COMMA: Coordinating the Migration of Multi-tier Applications *

Jie Zheng† T. S. Eugene Ng† Kunwadee Sripanidkulchai* Zhaolei Liu†
Rice University† NECTEC, Thailand*

Abstract

Multi-tier applications are widely deployed in today's virtualized cloud computing environments. At the same time, management operations in these virtualized environments, such as load balancing, hardware maintenance, workload consolidation, etc., often make use of live virtual machine (VM) migration to control the placement of VMs. Although existing solutions are able to migrate a single VM efficiently, little attention has been devoted to migrating related VMs in multi-tier applications. Ignoring the relatedness of VMs during migration can lead to serious application performance degradation.

This paper formulates the multi-tier application migration problem, and presents a new communication-impact-driven coordinated approach, as well as a system called COMMA that realizes this approach. Through extensive testbed experiments, numerical analyses, and a demonstration of COMMA on Amazon EC2, we show that this approach is highly effective in minimizing migration's impact on multi-tier applications' performance.

1. Introduction

Server virtualization is a key technology that enables infrastructure-as-a-service cloud computing, which is the fastest growing segment of the cloud computing market and is estimated to reach $9 billion worldwide in 2013 [11]. Optimally managing pools of virtualized resources requires the ability to flexibly map and move running virtual machines (VM) and their data across and within pools [22]. Live migration of VM's disk, memory, and CPU states enables such management capabilities. This paper is a novel study on how to effectively perform live VM migration on multi-tier applications.

Applications that handle the core business and operational data of enterprises are typically multi-tiered. Figure 1 shows Amazon Web Services' [7] referential multi-tier architectures for highly-scalable and reliable web applications. A multi-tier application deployed in the cloud typically includes many interacting VMs running on multiple hosts, such as web server VMs, application server VMs, and database VMs. Such VMs are subjected to be migrated inside a data center or across data centers for management reasons. For instance, due to hardware maintenance, VMs running on physical machines sometimes need to be evacuated. For large corporations, multi-tier applications could be deployed in multiple data centers in different regions. Among the top 1 million domains that use EC2 or Azure to host their services, 44.5% are hosted in two geographical zones, and 22.3% are hosted in three or more geographical zones [13]. For stateful application components, migration could potentially be used in cases where the enterprise needs to re-allocate computing resources over distant data centers or dynamically bring their services' presence into different regions.

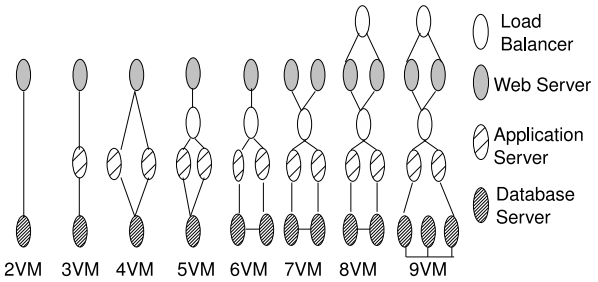


Figure 1. Examples of multi-tier web application architectures. Components that interact are connected by links.

1.1 The components split problem

Because the VMs in a multi-tier application are highly interactive, during migration, the application's performance can severely degrade if the dependent components of an application become split across a high latency and/or congested network path between the source and destination migration sites. Such a slow network path may be encountered within a data center network's aggregation layers, and in networks inter-connecting data centers.

To illustrate this problem, Figure 2 shows an example of migrating a 3-tier e-commerce application across a slow network path. The application has 4 VMs (shown as ovals) implementing a web server, two application servers, and a database server. An edge between two components in the figure indicates that those two components communicate with

* This research was sponsored by the NSF under CNS-1305379, CNS-1018807 and CNS-1162270, by an Alfred P. Sloan Research Fellowship, an IBM Scholarship, an IBM Faculty Award, and by Microsoft Corp.

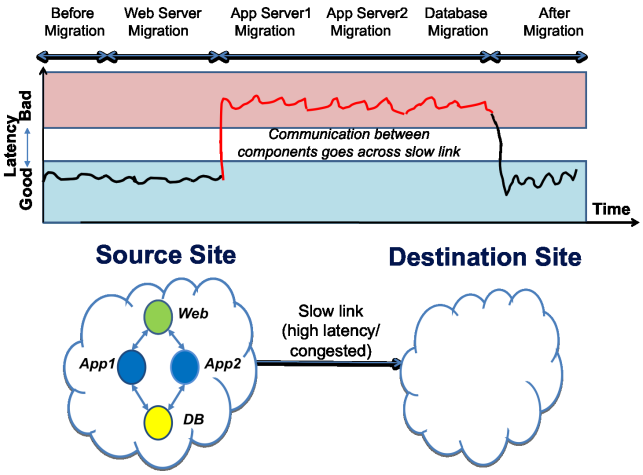


Figure 2. The components split problem in multi-tier application migration.

one another. Let us assume that the four VMs are migrated one by one in the sequence of web server, application server 1, application server 2 and database server. When the web server finishes migration and starts running at the destination site, the communication between the web server and application servers goes across the slow path, resulting in degraded end-to-end request handling latency. When the application servers finish migration, although the communications between the web server and the application servers no longer need to traverse the slow path, it becomes necessary for the application servers to communicate with the database server over the slow path. Only when the database server finally finishes migration will the entire set of VMs be run in the destination site, and the request latency returns to the normal level.

Although existing solutions for migrating an *individual* VM are highly developed [10, 15, 17], when it comes to organizing the migration of a group of related VMs, existing solutions lack sophistication. They either employ *sequential migration*, where VMs are migrated one after another, or *parallel migration*, where VMs are migrated simultaneously. Figure 3 shows that these two migration strategies may result in poor performance when applied to multi-tier applications. Sequential migration results in a potentially long period of performance degradation from when the first VM finishes migration until the last VM finishes migration. Parallel migration is not able to avoid such potential degradation either because the amount of data to migrate for each VM is different and therefore the VMs in general will not finish migration simultaneously. The application will experience performance degradation as long as components are split across a slow path. Furthermore, if the bandwidth required for migrating all VMs in parallel exceeds the actual available bandwidth, additional application performance degradation will be inflicted.

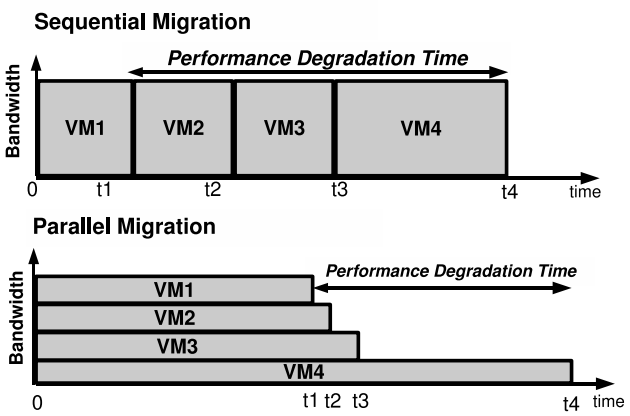


Figure 3. Sequential and parallel migration of a multi-tier application.

1.2 Contributions

This paper makes the following contributions.

1. **Problem formulation (Section 2):** We formulate the multi-tier application migration problem as a performance impact minimization problem, where impact is defined as the volume of communications impacted by components split. It is important to note that simply aiming at simultaneously finish migrating all VMs is not appropriate because bandwidth limitations could render this aim infeasible unless drastic application performance degradation is inflicted. Furthermore, note that defining performance impact in terms of components split duration instead is less desirable because it does not account for the actual impact on application communications.
2. **Communication-impact-driven coordinated approach (Section 3):** We propose a centralized architecture to coordinate the migration of multiple VMs in order to minimize the impact on application communications. The architecture consists of a centralized controller and a local process running inside each VM's hypervisor. Each local process periodically reports data on the VM's migration status and implements the migration settings given by the controller. The controller periodically computes the proper settings for each local process. Thus, this approach is able to adapt to run-time variations in network bandwidth, I/O bandwidth, and application workload.
3. **Algorithm for computing VM migration settings (Section 3):** We propose a novel algorithm which works in two stages. In the first stage, it periodically computes and coordinates the speed settings for migrating the static data of VMs. In the second stage, it coordinates the migration of dynamically generated data. VMs are grouped according to their migration resource requirements to ensure the feasibility of migration. The algorithm then performs *inter-group scheduling* to minimize the impact on

application communications, and performs *intra-group scheduling* to efficiently use network bandwidth for migration.

4. **Extensive evaluation (Sections 4 & 5):** We have fully implemented our approach in a system called COMMA[1]. Through extensive testbed experiments, numerical analyses, and a demonstration of COMMA on Amazon EC2, we show that our approach is highly effective in minimizing migration's impact on multi-tier applications' performance.

2. Problem formulation and challenges

2.1 Background of live migration

Live migration refers to the process of migrating a running VM (**the entire disk, memory, CPU states**) between different physical machines without incurring significant application downtime. Live migration is widely used for planned maintenance, failure avoidance, server consolidation, and load balancing purposes. Live migration also enables a range of new cloud management operations across the wide area such as follow-the-sun provisioning [22]. Thus, live migration happens over a wide range of physical distances, from within a machine rack to across data centers located in different continents.

Full migration of a VM includes migrating (1) the running state of the VM (i.e., CPU state, memory state), (2) the storage or virtual disks used by the VM, and (3) the client-server connections.

Live migration is controlled by the source and destination hypervisors. Live migration has four phases: storage pre-copy, dirty iteration, memory migration and a barely noticeable downtime. During the pre-copy phase, the virtual disk is copied once and all new disk write operations are logged as dirty blocks. During the dirty iteration, the dirty blocks are retransmitted, and new dirty blocks generated during this time are again logged and retransmitted. This dirty block retransmission process repeats until the number of dirty blocks falls below a threshold, and then memory migration begins. The behavior of memory migration is similar to that of storage migration, but the size is much smaller. At the end of memory migration, the VM is suspended. The remaining dirty blocks and pages are copied, and then the VM resumes at the destination.

2.2 Problem formulation

Let n be the number of VMs in a multi-tier application and the set of VMs be $\{vm_1, vm_2, ..., vm_n\}$. The goal is to minimize the performance degradation caused by splitting the communicating components between source and destination sites during the migration. Specifically, we propose a communication-impact-driven approach. To quantify the performance degradation, we define the unit of impact as the volume of traffic between VMs that need to crisscross between the source and destination sites during migration. More concretely, by using the traffic volume to measure the impact, components that communicate more heavily are treated as more important. While many other metrics could be selected to evaluate the impact, e.g. the end-to-end latency of requests, the number of affected requests, performance degradation time, we do not adopt them for the following reasons. We do not adopt the end-to-end latency of requests and the number of affected requests because it is application dependent and requires extra support for measurement at the application level. We do not adopt performance degradation time because it ignores the communication rate between components. We define the communication impact as the volume of traffic which does not require any extra support from the application and is therefore *application-independent*.

Let traffic matrix TM represent the communication traffic rates between VMs prior to the start of migration. Our impact model is based on the traffic prior to migration rather than the traffic during migration. During migration, the traffic rate of the application may be distorted by a variety of factors such as network congestion between the source and destination sites and I/O congestion caused by the data copying activities. Therefore, we cannot optimize against the traffic rate during migration because the actual importance of the interaction between components could be lost through such distortions. Let the migration finish time for vm_i be t_i. Our goal is to minimize the total communication impact of migration, where:

$$impact = \sum_{i=1}^{n} \sum_{j>i}^{n} |t_i - t_j| * TM[i,j] \qquad (1)$$

2.3 Challenges

To tackle the above problem, we first introduce the challenges for managing the migration progress of a single VM. The challenges for managing the migration progress of a single VM have been addressed in our previous work with a system called Pacer [27]. In this paper, we address the new and unique challenges for managing the migration progress of multi-tier applications.

2.3.1 Managing the migration progress of a single VM

Managing the migration progress for a single VM comprises the functions to monitor, predict and control VM migration time. The migration time of VM is difficult to predict and control for the following two reasons:

- **Dynamicity and interference:** VM migration time depends on many static and dynamic factors. For example, the VM image size and memory size are static factors, but the actual workload and available resources (e.g. disk I/O bandwidth and network bandwidth) are dynamic. During the migration, the network traffic and disk I/O from mi-

[1] COMMA stands for COordinated Migration of Multi-tier Applications

gration can interfere with the network traffic and disk I/O from application. That can cause the migration speed and disk dirty rate to change. Because of these dynamic factors, the accurate prediction and control of migration time is challenging.

- **Convergence:** We define the term "*available migration bandwidth*" as the maximal migration speed that migration could achieve considering all bottlenecks such as network and disk I/O bottlenecks. If the available bandwidth is not allocated properly, the migration could fail because the application may generate new data that needs to be migrated at a pace that is faster than the available migration bandwidth. For example, if the available migration bandwidth is 10MBps but the VM generates new data at the speed of 30MBps, migration will not converge in the dirty iteration phase and migration will fail. For a single VM migration, the mechanism to handle non-convergence is either to set a timeout to stop migration and report a failure, or to throttle write operations to reduce the new data generation rate. The latter mechanism will degrade application performance.

For the above challenges of single VM migration, Pacer [27] is able to achieve accurate prediction and control of migration progress. Pacer provides algorithms for predicting dirty set and dirty rate in the pre-copy phase and the dirty iteration phase. These algorithms are leveraged by COMMA to gather information needed for coordination (details in Section 3).

2.3.2 Managing the migration of a multi-tier application

The management of the migration of a multi-tier application becomes even more complicated because of the dependencies between VMs.

1. **Multiple VM migration coordination:** At the level of a single VM, the migration process can be predicted and controlled using Pacer [27]. However, if we rely on an architecture where all VM migration processes act independently, it would be difficult to achieve the joint migration goal for all VMs of the application. It is necessary to design a new architecture where a higher level control mechanism governs and coordinates all VM migration activities for the application.
2. **Convergence in multi-tier application migration:** For multiple VM migrations, the convergence issue mentioned above becomes more complicated but also more interesting. If the network bandwidth is smaller than any single VM's new data generation rate, the only reasonable option is sequential migration. If the network bandwidth is large enough to migrate all VMs together, the problem is easily handled by parallel migration. When the network bandwidth sits in between the previous two cases, we need a mechanism to check whether it is possible to migrate multiple VMs at the same time, to decide how to combine multiple VMs in groups that can be migrated together and to decide how to schedule the migration start and finish time of each group to achieve the goal of minimizing the communication impact.
3. **Dynamicity in multi-tier application migration:** For single VM migration, Pacer [27] can predict the migration time and control the migration progress. However, for multi-tier application migration, the case is more complicated because the VMs are highly interactive and the dynamicity is more unpredictable, since the traffic from multiple VM migrations and the application traffic from all VMs can interfere with each other. In this case, we need a measurement and adaptation mechanism that handles the dynamicity across all VMs.
4. **System efficiency:** The computation complexity for obtaining an optimal solution to coordinate the migration of a multi-tier application could be very high. It is important that the coordination system is efficient and has low overhead. Furthermore, the system should ensure that the available migration bandwidth is utilized efficiently.

Challenge	Solution
Multiple VM migration coordination	Centralized architecture
Convergence in multi-tier application migration	Valid group and inter group scheduling
Dynamicity in multi-tier application migration	Periodic measurement and adaptation
System efficiency	Inter-group scheduling heuristic and intra group scheduling

Table 1. Challenges and solutions in COMMA.

3. System design

3.1 Overview

COMMA is the first migration coordination system for multiple VMs. It relies on a centralized architecture and a two-stage scheduling routine to conduct the coordination. The challenges mentioned in Section 2.3.2 and the corresponding key features that tackle those challenges are summarized in Table 1.

The centralized architecture of COMMA is the key to orchestrating the migration of multiple VMs. The architecture consists of two parts: 1) a centralized controller running on a server which is able to talk to the hypervisors that run the multi-tier application, and 2) a local process running inside each VM's hypervisor. The local process provides three functions: 1) monitor the migration status (such as actual migration speed, migration progress, current dirty blocks and dirty rate) and periodically report to the controller those status to help manage the migration progress; 2) predict the future dirty set and dirty rate to help COMMA estimate the remaining migration time. The dirty set and dirty rate prediction algorithms come from Pacer [27]; 3) a control interface that receives messages from the controller and thus start,

stop or pace the migration speed according to those messages. Based on the reported migration status from all VMs, the controller executes a scheduling algorithm to compute the proper settings for each VM migration process to achieve the performance objective. Given the computation results, the controller sends control messages to the local processes. The control messages include the migration speed and the time when the migration speed should apply, and then each local process would execute the corresponding action. This periodic control and adaptation mechanism with controller coordination overcomes the migration dynamicity and interference problems, and helps to achieve the overall objective of finishing the migration with the minimal impact.

More specifically, COMMA works in two stages. In the first stage, it coordinates the migration speed of the static data of different VMs such that all VMs complete the static data migration at nearly the same time. Before migration, the user provides the list of VMs to be migrated as well as their source hypervisors and destination hypervisors to the controller, and then the controller queries the source hypervisors for each VM's image size and memory size. At the same time, COMMA uses `iperf` [1] to measure the available network bandwidth between the source and destination, and uses `iptraf` [4] to measure the communication traffic matrix of VMs. At the beginning, the measured network bandwidth is considered as the available migration bandwidth. Periodically (every 5 seconds in our implementation), the controller gathers the actual available bandwidth and the migration progress of each VM, and then it paces the migration speed of each VM so that their precopy phases complete at nearly the same time. Subsequently COMMA enters the second stage. COMMA provides mechanisms to check whether it is possible to migrate multiple VMs at the same time, to decide how to combine VM migration in a group to achieve convergence for all VMs in the group called *valid group*, and to decide how to schedule the starting and finishing time of each group to minimize the communication impact called *inter-group scheduling*. Besides, COMMA performs *intra-group scheduling* to schedule each VM inside the same group in order to best maximize the bandwidth utilization.

3.2 Scheduling algorithm

The algorithm works in two stages. In the first stage, it coordinates the migration speed of the static data of VMs (phase 1) so that all VMs complete the precopy phase at nearly the same time. In the second stage, it coordinates the migration of dynamically generated data (phase 2, 3, 4) by inter-group and intra-group scheduling. The definitions of the four phases of migration are in Section 2.1.

Phase 1 migrates static content, and there is no inherent minimum speed requirement. Phase 2 and 3 migrate dynamically generated content. The content generation rate implies a minimum migration speed that must be achieved or otherwise throttling might become necessary (which causes application performance degradation). Therefore, we should dedicate as much of the available bandwidth to phase 2 and 3 in order to prevent application performance degradation. This clearly implies that the phase 1 migration activities should not overlap with phase 2 and 3.

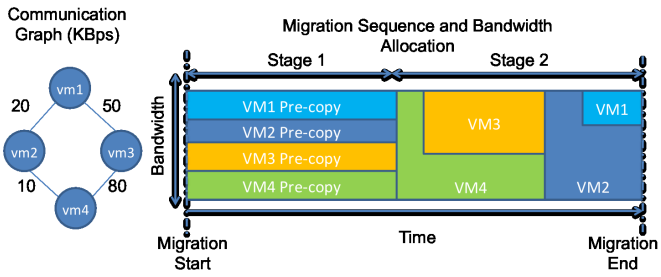


Figure 4. An example of coordinating a multi-tier application migration with COMMA.

3.2.1 First stage

The goal of the first stage is to migrate VMs in parallel and finish all VMs' phase 1 at the same time. Assuming the data copying for each VM is performed over a TCP connection, it is desirable to migrate VMs in parallel because the aggregate transmission throughput achieved by the parallel TCP connections tend to be higher than a single TCP connection.

In this stage, the amount of migrated data is fixed. The controller adjusts each VM's migration speed according to its virtual disk size (see Equation 2).

$$speed_{vm_i} = \frac{\text{DISK\_SIZE}_i * \text{BANDWIDTH}}{\text{TOTAL\_DISK\_SIZE}} \quad (2)$$

During migration, the controller periodically gathers and analyzes the actual available network bandwidth, migration speeds and the progress of VMs. Then it adjusts the migration speed settings of VMs to drive phase 1 migrations to finish at the same time.

Figure 4 shows an example of migrating 4 VMs with COMMA. In the first stage, the controller coordinates the migration of 4 VMs such that their precopy phases complete at the same time. At the end of the first stage, each VM has recorded a set of dirty blocks which require retransmission in the next stage.

3.2.2 Second stage

In the second stage, we introduce the concept of "valid group" to overcome the second challenge mentioned in Section 2.3.2. COMMA performs inter-group scheduling to minimize the communication impact and intra-group scheduling to efficiently use network bandwidth.

To satisfy the convergence constraint, the VMs in the multi-tier application are divided into valid groups according to the following rule: the sum of the VMs' maximal dirty rates in a group is no larger than the available network bandwidth (See Equation 3). The maximal dirty rate is usually achieved at the end of dirty iteration, since at this

time most blocks are clean and they have a high probability of getting dirty again. The maximal dirty rate is needed before the second stage but it is unknown until the migration finishes, and thus we leverage the dirty rate estimation algorithm in Pacer [27] to estimate the maximal dirty rate before the second stage starts. In the second stage, we migrate the VMs in groups based on the inter-group scheduling algorithm. Once a group's migration starts in the second stage, we wait for this group to finish. At the same time, we continue to monitor the actual bandwidth, dirty rate and dirty set for other not-migrated-yet groups. We update the schedule for not-migrated-yet groups by adapting to the actual observed metrics.

$$\sum_{vm_i \in group} \{Max\_dirty\_rate_i\} \leq \text{BANDWIDTH} \quad (3)$$

3.3 Inter-group scheduling

In order to minimize the communication impact, COMMA needs to compute the optimal group combination and migration sequence, which is a hard problem. We propose two algorithms: a brute-force algorithm and a heuristic algorithm. The brute-force algorithm can find the optimal solution but its computation complexity is high. In Section 4, we show that the heuristic algorithm reduces the computation overhead by 99% without losing much in optimality in practice.

3.3.1 Brute-force algorithm

The brute-force algorithm lists all the possible combinations of valid groups, performs the permutation for different migration sequence and computes the communication impact. It records the group combination and migration sequence which generates the minimal impact.

Given a set of VMs, the algorithm generates all subsets first, and each subset will be considered as a group. The algorithm eliminates the invalid groups that do not meet the requirement in Equation 3. It then computes all combinations of valid groups that exactly add up to a complete set of all VMs. Figure 4 shows one such combination of two valid groups that add up to a complete set: $\{vm1, vm2\}$ and $\{vm3, vm4\}$. Next the algorithm permutes each of such combination to get sequences of groups, and those sequences stand for different migration orders. The algorithm then computes the communication impact of each sequence based on the traffic matrix and the migration time reported from the intra-group scheduling algorithm. Finally the algorithm will select the group combination and the sequence with the minimal communication impact.

Let n be the number of VMs in the application. The time complexity for the brute-force algorithm is $O(2^n * n!)$, because it takes $O(2^n)$ to compute all the subsets and takes $O(n!)$ to perform permutation for each combination.

3.3.2 Heuristic algorithm

Our heuristic algorithm tries to estimate the minimal impact by prioritizing VMs that need to communicate with each other the most. Given the traffic matrix, we can get a list L of the communication rates between any two VMs. Each element in L includes $(rate, VM_i, VM_j)$. It represents the communication between node VM_i and node VM_j with $rate$. The heuristic algorithm takes the traffic matrix as input and generates the VM group set S as follows.

- Step 1: Sort the communication rates in L by descending order. S is empty at the beginning.
- Step 2: Repeatedly take the largest rate element $(rate, VM_i, VM_j)$ from L. Check whether VM_i and VM_j are already in S
  - Case 1: Neither VM_i nor VM_j is in S. If the two VMs can be combined into a valid group, insert a new group $\{VM_i, VM_j\}$ into S. Otherwise, insert two groups $\{VM_i\}$ and $\{VM_j\}$ into S.
  - Case 2: Only one VM is in S. For example, VM_i is in S and VM_j is not in S. Find the group which includes VM_i. Check whether VM_j can be merged into the group based on the convergence constraint in Equation 3. If it is still a valid group after merging, then VM_j is merged into the group. Otherwise, a new group $\{VM_j\}$ is inserted into S. For the case that VM_j is in S and VM_i is not, it is similar.
  - Case 3: Both VM_i and VM_j are in S. If the two groups can be merged into one group with convergence constraint, then merge the two groups.
- Step 3: At the end of step 2, we have S which includes the valid group of VMs. The algorithm then compares permutations on the groups to find the one with minimal impact.

The time complexity for the heuristic algorithm is $O(n!)$ because the algorithm is dominated by the last step. Sorting in step 1 takes $O(n^2 logn)$ since there are at most $n(n-1)$ elements in the list L which means every VM communicate with every other VM. Step 2 takes $O(n^2)$. The permutation in step 3 takes $O(n!)$ in the worst case when each VM forms a group.

3.4 Intra-group scheduling

To migrate the VMs in a valid group, one possible solution is to allocate bandwidth equals to the VM's maximal dirty rate to the corresponding VM. Then, we start the migration of all VMs in the group at the same time. The definition of valid group guarantees that we have enough bandwidth to support all VMs in the group migrating concurrently.

However, starting the VMs' migration at the same time is not an efficient use of available migration bandwidth. Figure 5 shows the migration of three VMs in the dirty iteration

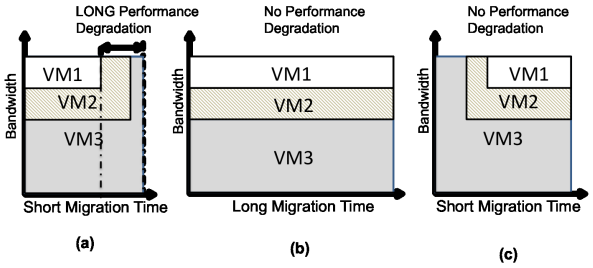


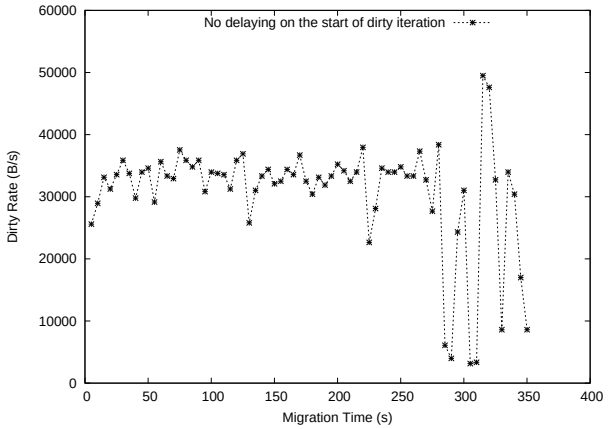
Figure 5. Intra-group scheduling. (a) Start VM migrations at the same time, but finish at different times. Result in long performance degradation time. (b) Start VM migrations at the same time and finish at the same time. Result in long migration time due to the inefficient use of migration bandwidth. (c) Start VM migrations at different times and finish at the same time. No performance degradation and short migration time due to efficient use of migration bandwidth.

with different mechanisms to illustrate this inefficiency. Figure 5(a) shows that 3 VMs start dirty iteration of migration at the same time. Different VMs have different migration speed and dirty rate. Therefore, they finish migration at different times without coordination. For example, VM_1 takes 5 minutes to migrate most of the dirty blocks or pages. Then it could enter phase 4 to pause the VM and switch over to run in the destination. VM_3 may take 10 minutes to finish. That results in 5 minutes of performance degradation. Recall that the goal of COMMA is to reduce the communication impact during migration. Therefore, the ideal case is that the VMs in the group finish migration at the same time. In order to make them finish at the same time, we could force VM_1 and VM_2 to hold in the dirty iteration and continue migrating new generated dirty blocks until VM_3 is done as Figure 5(b) shows. This mechanism is not efficient because it wastes a lot of migration bandwidth in holding VM_1 and VM_2 in the dirty iteration.

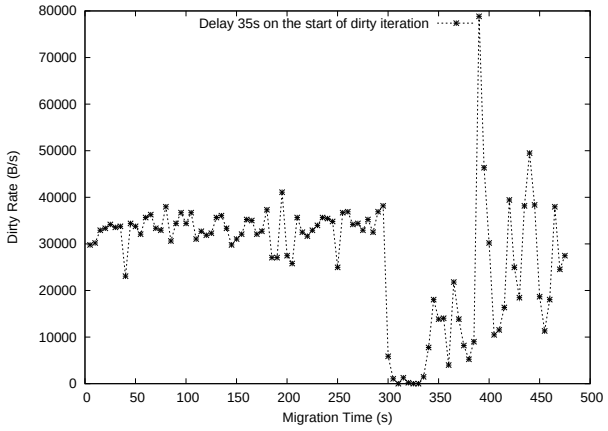
To efficiently use the migration bandwidth, the intra-group scheduling algorithm schedules the migration of VMs inside a group to finish at the same time but it allows them to start the dirty iteration at different times as Figure 5(c) shows.

The design is based on the following observations in practice. (1) Delaying the dirty iteration start time of VMs with light workload can allow for more bandwidth to be allocated to VMs with heavy workload. (2) At the end of the first stage, most of the VM's frequently written blocks are already marked as dirty blocks, and the dirty rate is low at this time. Therefore, delaying the start time of dirty iteration will not significantly increase the number of dirty blocks. (3) Once the dirty iteration starts, it is better to finish migration as soon as possible to save the bandwidth.

While observations (1) and (3) are quite intuitive, observation (2) is less so. To illustrate observation (2), we perform migrations of a file server with 30 clients and analyze



(a) No delay



(b) Delay 35s

Figure 6. An example of delaying the start of dirty iteration for the migration.

its dirty rate. Figure 6(a) shows the migration without any delay for the dirty iteration. From 0 to 280s, migration is in the pre-copy phase and its dirty rate is very stable around 32KBps. Dirty iteration start from 280s to 350s. The dirty rate is very low at the beginning and increases as dirty iteration proceeds. Figure 6(b) show the migration with 35s delay on the start of dirty iteration. During this period, we can see the dirty rate is almost zero. It means there is no more clean blocks getting dirty.

Initially we assume that the minimal required speed for each VM is equal to the VM's maximal dirty rate. We then use the method in Pacer [27] to compute a predicted migration time for each VM. The algorithm would schedule different dirty iteration start times for different VMs according to their predicted migration time so that every VM is expected to finish the migration at the same time.

Available network bandwidth may be larger than the sum of the VMs' minimal required migration speed. If there is extra available bandwidth, the bandwidth will be further allocated to the VMs to minimize the total migration time

of the group. This allocation is done iteratively. Suppose the group has N VMs, the extra available bandwidth is first allocated to vm_N, where the subscript indicates the VM's start time order in the schedule. That is, vm_N is the VM that starts the latest in the schedule. The allocation of this extra bandwidth reduces vm_N's migration time, and thus its start time can be moved closer to the finish time target in the schedule. Next, the extra available bandwidth prior to the start of vm_N is given to vm_{N-1}. vm_{N-1}'s migration time is thus reduced also. Then the extra available bandwidth prior to the start of vm_{N-1} is given to vm_{N-2} and so on, until the migration time for the first VM to start is also minimized.

3.5 Adapting to changing dirty rate and bandwidth

The maximal dirty rate and migration bottleneck are the key input parameters in the scheduling algorithm. When those parameters fluctuates, COMMA is able to handle it with adaptation. COMMA will periodically estimate the maximal dirty rate, measure the available bandwidth and recompute the schedule for not-yet-migrated groups. When COMMA detects that available bandwidth is smaller than the sum of any two VM's maximal dirty rate, the migration will be degraded to sequential migration to ensure convergence. In the extremely rare case, if the available bandwidth is smaller than a single VM's maximal dirty rate, throttling is performed to that VM such that the dirty rate is reduced and migration could converge.

4. Evaluation

4.1 Implementation

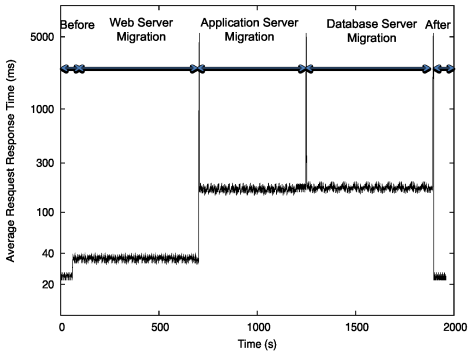
COMMA is implemented on the kernel-based virtual machine (KVM) platform. KVM consists of a loadable kernel module, a processor specific module, and a user-space program – a modified QEMU emulator. COMMA's local process for each VM is implemented on QEMU version 0.12.50, and COMMA's centralized controller is implemented as a light-weight server with C++.

4.2 Experiment setup

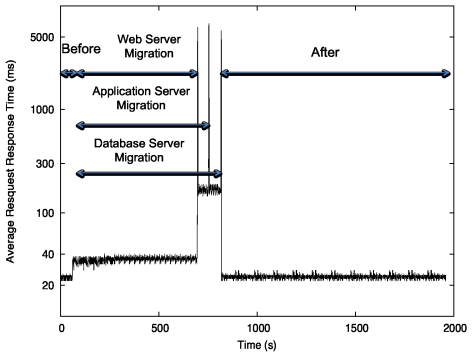
The experiments are set up on six physical machines. Each machine has a 3GHz Quad-core AMD Phenome II X4 945 processor, 8GB RAM, 640GB SATA hard drive, and Ubuntu 9.10 with Linux kernel (with the KVM module) version 2.6.31.

4.3 Application level benefits of COMMA

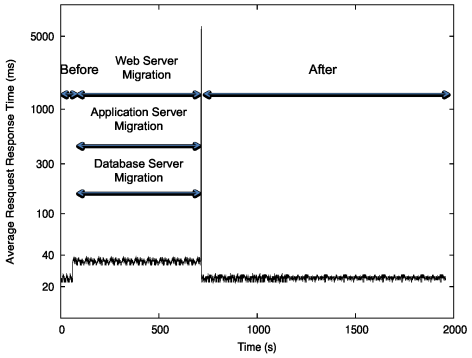
To directly show the benefits of COMMA during the migration of a multi-tier application, we conduct experiments to migrate RUBiS [19], a well-known benchmark for server performance, using sequential migration, parallel migration, and COMMA. RUBiS is a 3-tier application including web server, application server and database server. We measure the application performance by computing the average response time of the request from clients every second. In the



(a) Sequential migration



(b) Parallel migration



(c) COMMA

Figure 7. Application performance during migration of a 3-tier application.

experiment setting, each RUBiS server runs on one VM, and each VM is provisioned on one physical machine. In the experiment, we migrate 3 VMs from 3 source hypervisors to 3 destination hypervisors, with an emulator [3] to emulate a slow link between all the source and destination hypervisors with a round trip latency of 100ms. The architecture of RUBiS is the same as the 3VM setting in Figure 1. Those 3 VMs have the same image size of 8GB. The memory size of the web server, application server, and database server is 2GB, 2GB, and 512MB respectively. The workload is 100 clients. Figure 7 shows the application performance before, during, and after migration, with the different migration approaches.

In sequential migration, the average response time is 20-25ms before migration. Right after migration starts, response time increases to 30-40ms because of the interference from the migration traffic. At the end of web server's migration, there is a response time spike, because the VM is being suspended for a short downtime to finish the final phase of migration. Immediately after that, the web server starts running on the destination hypervisor, and the communication traffic between the web server and the application server goes through the slow link. As a result, the application performance is degraded to 150-190ms. At the end of application server's migration there is also a spike, and then the application server starts running on the destination. Consequently, the communication between the application server and the database server goes through the slow link. This performance degradation lasts for more than 1000 seconds until the database server finishes migration, and then the average response time drops back to 20-25ms. In parallel migration, the degradation time is still high at 82 seconds, and there are still three response time spikes because the three VMs finish migration at different times. Finally, we conduct the migration with COMMA. There is only one response time spike, because all the three VM finishes at nearly the same time (within a period of 1s), and thus the performance degradation time is only 1s.

4.4 COMMA's ability to minimize the migration impact

In this experiment, we will show that COMMA is able to minimize the communication impact, which is defined in equation 1 of section 2.2. For the experiment setting, we add one more application server to the above RUBiS [19] setup. The purpose is to deploy the 4 VMs on at most 3 physical machines with different placements to mimic the unknown VM placement policy in public clouds. The architecture is the same as the 4VM setting in Figure 1. The number of clients is 300.

Table 2 shows that sequential migration has the longest migration time and the highest impact in all cases. More than 2GB of data are affected by sequential migration. Parallel migration reduces the impact to less than 1GB, but this is still much higher than the impact of COMMA. COMMA has up to 475 times of reduction on the amount of data affected by migration.

As the result shows, COMMA has a slightly larger migration time than parallel migration. The reason is that COMMA tries to make all VMs finish migration at the same time, but parallel migration does not. When some VMs finish earlier, the other undergoing VMs that share the same resources can take advantages of the released resource and finish migration earlier.

4.5 Importance of the dynamic adaptation mechanism

While the above experiment shows the high communication impact for sequential and parallel migration, one could come up with alternative migration approaches to reduce the communication impact. Some approaches include reordering the migration sequence in sequential migration, or configuring the migration speed based on static migration info such as the VM disk size. However, without the periodic measurement and adaptation mechanism in COMMA, those approaches cannot achieve the goal of minimizing the communication impact, because they cannot handle the dynamicity during migration.

VM Placement	Sequential Migration		Parallel Migration		COMMA Migration	
	Migr. Time (s)	Impact (MB)	Migr. Time (s)	Impact (MB)	Migr. Time (s)	Impact (MB)
{web,app1,app2,db}	2289	2267	2155	13	2188	7
{web,db},{app1,app2}	2479	2620	918	72	1043	2
{web,app1},{db,app2}	2425	2617	1131	304	1336	2
{web}{app1,app2}{db}	2330	2273	914	950	926	2
{web,app1}{app2}{db}	2213	1920	797	717	988	4
{web}{app1}{app2,db}	2310	2151	1012	259	1244	5

Table 2. Comparisons of three approaches for migrating a 3-tier application. {...} represents set of VMs placed on one physical machine.

The experiment is based on SPECweb2005, which is another popular web service benchmark [2]. It contains a frontend Apache server with an image size of 8GB and a backend database server with an image size of 16GB. The workload is 50 clients. Table 3 shows the results of six migration approaches. The first two approaches are sequential migration with different orders. The sequential migration approach causes a large impact of 265MB and 139MB for the two different migration orders.

The next three approaches are parallel migration with different upper speed limits. In the first experiment, both VMs are configured with the same migration speed limit of 32MBps. They do not finish at the same time, with the impact of 116MB. In the second experiment, the migration speed limit for the frontend VM (8GB) is set to be 16MBps, and for the backend VM (16GB) the speed limit is 32MBps. By setting the migration speed limit proportional to the image size, the user may expect the two VMs to finish migration at the same time. However, this does not happen because the migration cannot achieve the configured speed limits most of the time due to an I/O bottleneck of 15MBps. To avoid this bottleneck, a compromise solution is to decrease the configured speed limits. In the third parallel migration experiment, the configured speed limits are 5MBps and 10MBps. The degradation time is decreased but is still 36s, and the impact is 9MB. However, the low migration speed brings the side effect of longer migration time. These three experiments show that it is impossible for users to purposefully configure the migration speed to achieve low communication impact and timely migration at the same time. In a real cloud environment, guessing the proper speed configuration will be even harder with the additional compet-

	Sequential Migr.		Parallel Migr.			COMMA
	frontend first	backend first	32/32 MBps	16/32 MBps	5/10 MBps	
Impact(MB)	265	139	116	122	9	0.2
Migr. Time(s)	1584	1583	1045	1043	1697	1043

Table 3. Manually tuned sequential and parallel migration vs. COMMA's fully automated approach.

Component Type	Image Size	Mem Size	Dirty Set	Max Dirty Rate
Web/App Server Load Balancer	8GB	1GB	100MB	2MBps
Database	8GB	1GB	1GB	15MBps

Table 4. Example VM and workload parameters for numerical analyses.

ing traffic or more complicated dynamics. With COMMA, the controller can coordinate the migration progress of the two VMs automatically. The two VMs finish migration as quickly as possible and have only a communication impact of 0.2MB.

4.6 Benefits of the heuristic algorithm

In this experiment, we evaluate the communication impact and the computation time for the brute-force inter-group scheduling algorithm and the heuristic inter-group scheduling algorithm. We perform numerical analyses to evaluate the different migration approaches on the different multi-tier web service architectures shown in Figure 1.

Assume that the VMs have the characteristics in Table 4, and the available migration bandwidth is 256Mbps which is shared by all VM's migrations. The parameters that we select are image size, memory size, dirty set size and max dirty rate. These are the four key parameters for determining the migration time using the method in [27]. We select a set of representative configurations to enable our numerical analyses. The image size and memory size follow the recommendation from the VMware VMmark benchmark configuration [20]. Dirty set is defined as the written and not-yet-migrated data bytes on the VM's virtual disk at the end of disk image pre-copy. Dirty rate is defined as the speed at which the VM's virtual disk and memory is written. Different workloads have different dirty rate and dirty set. We use a higher value for the database server to mimic the intensive disk write operations in typical database servers.

We generate a random number between 0 to 100KBps as the communication rate when there is a link between two VMs, and each experiment is run 3 times with different random number seeds. Table 5 shows the average results. In the first four cases ($VM \leq 5$), all VMs can be coordinated to finish at the same time and the impact is 0. In larger scales of ($VM \geq 6$), the coordination algorithm will perform the best effort to schedule VM's migration and achieve the minimal impact. The coordination with the brute force algorithm achieves a slightly lower impact than the coordination with the heuristic algorithm. Take the migration of 9 VMs for example, comparing to the sequential migration, COMMA with the brute-force algorithm could reduce the impact by 97.9% and COMMA with the heuristic algorithm could reduce the impact by 96.9%.

	Sequential Migration	Parallel Migration	COMMA-Bruteforce Migration	COMMA-Heuristic Migration
2VM	28	3	0	0
3VM	84	3	0	0
4VM	114	3	0	0
5VM	109	3	0	0
6VM	222	INF	1	2
7VM	287	INF	2	2
8VM	288	INF	1	2
9VM	424	INF	9	13

Table 5. Communication impact (MB) with different migration approaches. INF indicates that migration cannot converge.

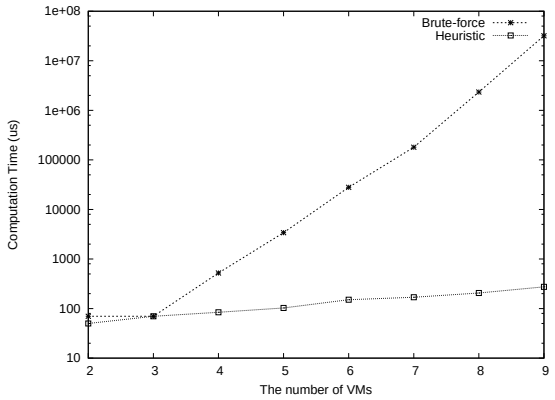


Figure 8. Computation time for brute-force algorithm and heuristic algorithm.

Figure 8 shows the computation time for the brute-force algorithm and the heuristic algorithm. When the number of VM increases to 9, the computation time for the brute-force algorithm rapidly increases to 32 seconds, while the computation time for the heuristic algorithm is a much more reasonable 274us. In other words, the heuristic algorithm reduces the computation overhead up to 99%.

5. EC2 demonstration

To demonstrate COMMA in a real commercial hybrid cloud environment, we conduct an experiment using Amazon EC2 public cloud. The experiment migrates two SPECweb2005 VMs from a university campus network to EC2 instances with the same settings as the experiment in Section 4.5 except that the workload is reduced to 10 clients. Since KVM cannot run on top of EC2 instances, we run QEMU with the "no-kvm" mode, which reduces the application's performance. Reducing to 10 clients ensures the convergence

Figure 9. Live migration of multi-tier applications to EC2.

	Sequential Migr.		Parallel Migr.			Coord.
	Migration		Migration			
	frontend first	backend first	32/32 MBps	16/32 MBps	5/10 MBps	
Impact(MB)	28	17	19	6	6	0.1
Migr. Time(s)	871s	919s	821s	885s	1924s	741s

Table 6. Manually tuned sequential and parallel migration vs. COMMA in EC2 demonstration.

of the dirty iteration and memory migration phases. We use EC2's High-CPU Medium instances running Ubuntu 12.04.

The result is in Table 6. In the sequential approach, the performance degradation time is equal to the time of migrating the second VM, and thus the migration impact could be as high as 28MB and 17MB for the two different migration orders. For the parallel approach with the same migration upper speed limit for both VMs, the degradation impact is still 19 MB, which is not much better than the impact of sequential approach. We next set the migration speed limit proportional to the size of the VM image. In this case, the impact decreases to 6MB, but this approach does not fully utilize the available bandwidth. Consequently, the migration time increases, especially in the last case with the migration speed limits of 5/10 MBps. For COMMA, migration's impact is very tiny, and the migration time is the shortest because it utilizes bandwidth efficiently. COMMA reduces the communication impact by 190 times compared to that of parallel migration. The above results show that COMMA is able to successfully coordinate the migration of multi-tier applications across the wide area with extremely low impact.

6. Related work

To the best of our knowledge, no previous work is directly comparable to COMMA, which is the first paper to address the problem of live migration of multi-tier applications. The goal of COMMA is to reduce the application performance degradation during migration.

There is some related work on performance modeling and measurement of single VM live migration [6, 8, 9, 21, 23, 25]. Wu *et al.* [23] create the performance model with regression methods for migrating a VM running different resource-intensive applications. Breitgand *et al.* [8] quantify the trade-off between minimizing the copy phase duration and maintaining an acceptable quality of service during the pre-copy phase for CPU/memory-only migration. Akoush *et al.* [6] provides two simulation models to predict memory migration time. Voorsluys *et al.* [21] present a performance evaluation on the effects of live migration. Zhao *et al.* [25] provide a model that can characterize the VM migration process and predict its performance, based on a comprehensive experimental analysis. Checconi *et al.* [9] introduce a stochastic model for the migration process and reserves resource shares to individual VMs to meet the strict timing constraints of real-time virtualized applications. Relative to these previous works, not only does COMMA address a different set of problems which targets at multiple VM migrations, COMMA also takes an approach based on real measurements and run-time adaptation, which are found to be crucial to cope with workload and performance interference dynamics, to realize a complete system.

There exists related work on multiple simultaneous migrations [5, 18]. Nicolae *et al.* [18] proposes a hypervisor-transparent approach for efficient live migration of I/O intensive workloads. It relies on a hybrid active push-prioritized prefetch strategy to speed up migration and reduce migration time, which makes it highly resilient to rapid changes of disk state exhibited by I/O intensive workloads. AI-Kiswany [5] employs data deduplication in live migration to reduce the migration traffic. It presents VMFlockMS, a migration service optimized for cross-data center transfer and instantiation of groups of virtual machine images. VMFlockMS is designed to be deployed as a set of virtual appliances which make efficient use of the available cloud resources. The purpose of the system is to locally access and deduplicate the images and data in a distributed fashion with minimal requirements imposed on the cloud API to access the VM image repository. Some other work for live migration focuses on reducing migration traffic by compression [12, 14], deduplication [24] and reordering migrated blocks [16, 26]. The purposes of above related work are either to reduce migration traffic or to reduce migration time which are very different from what this paper focuses on.

7. Conclusions

We have introduced COMMA – the first coordinated live VM migration system for multi-tier applications. We have formulated the multi-tier application migration problem, and presented a new communication-impact-driven coordinated approach, as well as a fully implemented system on KVM that realizes the approach. COMMA is based on a two-stage scheduling algorithm to coordinate the migration of VMs with the goal of minimizing the migration's impact on inter-component communications. From a series of experiments, we have shown the significant benefits of COMMA in reducing the communication impact, while the schedul-

ing algorithm of COMMA incurs little overhead. We believe COMMA will have far reaching impact because it is applicable to numerous intra-data center and inter-data center VM migration scenarios. Furthermore, the techniques underlying COMMA can be easily applied to other virtualization platforms such as VMware, Xen and Hyper-V.

References

[1] iperf. http://sourceforge.net/projects/iperf/.

[2] Specweb2005. `http://www.spec.org/web2005/`.

[3] WANem. http://wanem.sourceforge.net.

[4] iptraf. http://iptraf.seul.org/, 2005.

[5] S. AI-Kiswany, D. Subhraveti, P. Sarkar, and M. Ripeanu. Vmflock: Virtual machine co-migration for the cloud. In *HPDC*, 2011.

[6] S. Akoush, R. Sohan, A. Rice, A. W.Moore, and A. Hopper. Predicting the performance of virtual machine migration. In *IEEE 18th annual international symposium on modeling, analysis and simulation of computer and telecommunication systems*. IEEE, 2010.

[7] Amazon. Aws reference architecture. `http://aws.amazon.com/architecture/`.

[8] D. Breitgand, G. Kutiel, and D. Raz. Cost-aware live migration of services in the cloud. In *USENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services*. USENIX, 2011.

[9] F. Checconi, T. Cucinotta, and M. Stein. Real-time issues in live migration of virtual machines. In *Euro-Par 2009–Parallel Processing Workshops*, pages 454–466. Springer, 2010.

[10] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live migration of virtual machines. In *NSDI'05*, 2005.

[11] Gartner. `http://www.gartner.com/newsroom/id/2352816`, 2013.

[12] S. Hacking and B. Hudzia. Improving the live migration process of large enterprise applications. In *VTDC'09: Proceedings of the 3rd International Workshop on Virtualization Technologies in Distributed Computing*, 2009.

[13] K. He, A. Fisher, L. Wang, A. Gember, A. Akella, and T. Ristenpart. Next stop, the cloud: Understanding modern web service deployment in ec2 and azure. In *IMC*, 2013.

[14] H. Jin, L. Deng, S. Wu, X. Shi, and X. Pan. Live virtual machine migration with adaptive memory compression. In *IEEE International Conference on Cluster Computing*, 2009.

[15] KVM. Kernel based virtual machine. `http://www.linux-kvm.org/page/Main_Page`.

[16] A. Mashtizadeh, E. Celebi, T. Garfinkel, and M. Cai. The design and evolution of live storage migration in vmware esx. In *Proceedings of the annual conference on USENIX Annual Technical Conference*. USENIX Association, 2011.

[17] M. Nelson, B.-H. Lim, and G. Hutchins. Fast transparent migration for virtual machines. In *USENIX'05*, USA, 2005.

[18] B. Nicolae and F. Cappello. Towards efficient live migration of I/O intensive workloads: A transparent storage transfer proposal. In *HPDC*, 2012.

[19] RUBiS. `http://rubis.ow2.org`.

[20] VMWare. VMmark Virtualization Benchmarks. http://www.vmware.com/products/vmmark/, Jan. 2010.

[21] W. Voorsluys, J. Broberg, S. Venugopal, and R. Buyya. Cost of virtual machine live migration in clouds: A performance evaluation, 2009.

[22] T. Wood, P. Shenoy, K.K.Ramakrishnan, and J. V. der Merwe. Cloudnet: Dynamic pooling of cloud resources by live wan migration of virtual machines. In *ACM VEE*, 2011.

[23] Y. Wu and M. Zhao. Performance modeling of virtual machine live migration. In *Proceedings of the 2011 IEEE 4th International Conference on Cloud Computing*. IEEE, 2011.

[24] X. Zhang, Z. Huo, J. Ma, and D. Meng. Exploiting data deduplication to accelerate live virtual machine migration. In *IEEE International Conference on Cluster Computing*, 2010.

[25] M. Zhao and R. J. Figueiredo. Experimental study of virtual machine migration in support of reservation of cluster resources. In *Proceedings of the 2nd international workshop on Virtualization technology in distributed computing*, page 5. ACM, 2007.

[26] J. Zheng, T. S. E. Ng, and K. Sripanidkulchai. Workload-aware live storage migration for clouds. In *ACM VEE*, Apr. 2011.

[27] J. Zheng, T. S. E. Ng, K. Sripanidkulchai, and Z. Liu. Pacer: A progress management system for live virtual machine migration in cloud computing. *IEEE Transactions on Network and Service Management*, 2013, to be appear.