e^y))$, and we replace the inequality constraints (in a standard form) from $f(r) \leq 1$ to $\log(f(e^y)) \leq 0$.

6 Performance Evaluation

In this section we show several results obtained from our simulation campaign. By solving our geometric program across a wide range of parameter settings, we are able to answer qualitative design questions, such as live-migration should always be used when transferring multiple VMs, and quantitative questions, such as few dirty page transferring rounds are often enough to minimize the total migration time.

Summarizing the details in three main take home messages, we found that, under the simulation settings described in Section 6.1, to minimize the total migration time: (1) the downtime should be as limited as possible, (2) we should always have at least one transferring round, but (3) prolonging the pre-copy time "too much" does not help improving the total migration time.

6.1 Simulation Environment

To validate our model and gain insights on the performance of multi-VM live-migration, we conduct a simulation campaign using the geometric program solver GGPLAB [50]. GGPLAB uses the *primal dual interior point method* [51] to solve the convex version of the original geometric program. All our code is available at [52]. We were able to obtain similar results across a wide range of the parameter space, but we present only a significant representative subset that summarizes our take-home messages. Even though GGPLAB would support optimization problems with larger inputs, when properly configured, our performance evaluation results are limited to a small set of VM migrations. Scalability is not a major concern in our settings since the majority of Cloud applications for enterprises need to simultaneously migrate only small sets of VMs.

With the intent of keeping a general approach, we simulated different distributions of the VM memory size, which is a critical parameter in live migrations: (i) a uniform distribution, to consider complete randomness; (ii) a gaussian distribution, to take into account randomness from a large number of samples around a mean value; (iii) a bimodal distribution, to consider two kinds of VMs, small vs. large, a common case in Cloud computing practices. In some cases, we assumed a constant VM memory size and introduced

3. Note that, in case of a single VM migration ($M = 1$) and choosing $C_{\mathrm{pre}} = C_{\mathrm{down}} = 1$, the expression in (9) gives exactly the time needed to perform the push and stop-and-copy phases, *i.e.*, the actual VM transfer time.

4. To avoid unnecessary complexity with the notation, in this subsection we assume that i is the only decision variable index.

5. Posynomials, which should not be confused with polynomials, are closed under addition, multiplication, and non-negative scaling. Monomials are closed under multiplication and division. If a posynomial is multiplied or divided by a monomial, the result is a posynomial [13].

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2016.2567381, IEEE Transactions on Cloud Computing

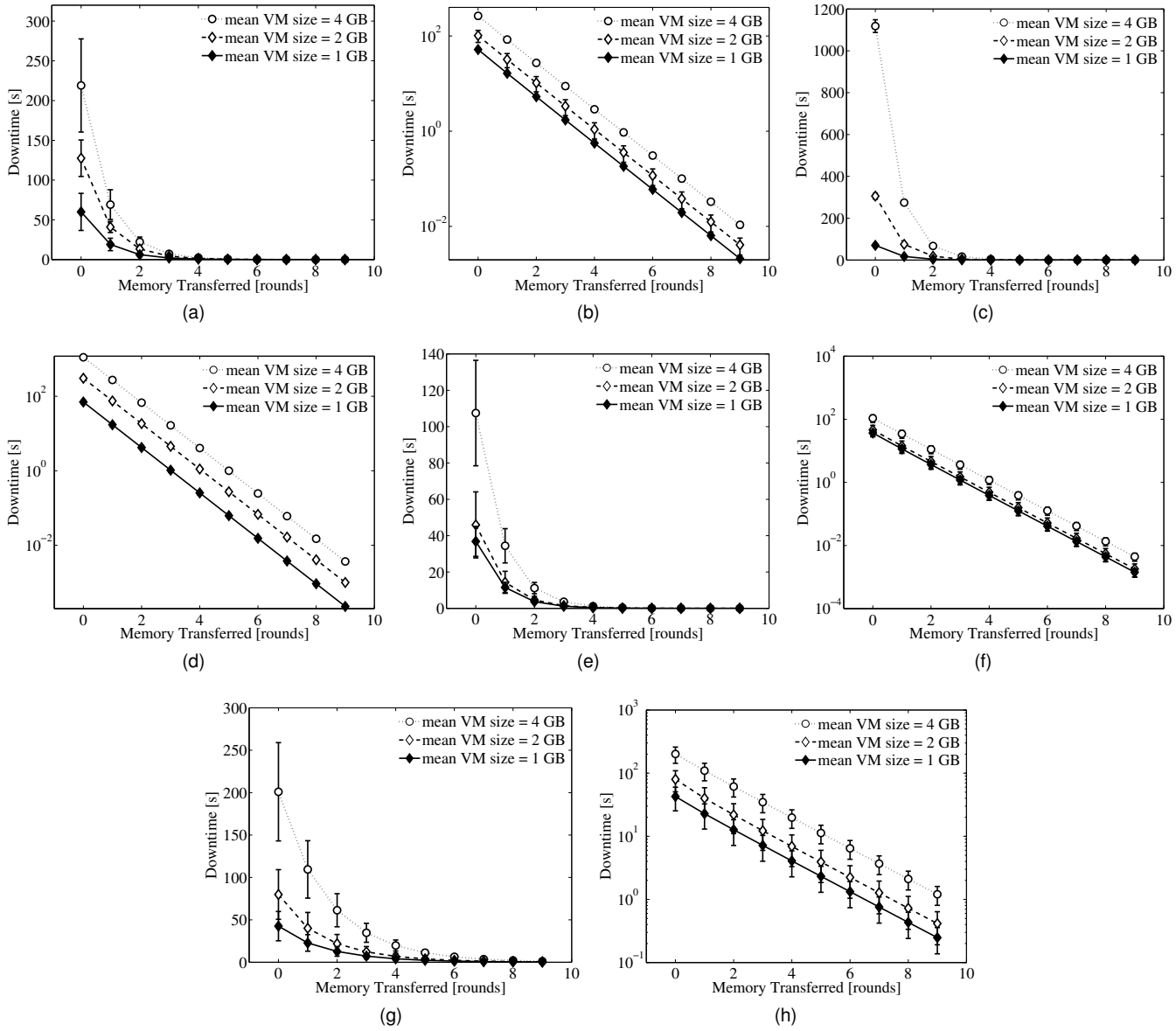


Fig. 4. The downtime significantly decreases when increasing the number of rounds. (a) Impact of downtime only for multiple (3) VMs. The maximum rate was set to 1 Gbps, and the VM size generated from a gaussian distribution with mean value reported in the legend and standard deviation of 300 MB. The values of minimum migration time are obtained solving Problem (10) when $C_{\rm pre} = 0$ and $C_{\rm down} = 1$. (b) Same data as in (a) with semi-logarithmic scale. (c) Same data as in (a) but with memory size generated following a bimodal distribution where 20% of the time the VM has size 10 times bigger than what expressed in the legend. (d) Same data as in (c) but with semi-logarithmic scale. (e) Same data but the VM size is generated according to a uniform distribution with average reported in the legend and standard deviation 300 MB. (f) Same data as in (e) with semi-logarithmic scale. (g) Same as (e) with 6 VMs instead of 3. (h) Same data as in (g) with semi-logarithmic scale.

randomness in the migration process by sampling the page dirtying rate from a gaussian distribution.

We run our simulations on a Linux machine with an Intel i3 CPU at 1.4 GHz and 4 GB of RAM. Unless otherwise specified, our results were obtained with 95% confidence intervals.

6.2 Simulation Results

(1) The downtime can be reduced up to two order of magnitudes while increasing the number of transferring rounds. In Figure $4a$ we show the impact of the downtime, when migrating simultaneously 3 VMs. The maximum rate was set to 1 Gbps, and the values of minimum migration time are obtained solving Problem (10) when $C_{\rm pre} = 0$ and $C_{\rm down} = 1$. The size of the VMs is chosen from a gaussian distribution with mean given in the legend and standard deviation 300 MB. In Figure $4b$, we plot the same data as in (a) with semi-logarithmic scale, to clarify that the downtime may diminish of two orders of magnitude, as we increase the number of rounds. In Figure $4c$ we have have generated the VM sizes sampling from a bimodal distribution where 20% of the time the VM has size 10 times bigger than what expressed in the legend, and in Figure $4d$ the same data is expressed with semi-logarithmic scale. In Figure $4e$, we have instead generated the VM sizes sampling from a uniform distribution with average reported in the legend, and standard deviation of 300 MB, whereas in Figure $4f$ we report the same data plotted in $4e$ with a semi-logarithmic

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2016.2567381, IEEE Transactions on Cloud Computing

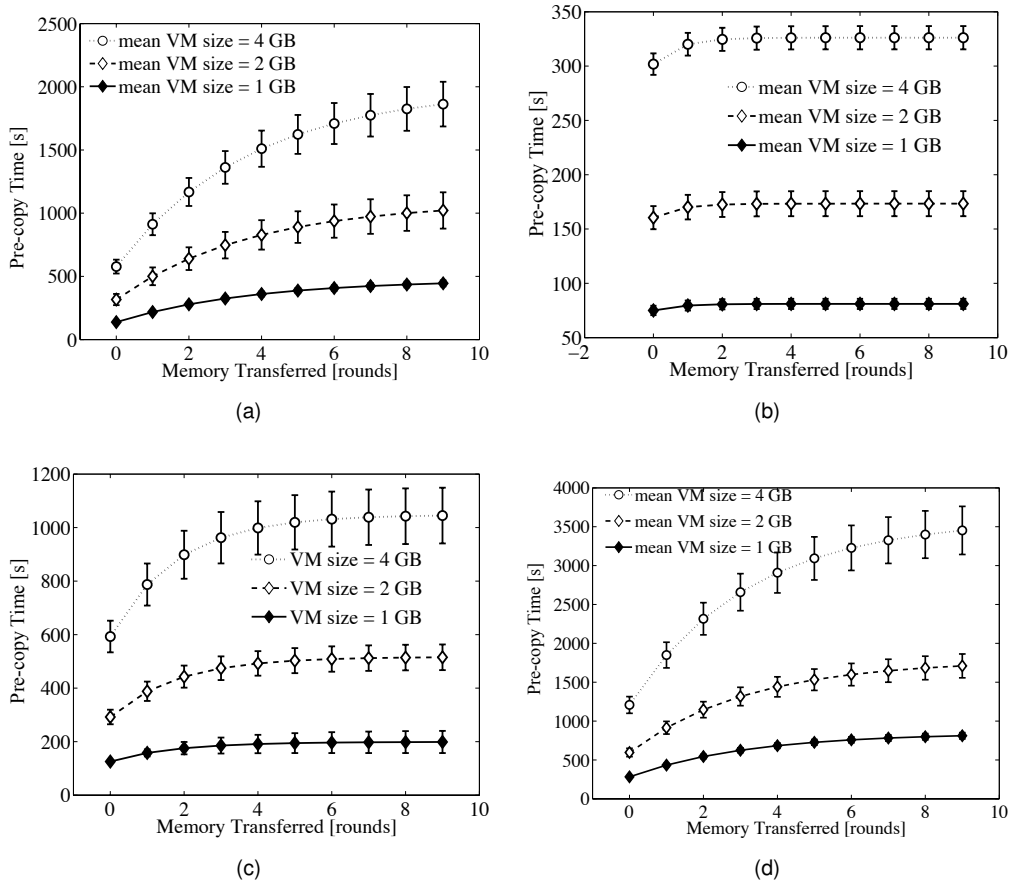


Fig. 5. The pre-copy time reaches a saturation point after a few dirty page transferring rounds. (a) Impact of pre-copy time during the live migration of 3 VMs. The maximum rate was set to $R = 0.5$ Gbps, and the VM size is chosen from a gaussian distribution with mean as shown in the legend and standard deviation 300 MB. The values of minimum migration time are obtained solving Problem (10) when $C_{\text{pre}} = 1$ and $C_{\text{down}} = 0$. (b) Same data as in (a) but with a maximum rate $R = 1$ Gbps to divide among the 3 VMs. (c) The size of the 3 VMs is fixed as expressed in the legend, but the page dirtying rate D is generated from a gaussian distribution with average 2500 pps, and a standard deviation of 300 pps. (d) Same data as in (c) but with 6 VMs instead of 3.

scale. We then analyze the impact of doubling the number of VMs to be migrated (from 3 to 6), still sampling the VM size from a uniform distribution, in both linear and semi-logarithmic scale (Figures $4g$ and $4h$).

Although a diminishing delay is expected when the number of allowed rounds increases, quantifying the down-time decrease is important to gain insights on how to tune or re-architect the QEMU-KVM hypervisor migration functionalities. For example, delay-sensitive applications like online gaming or high-frequency trading may require memory migrations with the maximum possible number of transferring rounds. For bandwidth-sensitive applications instead, *e.g.*, peer-to-peer applications, it may be preferable to reduce the migration bit rate, tolerating a longer service interruption as opposed to a prolonged phase with a lower bit rate.

We have also tested our model with different values of maximum bit rate, as well as different dirtying rates, but we did not find any significant qualitative difference and hence we omit such results.

(2) The total migration time improvement diminishes as we increase the number of transferring rounds. This diminishing effect is a direct consequence of the diminishing duration of each subsequent transferring round. This result gives insights on the importance on allocating enough bandwidth, to guarantee a given quality of service to VMs running memory-intensive applications or with high dirtying rate.

The values of minimum migration time were obtained solving Problem (10) when $C_{\text{pre}} = 1$ and $C_{\text{down}} = 0$. In Figure $5a$, the impact of the pre-copy time only is computed for 3 VMs at a maximum available rate of 0.5 Gbps. The size of the 3 VMs is chosen from a gaussian distribution with mean as shown in the legend and standard deviation 300 MB. In Figure $5b$ we show another experiment with identical parameter set except that we double the maximum available rate R (so that 1 Gbps is divided among the 3 VMs at each round.) As we can see, as the available capacity grows, the gain "saturation point" is reached faster. This result is expected and it is a good sanity check of our simulator [52].

In Figure $5c$ we show a similar experiment but with 3 VMs of constant memory size as expressed in the legend, and we introduce randomness by changing the number of dirty pages per second. We sample the values of the dirty rate D from a gaussian distribution with average 2500 pps, and a standard deviation of 300 pps. Each page is assumed to have the Linux standard size of 4096 Bytes. Finally, in Figure $5d$ we show the effect of doubling the number of VMs to be migrated (6 instead of 3.) Even in this case, we omit similar results with respect to the distributions of VM

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2016.2567381, IEEE Transactions on Cloud Computing

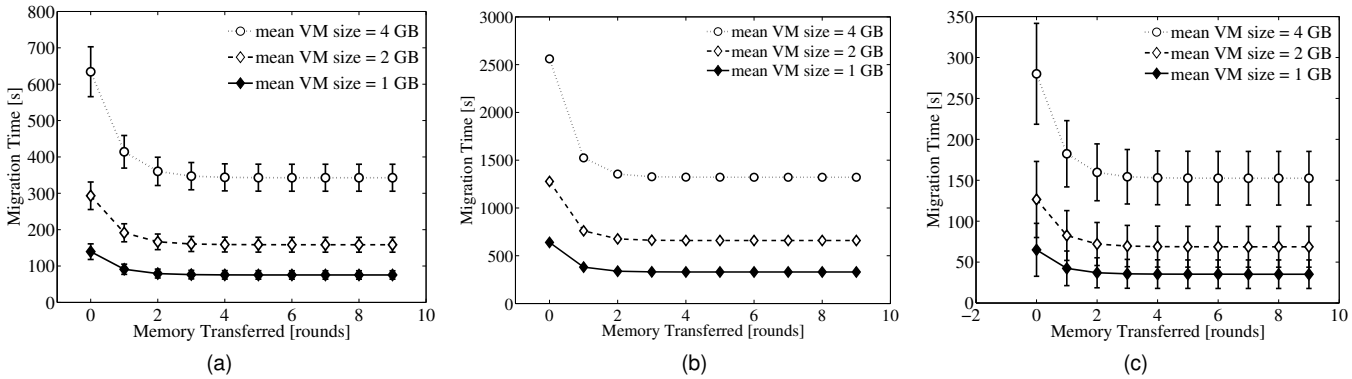


Fig. 6. (a) Impact of total migration time during the live migration of 3 VMs. The maximum rate was set to 1 Gbps, and the VM size is gaussian with average shown in the legend and standard deviation 300 MB. The values of minimum migration time are obtained solving Problem (10) when $C_{\mathrm{pre}} = 1$ and $C_{\mathrm{down}} = 1$. (b) Same data as in (a) but with memory size generated following a bimodal distribution where 20% of the time the VM has size 10 times bigger than what expressed in the legend. (c) The VM size is generated according to a uniform distribution with average reported in the legend and standard deviation 300 MB.

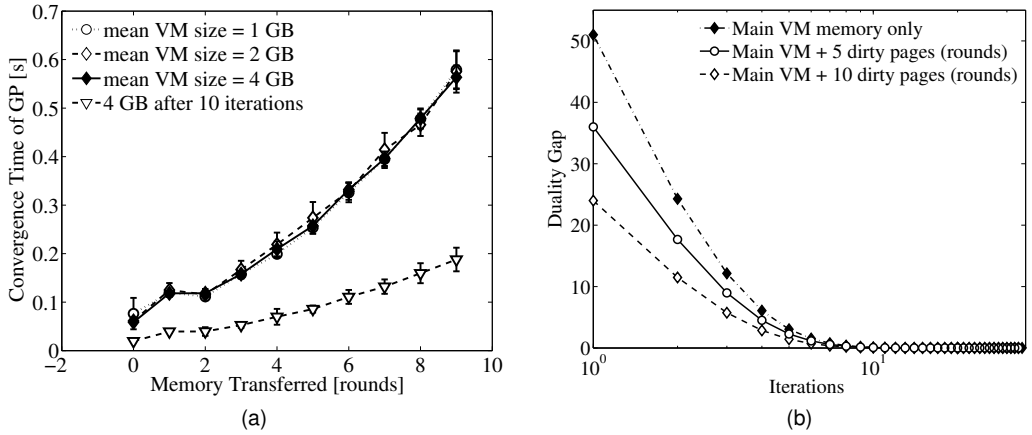


Fig. 7. (a) Convergence time of the geometric program solver: the primal-dual interior point is solved with an iterative method. (b) After only 10 iterations, the primal-dual interior point method that solves our geometric program finds rates very close to the optimal, so in practice, the optimization may stop after few iterations and start the transfers without waiting for the rates to reach their optimal value. The VM size is 1GB, the maximum rate was set to 1 Gbps, the page dirtying rate to 2500 pps, and the number of rounds as reported in the legend. The values indicate the optimality gap for the variables of Problem (10) when $C_{\mathrm{pre}} = C_{\mathrm{down}} = 1$.

sizes, and of dirty pages per second.

(3) Few transferring rounds are enough to minimize the total migration time. In this experiment, we confirm the previous result *(2)* by evaluating the full posynomial function representing the total migration time $T_{\mathrm{mig}}(\bar{n}, r)$, *i.e.*, when both the downtime and the pre-copy time have equal weight $C_{\mathrm{down}} = C_{\mathrm{pre}} = 1$ in the geometric program objective function. Note that, even after the change of variables, our geometric program is convex but still a non-linear optimization problem—the posynomial function $T_{\mathrm{mig}}(\bar{n}, r)$ is a summation of two non-linear functions—so we should not expect to see the total migration time as merely a summation of the two values obtained solving separately the two subproblems with $C_{\mathrm{down}} = 0$ and $C_{\mathrm{pre}} = 0$, respectively.

In this experiment, we show results with 3 VMs chosen from a gaussian distribution with average size indicated by the legend, and standard deviation of 300 MB (Figure 6*a*); with a bimodal distribution, when 20% of the time the VM has size 10 times bigger than what is expressed in the legend (Figure 6*b*); and with VM following a uniform distribution with indicated average and a standard deviation of 300 MB (Figure 6*c*).

(4) Near optimal values of bit rate are reached after only few iterations of the primal-dual interior point algorithm.[6] Our problem seeks a minimization of the total migration time by optimally allocating the bandwidth.[7] However, if we run any optimization problem, practically we have to add to the total migration time also the time to solution, *i.e.*, the convergence time of the iterative algorithm that solves the geometric program. If the convergence time is (too) high, or if it is even comparable with the migration time, it may not make sense to wait for an optimal solution, and a "best effort" bit rate allocation could result into a lower total migration time. With the goal of showing that this is not the case, *i.e.*, the convergence time is low, we report in Figure 7*a* the convergence time measurements when using a machine with standard processing power. The convergence time is

6. Note how the *iterations* of the interior point method refer to the convergence process to an optimal bit rate allocation, and hence are different from the *rounds* that the hypervisor needs to accomplish the live migration.

7. As we assumed a constant page dirtying rate D, bit rates are assigned proportionally to the size of the VM to be transferred.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2016.2567381, IEEE Transactions on Cloud Computing

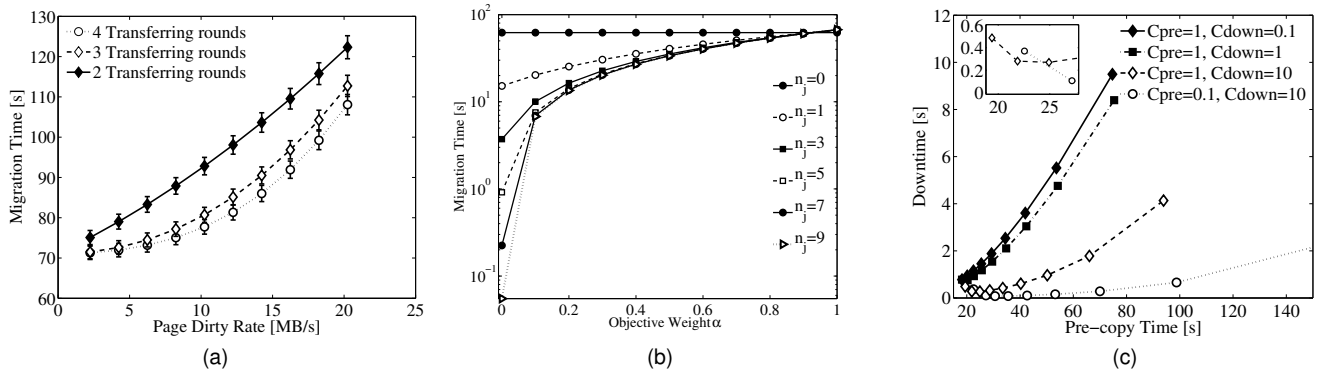


Fig. 8. (a) Impact of dirtying rate on migration time. (b) Impact of the relative weight α in the "tradeoff" objective function $\alpha \cdot T_{\rm pre} + (1-\alpha) T_{\rm down}$. (c) Tradeoff between pre-copy time and downtime for different weights in the cost function (each point represents a given value of $\bar{n} \geq 1$, increasing from left to right).

bounded by roughly 60 ms per round, while the iterative method runs on average for 30 iterations, with a standard deviation of 3 iterations.

As typical of most iterative methods, the first few iterations bring the major improvements to the objective functions. In Figure $7b$ we show that the duality gap converges rapidly to very small values after only 10 iterations.[8] This suggests that an iterative algorithm implementation that would adapt the live migration strategy, *i.e.*, the bit rate to the VM application, will rapidly reach its stopping condition. The lowest curve in Figure $7a$ shows the time to compute the first 10 iterations (approximately a third.) In summary, to transfer a VM of size 4 GB, with 1 Gbps, 200 ms are enough to compute a vector of acceptable (suboptimal) bit rates.

(5) As the page dirtying rate increases, it is convenient to increase also the number of transferring rounds. If we fix the page dirtying rate above a given value (*e.g.*, 5 MB/s in Figure $8a$), we notice how an extended pre-copy phase, *i.e.*, a higher number of transferring rounds, reduces the total migration time. In particular, increasing from 2 to 3 rounds brings a significant improvement, whereas additional rounds have a more limited impact, reaching a saturation effect similar to the one shown in Figure 6. In any case, such an improvement is less pronounced when the dirtying rate becomes very high (*e.g.*, above 20 MB/s in Figure $8a$) due to the power law that relates the total migration time to the dirtying rate.

(6) Live migration lowers the total migration time. To gain additional insights on the impact of the two different components of the total migration time, we solve the geometric program with different weights on the two terms, the pre-copy time and the downtime. We start by setting $C_{\rm pre} = \alpha$ and $C_{\rm down} = (1-\alpha)$, and assessing the impact of the migration time for different transferring rounds $\bar{n}$ as we increase α from 0 to 1 (Figure $8b$). $n_j = \bar{n}$ is a parameter of the figure, while α represents how much weight we assign to the pre-copy phase during the bit rate assignment. Migrating the VMs with a single stop-and-copy phase means setting $\bar{n} = 0$. In this case, the total migration time is equivalent to the pre-copy time, which is equivalent to the downtime. As soon as the number of rounds increases ($\bar{n} > 0$) the total migration time drops. This suggests that *we should not transfer (multiple) VMs with merely a stop-and-copy procedure, i.e., a live-migration is always beneficial for multiple VMs*.

(7) Increasing the number of transferring rounds does not help anymore after the first few rounds. Observing the different curves with $\bar{n} > 0$ in Figure $8b$, we note how increasing the number of rounds leads to a diminishing marginal improvement that depends on the relative weight α. In particular, when the pre-copy time component is more relevant, *i.e.* for higher values of α, saturation is reached with 1 or 2 rounds, whereas when the downtime dominates, 3 or more rounds can still bring some significant benefits. Another interesting observation is that, for a given number of rounds, as we assign more weight to the pre-copy time component of the objective function, the resulting total migration time increases.

The reason of such behavior can be understood observing the actual tradeoff between pre-copy time and downtime, as shown in Figure $8c$ where each point represents a given value of the number of rounds $\bar{n}$, increasing from left to right. When $C_{\rm pre}/C_{\rm down}$ increases, the geometric program tends to find optimal bandwidth values that reduce the pre-copy time and increase the downtime, so the curves are shifted towards the upper-left corner. However, for a given value of $\bar{n}$, the reduction of the pre-copy time is counterbalanced by an increased downtime, with a consequent increase of the total migration time that depends on the weight values. Also, as shown in the inset of Figure $8c$, when $C_{\rm down}$ is dominant and the number of rounds is small, increasing $\bar{n}$ means reducing the amount of dirty memory to be transferred in the stop-and-copy phase, thus reducing the downtime.

In summary, we can argue that even with the generalized objective function definitions given by Equations (7) and (8), which are suitable for multiple VMs migration, *there is a tradeoff between the impact of the migration process on the end-user's perceived quality of service (downtime) and on the resource usage within the Cloud network infrastructure (pre-copy time).*

(8) The optimization gain is maximized for a small number of transferring rounds. As a final result, we quantify the gain of our optimized geometric programming approach:

8. The duality gap is defined as the difference between the optimal value of the primal objective function, and the optimal of the dual problem at any given iteration [49].

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2016.2567381, IEEE Transactions on Cloud Computing

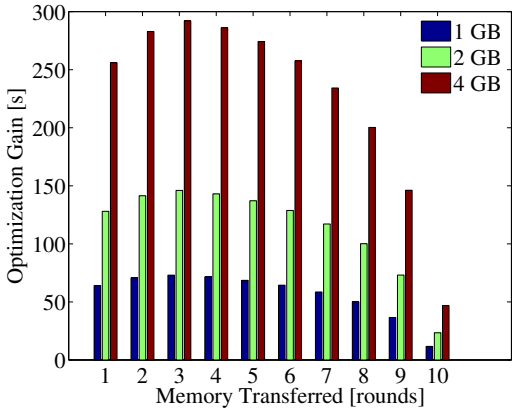


Fig. 9. Optimization gain obtained from our geometric program compared to a non-optimal, fixed migration bit rate allocation, with $C_{\text{pre}} = C_{\text{down}} = 1$, $M = 3$ and $r_{i,j} = R/3 \ \forall i, j$, where $R = 1$ Gbps.

in Figure 9 we show the optimization gain as we increase the number of transferring rounds to migrate 3 VMs with different size. In particular, we compute the total migration time difference with a non-optimal, fixed bit rate allocation equal to $R/3$, where $R = 1$ Gbps. The optimization gain is maximized for a few number of rounds, confirming the previous results. The main reason is that, when the number of rounds becomes too large, both the optimal and the fixed bandwidth allocation reach a saturation point.

7 Conclusion and Open Problems

In this paper we studied the live-migration of multiple virtual machines, a fundamental problem in networked Cloud service management. First, we described the live-migration process and its main challenges, and we reported some experimental results that help identifying the key parameters of the live-migration performance. Then, we used such insights to build a live migration model, and a geometric programming formulation that, when solved, returns the minimum total migration time by optimally allocating the bit rates across the multiple VMs to be migrated.

Our optimization problem aims at simultaneously limiting both the service interruption (downtime), and the time of VM pre-copy, along with a proper control of the iterative memory copying algorithm used in live migration.

By solving the geometric program across a wide range of parameter settings, we were able to answer qualitative design questions, such as live-migration should always be used when transferring multiple VMs, and quantitative questions, such as few dirty page transferring rounds are often enough to minimize the total migration time. We have also shown that, under realistic settings, the proposed geometric program converges sharply to an optimal bit rate assignment, making it a viable and useful solution for improving the current live-migration implementations.

As an interesting open question, we leave the design and implementation, based on widely used open-source virtualization environments such as KVM, of an experimental testbed supporting concurrent live-migration with optimized bit rate allocation in the different phases. Another interesting open question is the exploration of the migration of larger pools of VMs, or even entire virtual networks, for different time-scales and migration requests. One available tool to be used to run optimization experiments with larger inputs is the open-source NEOS Optimization Server [53].

References

[1] R. Buyya, J. Broberg, and A. M. Goscinski, Eds., *Cloud Computing: Principles and Paradigms*. New York: Wiley, 2011.
[2] A. Bartels, J. R. Rymer, and J. Staten, *The Public Cloud Market is Now in Hypergrowth. Sizing the Public Cloud Market, 2014 to 2020*. Forrester Research, 2014.
[3] R. P. Goldberg, "Survey of virtual machine research," *IEEE Computer*, vol. 7, no. 6, pp. 34–45, June 1974.
[4] V. Medina and J. M. García, "A survey of migration mechanisms of virtual machines," *ACM Computing Surveys*, vol. 46, no. 3, pp. 30:1–30:33, January 2014.
[5] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines," in *Proceedings of the 2nd USENIX Symposium on Networked Systems Design & Implementation (NSDI)*, Boston, MA, May 2005.
[6] X. Wang, Z. Du, Y. Chen, and S. Li, "Virtualization-based autonomic resource management for multi-tier web applications in shared data center," *Journal of Systems and Software*, vol. 81, no. 9, pp. 1591–1608, September 2008.
[7] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, January 2008.
[8] "Network Functions Virtualisation (NFV): Network operator perspectives on industry progress," ETSI White Paper, The European Telecommunications Standards Institute, 2013.
[9] F. Cuadrado, A. Navas, J. C. Duenas, and L. M. Vaquero, "Research challenges for cross-cloud applications," in *Proceedings of the IEEE INFOCOM 2014 Workshop on Cross-cloud Systems (CrossCloud '14)*, Toronto, Canada, April 2014.
[10] S. J. Bigelow. (2014, November) How can an enterprise benefit from using a VMware vApp? [Online]. Available: http://searchvmware.techtarget.com/answer/How-can-an-enterprise-benefit-from-using-a-VMware-vApp
[11] [Online]. Available: http://www.linux-kvm.org
[12] [Online]. Available: http://libvirt.org
[13] R. J. Duffin, E. L. Peterson, and C. M. Zener, *Geometric Programming: Theory and Application*. New York: Wiley, 1967.
[14] S. Devine, E. Bugnion, and M. Rosenblum, "Virtualization system including a virtual machine monitor for a computer with a segmented architecture," US Patent 6 397 242, 2002.
[15] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP '03)*, Bolton Landing, NY, October 2003.
[16] M. Nelson, B.-H. Lim, and G. Hutchins, "Fast transparent migration for virtual machines," in *Proceedings of the 2005 USENIX Annual Technical Conference (ATEC'05)*, Anaheim, CA, April 2005.
[17] D. Kapil, E. S. Pilli, and R. C. Joshi, "Live virtual machine migration techniques: Survey and research challenges," in *Proceedings of the 3rd IEEE International Advance Computing Conference (IACC 2013)*, Ghaziabad, India, February 2013.
[18] S. Acharya and D. A. D'Mello, "A taxonomy of live virtual machine (VM) migration mechanisms in cloud computing environment," in *Proceedings of the International Conference on Green Computing, Communication and Conservation of Energy (ICGCE 2013)*, Chennai, India, December 2013.
[19] R. Boutaba, Q. Zhang, and M. F. Zhani, "Virtual machine migration: Benefits, challenges and approaches," in *Communication Infrastructures for Cloud Computing: Design and Applications*, H. Mouftah and B. Kantarci, Eds. IGI-Global, 2013.
[20] S. Al-Kiswany, D. Subhraveti, P. Sarkar, and M. Ripeanu, "VMFlock: Virtual machine co-migration for the cloud," in *Proceedings of the 20th International ACM Symposium on High-Performance Parallel and Distributed Computing (HPDC '11)*, San Jose, CA, June 2011.
[21] K. Ye, X. Jiang, D. Huang, J. Chen, and B. Wang, "Live migration of multiple virtual machines with resource reservation in cloud computing environments," in *Proceedings of the 4th IEEE International Conference on Cloud Computing (CLOUD 2011)*, Washington, DC, July 2011.

[22] S. Kikuchi and Y. Matsumoto, "Performance modeling of concurrent live migration operations in cloud computing systems using PRISM probabilistic model checker," in *Proceedings of the 4th IEEE International Conference on Cloud Computing, (CLOUD 2011)*, Washington, DC, July 2011.

[23] U. Deshpande, X. Wang, and K. Gopalan, "Live gang migration of virtual machines," in *Proceedings of the 20th International ACM Symposium on High Performance Parallel and Distributed Computing (HPDC '11)*, San Jose, CA, June 2011.

[24] U. Deshpande, U. Kulkarni, and K. Gopalan, "Inter-rack live migration of multiple virtual machines," in *Proceedings of the 6th International Workshop on Virtualization Technologies in Distributed Computing (VTDC '12)*, Delft, The Netherlands, June 2012.

[25] K. Ye, X. Jiang, R. Ma, and F. Yan, "VC-migration: Live migration of virtual clusters in the cloud," in *Proceedings of the 13th ACM/IEEE International Conference on Grid Computing (GRID 2012)*, Beijing, China, September 2012.

[26] E. Keller, S. Ghorbani, M. Caesar, and J. Rexford, "Live migration of an entire network (and its hosts)," in *Proceedings of the 11th ACM Workshop on Hot Topics in Networks (HotNets-XI)*, Redmond, WA, October 2012.

[27] T. K. Sarker and M. Tang, "Performance-driven live migration of multiple virtual machines in datacenters," in *Proceedings of the IEEE International Conference on Granular Computing (GrC 2013)*, Beijing, China, December 2013.

[28] U. Mandal, M. Habib, S. Zhang, M. Tornatore, and B. Mukherjee, "Bandwidth and routing assignment for virtual machine migration in photonic cloud networks," in *Proceedings of the 39th European Conference and Exhibition on Optical Communication (ECOC 2013)*, London, UK, September 2013.

[29] U. Mandal, M. Habib, S. Zhang, P. Chowdhury, M. Tornatore, and B. Mukherjee, "Heterogeneous bandwidth provisioning for virtual machine migration over SDN-enabled optical networks," in *Proceedings of the Optical Fiber Communications Conference and Exhibition (OFC 2014)*, San Francisco, CA, March 2014.

[30] F. Callegati and W. Cerroni, "Live migration of virtualized edge networks: Analytical modeling and performance evaluation," in *Proceedings of the IEEE Workshop on Software Defined Networks for Future Networks and Services (SDN4FNS 2013)*, Trento, Italy, November 2013.

[31] W. Cerroni and F. Callegati, "Live migration of virtual network functions in cloud-based edge networks," in *Proceedings of the 2014 IEEE International Conference on Communications (ICC 2014)*, Sydney, Australia, June 2014.

[32] V. Ishakian, J. Akinwumi, F. Esposito, and I. Matta, "On supporting mobility and multihoming in recursive internet architectures," *Journal of Computer Communication*, vol. 35, no. 13, pp. 1561–1573, July 2012.

[33] R. Moskowitz and P. Nikander, *Host Identity Protocol (HIP) Architecture*, IETF Std. RFC 4423, May 2006.

[34] R. J. Atkinson and S. N. Bhatti, *Identifier-Locator Network Protocol (ILNP) Architectural Description*, IETF Std. RFC 6740, November 2012.

[35] D. Farinacci, V. Fuller, D. Meyer, and D. Lewis, *The Locator/ID Separation Protocol (LISP)*, IETF Std. RFC 6830, January 2013.

[36] T. Koponen, A. Gurtov, and P. Nikander, "Application mobility with HIP," in *Proceedings of the 12th International Conference on Telecommunications (ICT 2005)*, Cape Town, South Africa, May 2005.

[37] S. N. Bhatti and R. Atkinson, "Secure and agile wide-area virtual machine mobility," in *Proceedings of the 2012 IEEE Military Communications Conference (MILCOM 2012)*, Orlando, FL, October 2012.

[38] [Online]. Available: http://www.cisco.com/c/en/us/products/switches/nexus-7000-series-switches/white-paper-listing.html

[39] T. Wood, K. K. Ramakrishnan, P. Shenoy, and J. van der Merwe, "CloudNet: Dynamic pooling of cloud resources by live WAN migration of virtual machines," in *Proceedings of the 7th ACM International Conference on Virtual Execution Environments (VEE '11)*, Newport Beach, CA, March 2011.

[40] "Software-defined networking: The new norm for networks," ONF White Paper, The Open Networking Foundation, 2012.

[41] R. Bifulco, M. Brunner, R. Canonico, P. Hasselmeyer, and F. Mir, "Scalability of a mobile cloud management system," in *Proceedings of the 1st ACM Workshop on Mobile Cloud Computing (MCC '12)*, Helsinki, Finland, August 2012.

[42] R. Bradford, E. Kotsovinos, A. Feldmann, and H. Schioberg, "Live wide-area migration of virtual machines including local persistent state," in *Proceedings of the 3rd ACM International Conference on Virtual Execution Environments (VEE '07)*, San Diego, CA, June 2007.

[43] M. R. Hines, U. Deshpande, and K. Gopalan, "Post-copy live migration of virtual machines," *ACM SIGOPS Operating Systems Review*, vol. 43, no. 3, pp. 14–26, July 2009.

[44] [Online]. Available: http://wiki.qemu.org

[45] K. Z. Ibrahim, S. Hofmeyr, C. Iancu, and E. Roman, "Optimized pre-copy live migration for memory intensive applications," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC 2011)*, Seattle, WA, November 2011.

[46] K. Nagin, D. Hadas, Z. Dubitzky, A. Glikson, I. Loy, B. Rochwerger, and L. Schour, "Inter-cloud mobility of virtual machines," in *Proceedings of the 4th International Conference on Systems and Storage (SYSTOR '11)*, Haifa, Israel, May 2011.

[47] [Online]. Available: https://iperf.fr

[48] H. Liu, H. Jin, C.-Z. Xu, and X. Liao, "Performance and energy modeling for live migration of virtual machines," *Cluster Computing*, vol. 16, no. 2, pp. 249–264, June 2013.

[49] S. Boyd and L. Vandenberghe, *Convex Optimization*. New York: Cambridge University Press, 2004.

[50] [Online]. Available: http://stanford.edu/~boyd/ggplab/

[51] S. J. Wright, *Primal-Dual Interior-Point Methods*. Philadelphia: Society for Industrial and Applied Mathematics (SIAM), 1997.

[52] [Online]. Available: https://github.com/flavioesposito/live_migration

[53] [Online]. Available: http://www.neos-server.org/neos/

Walter Cerroni (M'01) obtained the Ph.D. in Electrical and Computer Engineering from the University of Bologna, Italy, in 2003. In 1999 he was a Junior Researcher with Alcatel, Dallas, Texas, USA. From 2003 to 2005 he was a Research Associate at the National Inter-University Consortium for Telecommunications (CNIT), Italy. Since 2005, he is an Assistant Professor of Communication Networks at the University of Bologna, Italy. In 2008 he was a visiting Assistant Professor at the School of Information Sciences, University of Pittsburgh, Pennsylvania, USA. His most recent research interests include: design, implementation and performance evaluation of virtual network function chaining in cloud computing platforms (*e.g.* OpenStack); modeling and design of inter- and intra-data center interconnection networks for cloud computing infrastructures; design of programmable, software-defined hybrid optical network architectures; performance evaluation of dynamic spectrum allocation techniques in flexible optical networks. He has been involved in several research projects, at both national and international levels. He has co-authored about 100 articles published in international journals, magazines and conference proceedings.

Flavio Esposito (M'11) received his Ph.D. in computer science from Boston University in 2013, and his Master of Science in Telecommunication Engineering from University of Florence, Italy in 2005. Flavio is currently a member of the Advanced Technology Group at Exegy, Inc and a Visiting Research Assistant Professor at University of Missouri, Columbia, MO. His research interests include architectures and protocols for (virtual) network management; design, implementation and evaluation of algorithms and protocols for service-based architectures, such as Software Defined Networks (SDN) and Delay-Tolerant Networks (DTN); Flavio worked at Alcatel-Lucent, and he was a research intern at Bell Laboratories, Holmdel, NJ, at Raytheon BBN Technologies, Cambridge, MA, and at EURECOM, France. He was a visiting researcher at MediaTeam, Oulu and at Centre for Wireless Communication, Oulu, Finland. Flavio also worked as Teaching Assistant in graduate courses, such as Computer Networking, as well as undergraduate courses, such as Introduction to Computer Science, and Introduction to Web Design and Internet Technologies. He is a member of both the ACM and the IEEE.