VMbuddies: Coordinating Live Migration of Multi-Tier Applications in Cloud Environments

Haikun Liu and Bingsheng He

Abstract—Enabled by virtualization technologies, various multi-tier applications (such as web applications) are hosted by virtual machines (VMs) in cloud data centers. Live migration of multi-tier applications across geographically distributed data centers is important for load management, power saving, routine server maintenance and quality-of-service. Different from a single-VM migration, VMs in a multi-tier application are closely correlated, which results in a *correlated VM migrations* problem. Current live migration algorithms for single-VM cause significant application performance degradation because intermediate data exchange between different VMs suffers relatively low bandwidth and high latency across distributed data centers. In this paper, we design and implement a coordination system called *VMbuddies* for correlated VM migrations in the cloud. Particularly, we propose an adaptive network bandwidth allocation algorithm to minimize the migration cost in terms of migration completion time, network traffic and migration downtime. Experiments using a public benchmark show that VMbuddies significantly reduces the performance degradation and migration cost of multi-tier applications.

Index Terms—Cloud, live migration, multi-tier application, virtual machine

1 Introduction

Internet applications have been prosperous in the era of cloud computing, which are usually hosted in virtual machines in geographically distributed data centers. Live migration of Internet applications across data centers is important for different scenarios including load management, power saving, routine server maintenance and quality-of-service [8], [11], [38], [42]. Additionally, Internet applications tend to have dynamically varying workloads that contain long-term variations such as time-of-day effects in different regions. It is desirable to move the interactive/web application to the data center that has better network performance to users for lower response time [41]. Also, workloads can be migrated across different data centers to exploit time-varying electricity pricing [32]. The recent advance of VM live migration techniques [10], [30] is able to relocate a *single VM* across data centers with acceptable migration cost [34], [38]. Typical Internet applications employ a multi-tier architecture, with each tier providing certain functionality. Specific to multi-tier applications, we need to migrate several tightly-coupled VMs in multi-tiers, instead of a single VM. Previous studies have demonstrated the potential performance penalty of multi-tier applications during migration [17], [23]. In this paper, we investigate whether and how we can reduce the migration cost without suffering application performance degradation.

A typical multi-tier web application consists of three tiers: presentation layer (web tier), business logic layer (App tier) and data access layer (DB tier) [18]. Different layers usually run on different VMs and have different memory access patterns. VMs are correlated because only when all VMs of the multi-tiers are migrated to another data center, they can completely and efficiently serve requests in that data center. We call this problem *correlated VM migrations*. Correlated VM migrations can cause significant performance penalty to multi-tier applications. Consider the following scenario: if the middle tier is first migrated, then the other two tiers must redirect the communication and data access traffic to another data center and wait for the processing results to be sent back. Moreover, because the multi-tier application and migration processes share the same link for data transferring, given the data-intensive nature of multi-tier applications and limited network bandwidth between two data centers, network bandwidth contention may cause significant performance degradation for both applications and VM migrations.

While live migration of VMs provides the ability to relocate running VMs from one physical host to another without perceivable service downtime [10], [30], the state-of-the-art VM migration techniques mainly target a *single* VM (either within a data center [10], [16], [24], [30] or across different cloud data centers [8], [38], [42]). These techniques cannot fundamentally solve the correlated VM migrations problem. We need effective and efficient mechanisms to coordinate correlated VM migrations across distributed data centers.

Ideally, the coordination system should avoid splitting multiple tiers between data centers so that the performance penalty of VM migrations on the multi-tier application is minimized. Meanwhile, we should diminish the VM migration cost in terms of migration completion time, network traffic and migration downtime. To achieve these goals, we need to address the following technical challenges. First,

- *H. Liu is with North China Electricity Power University, Baoding, HeBei 071003, China, and Nanyang Technological University, Singapore 635798. E-mail: haikunliu@gmail.com.*
- *B. He is with the School of Computer Engineering, Nanyang Technological University, 50 Nanyang Avenue, N4-02C-116, Singapore 635798, Singapore. E-mail: he.bingsheng@gmail.com.*

Manuscript received 8 Dec. 2013; revised 1 Apr. 2014; accepted 2 Apr. 2014. Date of publication 7 Apr. 2014; date of current version 6 Mar. 2015. Recommended for acceptance by S. Papavassiliou.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPDS.2014.2316152

migration completion time of different VMs may vary significantly due to VM configurations and workload characteristics in different tiers. It is challenging to orchestrate their migration progress and complete the migrations simultaneously. Second, network bandwidth is the critical resource of VM migration across data centers. So far there is little work on progress management for correlated VM migrations so that the total migration cost is minimized. The last challenge is how to implement a coordination system in existing virtualization platforms such as Xen and demonstrate its practical value.

To address the aforementioned challenges, we propose a coordination system called VMbuddies to solve the correlated VM migrations problem for multi-tier applications. VMbuddies embraces a synchronization protocol to ensure that all VMs complete their migrations simultaneously, avoiding intermediate data exchange across distributed data centers. In order to reduce the migration cost, we develop network bandwidth allocation algorithms according to memory access patterns of different workloads. For VMs with stable memory access pattern, we propose an analytical cost model of pre-copying algorithm that is widely used for live VM migrations in common virtualization platforms such as VMware, Xen, KVM and Hyper-V. Based on the cost model, we formulate the bandwidth allocation as a distributed constraint optimization problem and give an optimal solution so that the total migration cost is minimized. We further develop an adaptive bandwidth allocation algorithm for VMs with dynamic memory access patterns. With the synchronization protocol and bandwidth allocation mechanisms, we design and implement a prototype called VMbuddies in Xen.

We investigate the effectiveness and efficiency of VMbuddies by using a public multi-tier application benchmark RUBBoS [1]. The experiments are conducted with two complementary approaches: real private data centers and simulated wide area network (WAN). The evaluation compares a number of metrics (such as migration completion time, network traffic, migration downtime and application performance degradation) with several alternative approaches. The results show VMbuddies can significantly reduce the performance degradation of multi-tier applications during migrations, while only experiencing slight overhead of synchronization cost in terms of migration completion time and network traffic. Particularly, in comparison with Xen, VMbuddies achieves lower average response time by 77 percent and higher throughput by 18 percent, and reduces total migration cost by 36 percent on average.

The rest of this paper is organized as follows. Section 2 gives a brief introduction to our groundwork and formulates the correlated VM migration problem. Section 3 presents the synchronization protocol and bandwidth allocation algorithms. Section 4 describes the implementation details of VMbuddies. We present the evaluation methodologies and experimental results in Section 5. We review the related work in Section 6 and conclude in Section 7.

2 Correlated VM Migration Problem

This section introduces the background of single-VM migration techniques and correlated VM migrations.

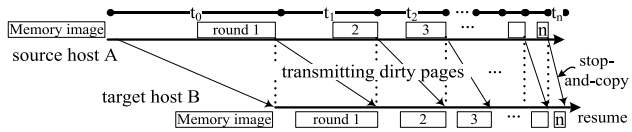

Fig. 1. Live migration algorithm performs pre-copying in iterative rounds.

Single-VM migration. Live migration of VMs has been an effective approach to manage workloads in a non-disruptive manner. Among different implementations of VM live migration, pre-copying algorithm [10], [30] is the most popular and widely used in today's virtualization platforms. Thus, this paper focuses on pre-copying algorithm in Xen, and the methodology in our paper can be extended to other VM migration techniques.

The basic workflow of pre-copying algorithm in Xen is described as follows. As shown in Fig. 1, memory pre-copying is conducted in several iterative *rounds*. The VM's physical memory is first transferred from host A to host B, while the source VM continues running in host A. Pages dirtied in each round must be iteratively re-sent in the next round to ensure memory consistency. That is, the data to be transmitted in each round are dirty pages generated in the previous round. After several rounds of pre-copying, a stop-and-copy phase is performed to transmit the remaining dirty pages while the source VM temporarily stops execution. When the final data transferring is done, the VM on host B resumes and takes over the VM on host A.

There are a number of factors affecting the migration cost in terms of migration downtime, migration completion time and total network traffic. The major factors include the size of VM memory, memory dirtying rate, network transmission rate and configuration of migration algorithm (e.g., conditions for starting the stop-and-copy phase). Among these factors, the size of VM memory and the memory dirtying rate are mostly determined by the VM and workloads. That means, they usually cannot be controlled by migration algorithms. Thus, this paper focuses on the optimizations on the latter two factors: network transmission rate and configuration of migration algorithms. We note that CPU resource show moderate performance impact on VM migrations. Previous study demonstrated that the migration process only takes around 30 percent of one CPU to attain the maximum network throughput over the gigabit link [30]. Based on this observation, we think network bandwidth is usually more important than CPU resource, especially for VM migration in WAN environments. Moreover, this paper focuses on the algorithms of network bandwidth allocation for VM migration, so we assume the CPU resource is always sufficient for migration processes.

Correlated VM migrations. The objective of this paper is to solve the correlated VM migrations problem raised in multi-tier applications. As illustrated in Fig. 2, a multi-tier web application is migrated from DC 1 to DC 2. Each tier is running on multiple VMs, and thus the VMs across different tiers are correlated with data dependency. Network bandwidth is a critical resource across distributed data centers. It is usually much smaller than the network bandwidth within a data center. Previous studies (e.g., [38]) assumed that the network bandwidth between two data center was 465 Mbps. Without loss of generality, we

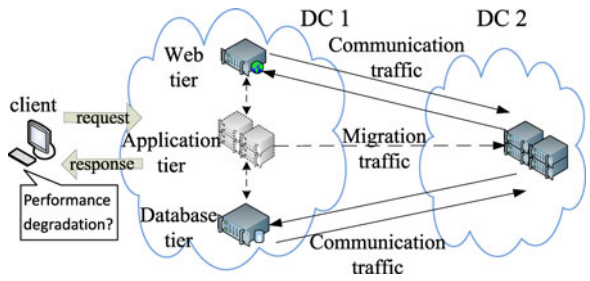

Fig. 2. Performance penalty due to live migration of a multi-tier web application across distributed data centers.

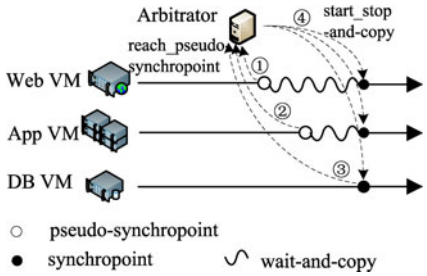

Fig. 3. Synchronization of live migration of multi-tier applications.

assume that the peak network bandwidth between DC 1 and DC 2 reserved for the migration processes is B. As discussed in Introduction, the application traffic and migration traffic share the same links between two data centers. The bandwidth contention between them may result in significant performance degradation in both application and VM migration. That motivates us to develop VMbuddies to coordinate the VM migration of multi-tier applications, and let the inter-cloud network bandwidth to be exclusively used by the migration traffic.

3 System Design

In this section, we first present the synchronization protocol for coordinating live migration of VMs in a multi-tier application. Next, we present bandwidth allocation algorithms to reduce the migration cost of all VMs. Particularly, we first consider the scenario where the memory dirtying rate is stable in the entire VM migration. We start with a cost model in estimating the migration completion time of all VMs, and develop an optimal bandwidth allocation algorithm to minimize the migration completion time. With this optimal algorithm as a building block, we identify the stable phases in memory dirtying rate, and develop an adaptive algorithm for bandwidth allocation in VMbuddies. In the adaptive algorithm, the bandwidth allocation for each stable phase can use the optimal algorithm developed in the scenario for stable memory dirtying rates.

3.1 Synchronization Protocol

To avoid inter-data center network traffic between correlated VMs, we develop a synchronization protocol to orchestrate all VMs to proceed the stop-and-copy phase at the same time.

As shown in Fig. 3, each VM migration may reach its stop-and-copy phase at different points of time (called *pseudo-synchropoint*). The pseudo-synchropoint depends on the termination conditions of pre-copying algorithm. In our synchronization protocol, we postpone the stop-and-copy phase until all VMs reach the stop-and-copy phase (called *synchropoint*). However, all VMs are still running during the synchronization, and the dirtied memory pages still need to be transmitted to the destinations. We call this phase "*wait-and-copy*". The bandwidth consumed in this phase is determined by the memory dirtying rates of the VMs. Algorithm 1 shows the pseudo code of the synchronization protocol for correlated VM migrations. The synchronization protocol relies on an arbitrator for control. The arbitrator implements a message-passing mechanism for controlling the VM migrations. When a VM reaches the pseudo-synchropoint, it should immediately send a message "*reach_pseudo-synchropoint*" to the arbitrator, and then proceed the "*wait-and-copy*" phase until it receives the "*start_stop-and-copy*" message from the arbitrator. The arbitrator uses a variable p to record the number of VMs that have reached the pseudo-synchropoint. Once all VMs have reached the synchropoint, the arbitrator broadcasts a message "*start_stop-and-copy*" to all VMs. To handle the potential migration failures, we adopt a simple approach for fault tolerance. We view the coordinated migration processes as a transaction in a batch. In case of failures, all correlated VM should resume at their original host, aborting the migration. More advanced fail-tolerant VM migration techniques will be studied in our future work.

Algorithm 1: Synchronization protocol for correlated VMs live migration

```
1.  let m be the number of VMs to be migrated
2.  let p ← 0      /*the number of VMs that reach the pseudo-synchropoint */

3.  Begin Migrate (VM[i])
4.      while VM[i] does not reach the pseudo-synchropoint  do
5.          pre-copy the memory of VM[i];
6.      endwhile
7.      send message "reach_pseudo-synchropoint" to arbitrator;
8.      proceed "wait-and-copy" phase;
9.      if receive message "start_stop-and-copy"  then
10.         proceed "stop-and-copy" phase;
11.     endif
12. End Migrate

13. Begin Arbitrator( )
14.   if receive message "reach_pseudo-synchropoint" form VM[i] then
15.       p ← p+1;
16.       if p == m  then      /*All VMs reach the synchropoint */
17.           for i=1 to m do
18.               send message "start_stop-and-copy" to VM[i];
19.           endfor
20.       endif
21.   endif
22. End Arbitrator
```

The synchronization protocol allows different resource allocations and VM scheduling mechanisms. Without an effective arbitration of resource allocation and scheduling, the synchronization may cause significant migration cost and performance degradation. To address this challenging problem, we investigate different bandwidth allocation algorithms to minimize migration cost of correlated VMs.

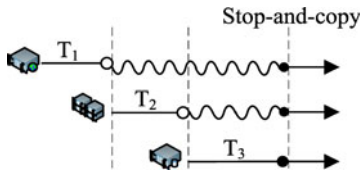

Fig. 4. LLL scheduling of multiple VM migrations.

3.2 A Naive Bandwidth Allocation

The synchronization protocol guarantees that all the correlated VM migrations start the *stop-and-copy* phase at the same time. A naive bandwidth allocation method is scheduling the VM migrations one by one. Each VM migration gains the majority of the network bandwidth for the best migration performance. As shown in Fig. 4, a VM starts live migration only when another VM has reached the pseudo-synchropoint. However, when the VM reaches the pseudo-synchropoint, it should proceed the "wait-and-copy" phase to postpone the *stop-and-copy* phase. This stage still consumes a portion of bandwidth (determined by its memory dirtying rate) to re-send the dirty pages. Clearly, the order of VM migrations affects the bandwidth allocation. To reduce the synchronization cost, an intuitive scheduling is to start VM migrations in the ascending order of their memory dirtying rates. The VM with the largest memory dirtying rate should be migrated last. Thus, we call this scheduling algorithm Largest memory Load VM Last (LLL).

LLL is simple but cannot effectively use the network bandwidth. The VMs in the *wait-and-copy* phase still consume network bandwidth, causing excessive network traffic between data centers. This overhead is exaggerated when the number of correlated VMs increases, and for the memory-intensive workloads like multi-tier applications. To address the overhead caused in the *wait-and-copy* phase, we propose a parallel VM migration scheduler and adaptive bandwidth allocation algorithms for all the VM migrations, as described in the next sections.

3.3 Optimal Bandwidth Allocation for Static Workloads

Unlike LLL, VMbuddies performs VM migrations in multiple tiers simultaneously, and smartly decide the bandwidth among VMs. A primary goal of VMbuddies is to determine how much network bandwidth should be allocated to each migration process to minimize total migration cost of multi-tier applications. We first consider a scenario that the memory access pattern of each VM is stable (i.e., memory dirty rate is stable and can be modeled as a constant), and then extend this algorithm to dynamic workloads.

3.3.1 Modeling Static Workloads Migration

Our model on static workload migration uses a single VM as the building block. In the following, we present an analytical model of pre-copying algorithm to estimate the migration cost of a single VM. Table 1 summarizes the parameters and notations used throughout this paper. Assume the pre-copying algorithm proceeds in n $(n \leq N)$ rounds. Let $v_i (0 \leq i \leq n)$ denote the data volume transmitted at each pre-copying round, and $t_i (0 \leq i \leq n)$ denote the elapsed time at each round. v_0 equals the VM memory size M. t_0 represents the time consumed for transferring the VM memory image and t_i $(1 \leq i \leq n)$ is the time of transferring the dirty pages generated during previous rounds. The data transmitted in round i is calculated in Eq. (1).

TABLE 1
Parameters for VM Live Migration Modeling

Symbol	Description
M	Size of VM memory image.
V	Total network traffic during migration.
T	Migration completion time.
r	Network transmission rate during migration.
d	Memory dirtying rate during migration.
B	Maximum network bandwidth reserved for VMs migration traffic.
V_{thd}	Threshold of the remaining dirty memory that should be transferred during the stop-and-copy phase.
N	The pre-defined maximum number of rounds for iterative pre-copying in migration algorithm.
W	The Writable Working Set (WWS) of applications.

$$v_i = \begin{cases} M, & \text{if } i = 0; \\ d \cdot t_{i-1}, & \text{otherwise.} \end{cases} \tag{1}$$

The elapsed time at round i is calculated in Eq. (2).

$$t_i = \frac{d \cdot t_{i-1}}{r} \tag{2}$$

We can derive the expressions of v_i and $t_i$step by step as follows: (1) $v_0 = M, t_0 = \frac{M}{r}$; (2) $v_1 = d * t_0 = \frac{Md}{r}, t_1 = \frac{v\_1}{r} = \frac{Md}{r^2}$; (3) $v_2 = d * t_1 = \frac{Md^2}{r^2}, t_2 = \frac{v\_2}{r} = \frac{Md^2}{r^3}$; and finally we get

$$v_i = d * t_{i-1} = \frac{M \cdot d^i}{r^i}, \quad t_i = \frac{M \cdot d^i}{r^{i+1}}. \tag{3}$$

Then the total network traffic during the migration can be summed up in Eq. (4)

$$V = \sum_{i=0}^{n} v_i = M \cdot \sum_{i=0}^{n} \frac{d^i}{r^i}. \tag{4}$$

Consequently, the migration completion time can be summed up in Eq. (5)

$$T = \sum_{i=0}^{n} t_i = \frac{M}{r} \cdot \sum_{i=0}^{n} \frac{d^i}{r^i}. \tag{5}$$

The migration completion time is the duration that has negative effect on application performance. It is a key performance metric of migration cost, especially for multi-tier applications migrating in cloud environments.

To evaluate the convergence rate of VM migration algorithm, we calculate the total number of rounds by the inequality $v_n \leq V_{thd}$. It is the condition to terminate the iterative pre-copying and to start the stop-and-copy phase. Furthermore, it should not be larger than the pre-defined parameter N. As a result, the number of pre-copying iterations becomes:

$$n = \min\left\{\left\lceil \log\frac{V_{thd}}{M} \Big/ \log\frac{d}{r}\right\rceil, \mathrm{N}\right\}. \quad (6)$$

For a given VM, M and V_{thd} (determined by migration algorithms) can be viewed as constants. Consequently, the iterative pre-copying would converge faster if d/r is smaller. We therefore define d/r as the *convergence coefficient* of VM live migration.

The above analysis assumes that the memory dirtying rate is smaller than the memory transmission rate on average. However, in practice, there are still scenarios that the memory dirtied at a higher speed than the network transfer. One observation is that, the memory dirtied in each round should never exceed the applications' writable working set (WWS) [10]. In this case, the pre-copying terminates once the number of iterations exceeds N or the network traffic exceeds the VM memory size by a factor of β (by default, the factor is 3 in Xen migration algorithm). Finally, in case where $d/r > 1$, the total network traffic and migration completion time should be estimated by Eq. (7) and Eq. (8), respectively

$$V = \min\{M + N * \alpha W, \beta M + W\}, \quad (7)$$

$$T = \min\{M + N * \alpha W, \beta M + W\}/r, \quad (8)$$

where αW is the amount of dirty memory to be transferred in each round of pre-copying, and α ($0 < \alpha < 1$) is a coefficient of proportionality that can be learnt by linear regression technique [26]. $M + N * \alpha W$ denotes the total memory transferred in maximum number of rounds, and $\beta M + W$ denotes the total memory transferred if migration traffic exceeds β times of the VM memory size. As the average memory dirtying rate d and W are both determined by the applications and can be measured in a long time window before VM migrations, we only need to determine the network transmission rate (i.e., parameter r), which can be referred to Eq. (5) or Eq. (8).

With the cost model of a single VM migration, we formulate the problem of correlated VM migrations in cloud environments. Assume the multi-tier application is comprised of m VMs. We should orchestrate the VM migrations to minimize the maximum migration completion time of all the VMs by allocating appropriate network bandwidth to each migration process. As the network bandwidth between geographically distributed data centers is shared and limited resource, this issue can be abstracted as a distributed constrain optimization problem (DCOP), as shown in Eqs. (9). Note, we instantiate the parameter T, V, M, d, r for VM k to be T_k, V_k, M_k, d_k, and r_k, respectively

$$\begin{cases} \min\{\max\{T_1, T_2, \cdots, T_m\}\} \\ T_k = f(r_k) = \sum_{i=0}^{n} t_i = \frac{M_k}{r_k} \cdot \sum_{i=0}^{n} \frac{d_k^i}{r_k^i} \\ s.t., \sum_{k=1}^{m} r_k \le B \\ s.t., d_k < r_k < B. \end{cases} \quad (9)$$

In case where $d_k \ge r_k, T_k = f(r_k)$ in Eqs. (9) should be replaced by Eq. (8).

3.3.2 Problem Solving

DCOP is originally a problem in which a group of agents must distributedly choose values for a set of variables so that the cost of a set of constraints over the variables is either minimized or maximized. It is known that DCOPs are NP-hard problems [44]. There are no polynomial-time algorithms to solve this NP-hard problem. However, observing that the polynomial in Eqs. (9) is a monotonic function of r, we can simplify the problem by Theorem 1.

Theorem 1. *When a set of VMs are migrated concurrently, if there exists an optimal bandwidth allocation to minimize the maximum migration completion time of the VMs, then the bandwidth resource should be fully used, i.e.,* $\sum_{k=1}^{m} r_k = B$, *and all VMs have equivalent migration completion time, i.e.,* $\forall \mathrm{i}, \mathrm{j}, \exists\ T_i = T_j$.

Proof (Sketch). Assume live migration of each VM is performed by a migration daemon concurrently. Given an arbitrary bandwidth allocation for these processes with constraint $\sum_{k=1}^{m} r_k = B$, the migration completion time of each VM can be figured out by $T_k = f(r_k)$, which is a monotonically decreasing function that T always decreases as r increases in the open interval (0, B). In case there is a difference of migration completion time between two arbitrary VMs, we deprive the migration daemon with a shorter completion time some bandwidth and re-allocate it to the migration daemon with a longer completion time. By iteratively using the step-by-step approximation method to adjust bandwidth resource for each migration process, all VMs should have the same migration completion time finally. □

Insight. Based on Theorem 1, the synchronization cost between different VMs is avoided as all VMs finish their migration at the same time. Our objective $min(max(T_1, T_2, \ldots T_m))$ can be transformed to a problem of solving multivariate polynomial equations over a finite field as Eqs. (10):

$$\begin{cases} T_1 = T_2 = \cdots = T_m \\ T_k = f(r_k) = \sum_{i=0}^{n} t_i = \frac{M_k}{r_k} \cdot \sum_{i=0}^{n} \frac{d_k^i}{r_k^i} \\ s.t., \sum_{k=1}^{m} r_k = B \\ s.t., d_k < r_k < B \end{cases} \quad (10)$$

In case where $d_k \ge r_k$, the $T_k = f(r_k)$ in Eqs. (10) should be replaced by Eq. (8). We note that if the above equations can be solved, the objective $min(\sum_{k=1}^{m} V_k)$ is also satisfied because the bandwidth resource is fully utilized and the migration completion time is minimized. However, Eqs. (10) are still hard to solve because $T_k = f(r_k)$ is a polynomial with high order. Fortunately, as $T_k = f(r_k)$ is a monotonically decreasing function in the field (0, B), there exists only one solution to satisfy the above equations. As $\sum_{k=1}^{m} r_k = B$ is a multivariate linear equation, if we search the solution by changing the multi-variate r_k, there should be numberless combinations that need to verify equations $T_1 = T_2 = \cdots = T_m$. In contrast, observing that $T_k = f(r_k)$ contains only one variate and their curves are monotonic and similar, we design a work backward strategy to solve

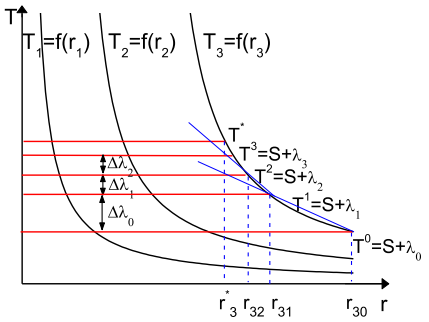

Fig. 5. A quasi-secant method to solve the multivariate polynomial equations.

Eqs. (10). That is, we tentatively search the solution set of equations $T_1 = T_2 = \cdots = T_m$ to satisfy $\sum_{k=1}^{m} r_k = B$.

We propose a quasi-secant method [2] to approximate the Eq. $\sum_{k=1}^{m} r_k = B$ iteratively, as shown in Fig. 5 and Algorithm 2. The solution consists of three steps. First, we figure out the minimum of $f(r_k)$ in the field $(0, B]$, denoted by $S = \min\{f(r_1), f(r_2), \ldots, f(r_m)\}$. Second, we use horizontal lines $T^i = S + \lambda_i$ $(\lambda_i > 0)$ to sweep the curves $T_k = f(r_k)$, where λ_i is the step size to increase the horizontal line T^i. Let r_{ki} denotes the root of $f(r_k) = S + \lambda_i$ in the iteration i. We use Newton's method [31] to approximate the root of these equations one by one and get a solution set, for example $R_0 = (r_{10}, r_{20}, r_{30})$. Third, we check whether the solution R_0 satisfies $\sum_{k=1}^{m} r_{ki} = B$ by calculating the error $d_i = \left|\sum_{k=1}^{m} r_{ki} - \mathrm{B}\right|$. The iterations will be terminated when the error $d_i \leq \delta$, where δ is a very small constant. The problem is how to determine the step size λ_i to speed up the roots searching.

We illustrate the iterative approximating process by a sketch diagram in Fig. 5. Take $T_3 = f(r_3)$ as an example, the line T^0 and T^1 sweeping it gets two points (r_{30}, T^0) and (r_{31}, T^1), respectively. Suppose the final solution of $T^* = f(r_3)$ is r_3^*, the line passing through these two consecutive points intersecting the vertical line $r = r_3^*$ determines λ_2. Then the next root r_{32} can be solved. Without lose of generality, λ_i can be deduced from the proportional relationship of three consecutive roots, as shown in Eq. (11):

$$\lambda_{i+1} = \lambda_i + \frac{d_i(\lambda_i - \lambda_{i-1})}{d_{i-1} - d_i}, \tag{11}$$

we define $\Delta\lambda_i = \lambda_{i+1} - \lambda_i$, and then we have:

$$\Delta\lambda_i = \frac{d_i \cdot \Delta\lambda_{i-1}}{d_{i-1} - d_i}. \tag{12}$$

By iteratively changing the step size in Algorithm 2, it should approximate the target solution finally. As the convergence order of quasi-secant method is the golden ratio (1.618), Algorithm 2 converges fast in practice. Actually, there are $C_m^2 = \frac{m(m-1)}{2}$ equations in Eqs. (10). However, step (2) in Algorithm 2 only needs to solve m equations in each round of iteration. The computation cost is linearly proportional to the number of VMs to be migrated. In this way, Algorithm 2 simplifies the problem complexity of root finding. In our experiments, the proposed approach is able to solve more than 10 high orders equations within one second, which is sufficient for typical multiple tier applications in practice. In contrast, a general *solve* method provided by Matlab cannot give out the solution in 12 hours.

Algorithm 2: Optimal bandwidth allocation for static workloads

(1) $S \leftarrow min(f(r_1), f(r_2), \ldots, f(r_m))$, $i \leftarrow 0$, $\lambda_0 \leftarrow 0$, $\lambda_1 \leftarrow q$, q is a random number bigger than 0, so $\Delta\lambda_0 = q$;

(2) Let $f(r_k) \leftarrow S + \lambda_i, (1 \leq k \leq m)$, solve these equations and find out a solution set $R_i = (r_{1i}, r_{2i}, \cdots, r_{mi})$;

(3) Verify whether the solution satisfy equation $\sum_{k=1}^{m} r_{ki} = B$ by calculating the distance $d_i = |r_{1i} + r_{2i} + \cdots + r_{mi} - \mathrm{B}|$, if $d_i > \delta$, then $i \leftarrow i + 1$, $\lambda_{i+1} \leftarrow \lambda_i + \Delta\lambda_i$, where $\Delta\lambda_i = d_i * \Delta\lambda_{i-1}/(d_{i-1} - d_i)$, $(i > 0)$, go to (2); otherwise, R_i is the target solution and the searching terminates.

(4) Allocate the bandwidth r_{ki} for live migration of VM-k.

3.4 Adaptive Bandwidth Allocation for Dynamic Workloads

The model and algorithm discussed in the preceding section assume that the memory dirtying rates are steady for all VMs. However, in practice, multi-tier applications can be service-oriented and dynamic (e.g., due to the daily or seasonal changes). The memory access pattern of different applications may vary in response to the load change of the service requests.

We design our bandwidth allocation algorithm adapting to the temporal changes of page dirtying rates. Particularly, we rely on the prediction on the page dirtying rate in a short-term period (namely *window*). We identify the relatively stable periods (namely *phases*), and view the bandwidth allocation problem for each phase as an instance of optimal bandwidth allocation problem in the scenario of stable page dirtying rate. That means, we can leverage the model and the optimal algorithm in the previous section to achieve local optimum in each phase.

Prior to starting VM migrations, we first measure the page dirtying rate of each VM during a long-term run. We then model memory access pattern of each VM based on the profiling runs. The modelling methodology is orthogonal to this paper. There are a number of existing learning methods and models [13], [39], which have demonstrated high accuracy and efficiency. For modelable workload such as on-off memory access patterns, as shown in Fig. 6a, we formulate the page dirtying rate into phases [39]. We first divide the workload curves into several small splines evenly, and then combine two successive splines iteratively if their difference is smaller than a pre-defined threshold. We recognize a significant pulse as a new phase if its successive spline has slight variation. As shown in Fig. 6a, the migration daemon orchestrates live migration of correlated VMs using the optimal bandwidth allocation algorithm in each stable phase. The migration daemons track the pages dirtied and analyze the phase change according to the historical statistics. Once a significant change of page dirtying rate is detected, they immediately report to the arbitrator for bandwidth re-allocation in

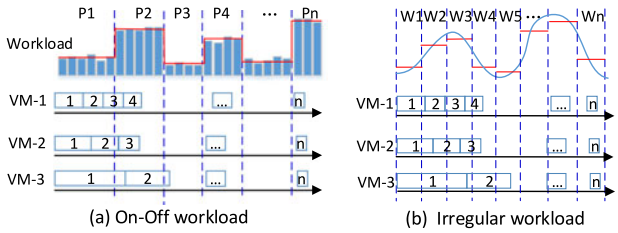

Fig. 6. Bandwidth allocation according to workload patterns: (a) model on-off workloads in several stable phases of page dirtying rate; (b) model irregular workloads in fix-sized time windows through average page dirtying rate.

response to the phase changes. The remaining data to be transferred also should be reported to the arbitrator for decision making. For simplicity, the bandwidth allocation takes effect at the new round of VM migration. At last, the synchronization protocol described in Section 3.1 is used to achieve the ultimate synchronization of correlated VM migrations.

We also need to handle the scenarios where some workloads may not show distinct phase changes, as shown in Fig. 6b. We can only measure the average memory dirtying rate in each time window, and thus the adaptive bandwidth allocations is sensitive to the sizes of time window. A smaller window means higher accuracy of memory dirtying rate prediction, but causes higher overhead due to the computation of bandwidth re-allocation. We need to carefully determine the size of windows to make tradeoff between predication accuracy and computation overhead. We experimentally study its impact in the evaluation (Section 5.5). In this scenario, the arbitrator periodically performs bandwidth reallocation in the end of each time window, so the migration daemons only need to sum up the pages to be transferred in next time window, including the remaining pages in current round of pre-copying and new pages dirtied in current window. The arbitrator again leverages the optimal algorithm described in the previous section to allocate bandwidth to each migration deamon.

4 Implementation

In this section, we present the details of design and implementation on VMbuddies. We design and implement VMbuddies based on Xen 4.1 platform. We conjecture that the techniques and optimizations proposed in this paper are also applicable for other virtualization platforms. Our implementation carefully considers efficiency and transparency of correlated VM migrations across data centers.

4.1 System Architecture

Fig. 7 shows the system architecture of VMbuddies. The system is composed of two major modules, namely migration daemon and arbitrator. The migration daemon in domain 0 consists of several components such as dirty memory tracker, data transmission rate controller, synchronization protocol, and some migration optimizations. The arbitrator consists of VM memory information collector, migration cost prediction model, bandwidth allocator, and VM migration scheduling components. When a multi-tier application needs to be migrated, the arbitrator

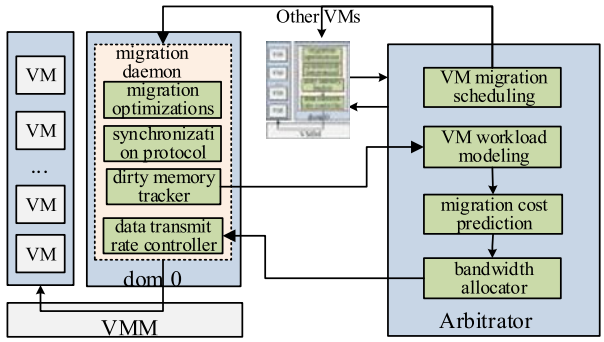

Fig. 7. Block diagram of system architecture.

needs to conduct the following actions for bandwidth allocations: 1) activates the dirty memory tracker in each migration daemon, 2) collects the page dirtying information of each VM, 3) performs workload predictions and calculates the memory dirtying rate, 4) estimates the migration cost, and 5) finally allocates the network bandwidth to each migration process. These actions are implemented with message-passing mechanisms among the arbitrator and other components in VMbuddies. Next, the VMs are migrated at the speed of the allocated bandwidth in multiple phases, according to Section 3.4. In the following, we first present two migration optimizations and then describe some other implementation details.

4.2 Optimizations of VM Live Migration

Although there is a lot of existing migration optimizations for cost reduction, we choose the most effective and simple ones for our scenario: multi-tier applications migration in WAN environments. Particularly, we highlight two techniques—ballooning to evict unused pages from VMs' memory image, and an intelligent pre-copying termination to adjust the number of iterations in the pre-copying algorithm.

Ballooning. It is desirable to flush non-essential data out of memory, and to transfer only a smaller working set during migration. VMbuddies leverages ballooning mechanism [16], [36] to reduce memory volume transferred during VMs live migration. We first profile a VM's memory usage to detect its memory footprint, and then the size of free memory can be determined. The balloon driver then requests the size of free memory from the guest OS kernel and returns it to the underlying hypervisor. In this way, ballooning mechanism deflates a VM's memory size by evicting free pages from the VM's memory image. This is extremely useful for VMs with large memory size and small application working set.

Intelligent pre-copying termination. Multi-tier applications can have very different memory and computation characteristics. Instead of using static conditions on pre-copying terminations as modeled in Section 3, VMbuddies terminates the pre-copying phase intelligently when the pre-copying iterations cannot reduce the migration downtime. Current Xen terminates the iterative pre-copying till either (1) the remaining dirty memory becomes smaller than 50 pages; or (2) the number of iterations exceeds 30 times; or (3) the network traffic exceeds the VM memory size by a factor of 3. However, our experiments on multi-tier

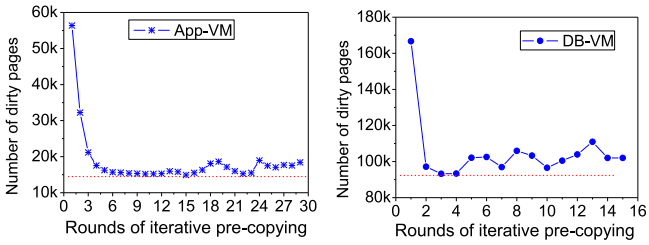

Fig. 8. Dirty pages generated in each round of pre-copying.

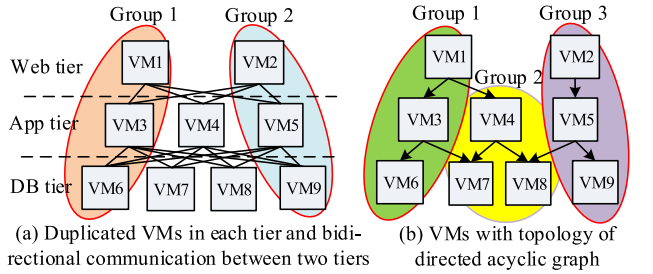

Fig. 9. Grouping VMs of different tiers to lower the parallelism degree of VM migrations in the cloud.

applications indicate that the first condition is seldom satisfied, and the migration algorithm often performs many unnecessary iterations. Fig. 8 shows the number of dirty pages generated in the each round of iterative pre-copying when we migrate a 3-tier web benchmark RUBBoS over a link with 400 Mbps bandwidth. The migrations of application-tier and DB-tier terminate the pre-copying phase according to conditions (2) and (3), respectively. However, only the first several rounds of pre-copying results in reduction of memory remaining to be sent in the next round. This indicates that the migration algorithm should intelligently determine the time to stop pre-copying to avoid unnecessary iterations.

VMbuddies leverages a heuristic to terminate iterative pre-copying. We observe that memory dirties approximately as fast as it can be transferred after a number of iterations. To detect this point, we monitor the number of pages sent and dirtied in each round until the difference between them becomes very small. At this time, the migration daemon should immediately notify the arbitrator that the VM reaches the pseudo-synchropoint. This simple optimization can significantly reduce the migration completion time and network traffic, with up to 45 percent reduction of migration cost on average in our experiments.

4.3 VM Grouping

Today's enterprise multi-tier applications are usually comprised of multiple VMs in each tier. Fig. 9a shows a web application with 2/3/4 configuration (i.e., 2, 3 and 4 VMs for web tier, application tier and database tier, respectively). If these VMs are migrated concurrently, the performance of VM migrations would be significantly degraded due to network bandwidth contention. To address this scalability problem, we propose VM grouping mechanism to lower the parallelism degree of multiple VM migrations. The other consideration is the layout of VMs. The topology of a multi-tier application (such as tree or directed acyclic graph (DAG)) usually affects the decision making of grouping and ordering correlated VM migrations. Overall, we consider the following two scenarios.

First, we consider a scenario where multiple VMs in each tier have the same function, and thus the multiple replicas are used for load balancing and fault tolerance, as shown in Fig. 9a. Actually, if we keep at least one VM of each tier in both source data center and destination data center, then the service is available at both sides after a re-configurationof connection management. A feasible grouping scheme is shown in Fig. 9a. If the VMs in Group 1 are the first to be migrated concurrently, then the VMs in Group 2 must be migrated in the end. This guarantees that at least one minimal organism can serve the requests at both sides without inter-datacenter communication. The other VMs excluded from these two groups can be migrated one by one to fully use the bandwidth resource.

Second, we consider a scenario where the applications show complicated topologies such as tree or directed acyclic graph, and multiple VMs have different functions, as shown in Fig. 9b. In this case, we need to minimize the inter-datacenter network traffic. We model the application's dependencies as a graph, where VMs are vertices and data flows as edges. The problem can be formulated as a uniform graph partitioning problem. We need to find the minimum cuts of the graph so that the cut sets have the smallest number of edges (unweighted case) or smallest sum of weights. We choose the most commonly used Fiduccia-Mattheyses algorithm [15] to solve this multilevel graph partitioning problem due to its high efficiency. For example, a three-way partitioning can split the DAG in Fig. 9b into three subgraphs. We note that such processing can be done prior to VM migrations so that no computation overhead is imposed on the migration daemons and applications.

4.4 Other Implementation Details

There are still many other implementation details about VMbuddies, including memory dirtying rate measurement, network bandwidth controller, disk migration, and network connections maintance. We provide these details in the Appendix of the supplementary file, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/ TPDS.2014.2316152.

5 Evaluations

In this section, we evaluate application performance degradation and VM migrations cost in terms of the following metrics:

1) *Service level degradation*. The response time and throughput of multi-tier applications during migration.
2) *Migration completion time*. The elapsed time from starting the first VM migration to the time that all VMs finish migration.
3) *Network traffic*. The total data volume transmitted during the VM migrations.
4) *Migration downtime*. It is the duration that the services of multi-tier applications are entirely unavailable.

5.1 Experimental Setup

Testbeds. We evaluate VMbuddies in both real WAN and in-lab testbeds. We deploy our prototype in data centers at Nanyang Technological University (NTU) and National University of Singapore (NUS). Our measurement has showed a stable throughput of 40 Mbps and a round trip latency of 20 ms between the two sites connected via VPN. At both sites, we have HP Z400 workstations equipped with Intel Xeon W3680 3.33 GHz 6 processors, 12 GB DDR3 memory, one 500 GB 7,200 rpm SATA hard disk, and 1 Gbit Ethernet interface. The host machines are with 64-bit Fedora 16 Linux distribution and the hypervisor is Xen 4.1 with Linux 3.1 kernel. The hosts are allocated with two CPU and 4 GB memory to insure that the resource for migration daemons is sufficient. The VMs run in para-virtualization mode, with the OSes having the same distribution of Linux as the hosts. By default, all VMs are configured to use 1 CPU and 1 GB RAM. This allows us to easily simulate the overloaded scenario in the benchmark. Because the real WAN testbed cannot cover all test cases due to the limited bandwidth, we also simulate a WAN testbed in our lab by using the Linux traffic shaping interface, which can freely control the bandwidth, latency and package loss experienced in WAN environments.

Workload. We use a common multi-tier web benchmark Rice University Bulletin Board System (RUBBoS) [1] to investigate the performance of VMbuddies. RUBBoS simulates an online news forum like slashdot.org. It is a typical open-sourced benchmark that is widely used in academic communities. We can freely modify the workload generator and report more statistical information as we need. We deployed the benchmark in a 3-tier architecture using Apache 2.2, Tomcat 7.0, and MySQL 5.5. We deploy the benchmark with VMs in $m/n/o$ configuration for the three tiers, however, we only need to focus on the results of $1/1/1$ configuration. This is because the result is representative for a general $m/n/o$ configuration through VM grouping. In our experiments, each tier runs within a single VM. To generate workload in the servers, some other servers are used to run client workload generators at our lab. We simulate three typical workload patterns [3]:

- Static and Light Workload (S& L): 200 concurrent users access the website in a stable arriving rate, as shown in Fig. 10a.
- Dynamic and Heavy Workload (D&H): 500 and 1000 concurrent users alternately access the website for 30 seconds so as to simulate a cyclical on-off workload pattern, as shown in Fig. 10b.
- Light workload with Symmetric Waves pattern (L&SW): This workload pattern consists of four-minute windows. In each window, we first increase the concurrent users from 10 to 350 evenly, and then we decrease the users inversely. We repeat such windows to simulate a wave workload in which the memory dirtying rate changes slowly over time, as shown in Fig. 10c. The workload shows almost equivalent average memory dirtying rate compared to the S&L. This pattern simulates the workload fluctuations in real world (e.g., due to daily changes).

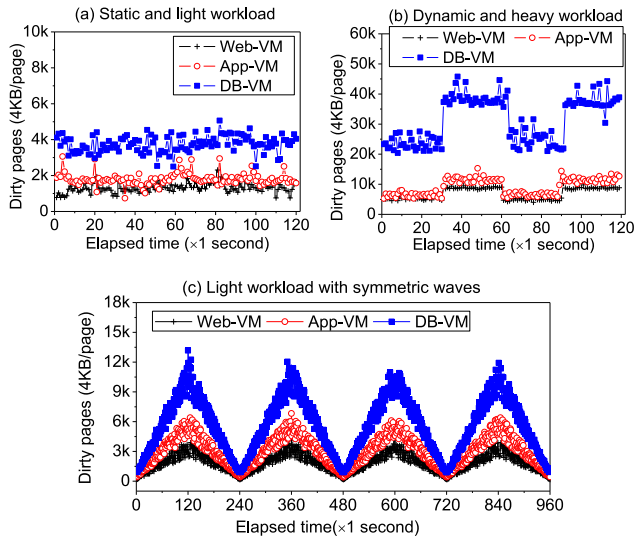

Fig. 10. The number of dirty pages observed for three types of RUBBoS workload.

Methodology. We evaluate the migration performance of RUBBoS by comparing the following approaches:

- *No Synchronization (NS)*. The multiple VMs are migrated sequentially using default Xen migration algorithm without synchronization. This is the baseline approach.
- *Migration Optimizations (MO)*. The multiple VMs are migrated one by one without synchronization, using optimized Xen migration algorithm (Section 4.2).
- *Largest memory Load VM Last scheduling*. The simple scheduling of VM migrations based on synchronization protocol (Section 3.2).
- *Optimal Bandwidth Allocation (OBA)* in Sections 3.3 and Sections 3.4.
- *LLL+MO, OBA+MO*. The multiple VMs are migrated using MO combining with LLL and OBA algorithm, respectively. *OBA+MO is the algorithm implemented in VMbuddies*.

We choose NS as the baseline because it has no synchronization costs and the migration performance is the best so that we can clearly study synchronization overheads caused by LLL and OBA. When the VMs are migrated in WAN testbeds, we first replicate the VMs' disk images to the destination before the memory migration begins. The results reported below are the average of the three trials (except failed migrations) and refer to the cost of memory migration and disk synchronization during VM live migrations.

5.2 Memory Dirty Rate and WWS Measurement

The shadow model of Xen allows us to track the number of dirty pages within any time window. We conducted an experiment to measure the memory dirtying rate of each tier when RUBBoS runs D&H, S&L and L&SW workloads. For each VM, we read the dirty bitmap and then clean it every one second. Fig. 10 plots the number of dirty pages generated by three workloads in different time windows. The x-axis shows elapsed time and the y-axis measures the number of pages dirtied within one second. The memory dirtying rate of each tier retains at relative low level of

TABLE 2
Measurement of Writable Working Set

Workloads	Writable Working Set (MB)		
	Web-VM	App-VM	DB-VM
S&L	63	96	502
D&H	166	212	818
L&SW	92	134	616

fluctuation for S&L workload. In contrast, the fluctuation is significant in terms of phase changes as the load changes for D&H workload. Moreover, the phase changes well align with each tier when workload changes. This result justifies our design on characterizing the memory access pattern of each VM by observing average memory dirtying rates periodically. Also, VMs in different tiers demonstrate different memory dirty rates, indicating different requirements of network bandwidth during migration.

To measure the VMs' writable working set, we first clean the dirty bitmap and then sample it every one second periodically. Note that the dirty bitmap was never cleaned so that the sampled number of each peek was the accumulation of dirty pages from the beginning of the test. If the page numbers in two consecutive peek are equal, this indicates that the writable working set becomes stable. Table 2 also shows the writable working set of different tiers of S&L, D&H and L&SW workloads in four minutes. Note that WWS denotes active memory pages in a VM so it is usually smaller than the VM's working set. The size of writable working set implies the opportunity of network traffic reduction by using memory ballooning.

5.3 Service Level Degradation

We first consider a hot spot mitigation scenario where the RUBBoS application with D&H workload must be migrated from an overloaded data center at NTU to another at NUS. We place the application to light-load servers and quadrupled each VM's CPU and main memory allocation once the VMs resume at the new data center.

Fig. 11 shows how the performance of RUBBoS web site is affected when the VMs were migrated. For MO (without synchronization), during the pre-copying phase of web-tier, the

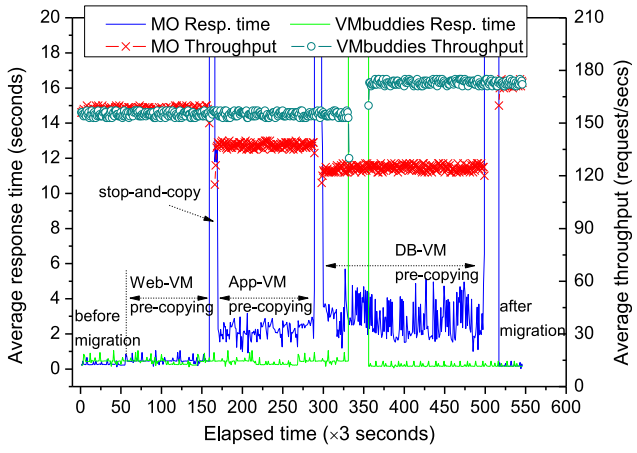

Fig. 11. The performance of RUBBoS application using D&H work-load when it is migrated in a real WAN testbed.

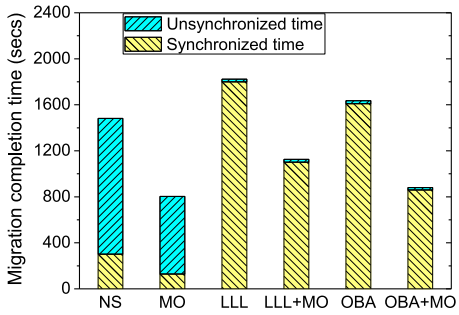

Fig. 12. The synchronized and unsynchronized time measured when the D&H workload is migrated in a real WAN testbed.

VM is still running at the source data center, and a slight performance degradation is observed. However, once the web-VM is moved to the destination data center and App-VM migration begins, the average response time significantly increases to 2.2 seconds compared to the prior 0.49 second. This degradation aggravates during the migration of DB-VM. The average response time raises 4.8X to 2.85 seconds, and the average throughput drops from 155 req/sec to 126 req/sec, nearly 18 percent throughput loss. The reason behind this observation is that the co-located service is decoupled in two data centers during the migrations, and the migration process also contends against the application for network bandwidth. In contrast, VMbuddies guarantee the correlated VMs always stay in the same data center, no matter before migration or after migration. The application only suffers 30 percent increase of response time and 1 percent loss of throughput due to VM migration overhead. VMbuddies reduces the average response time by 77 percent and improve 18 percent throughput compared to MO. When the VM migrations complete, the average response time drops to 0.13 second and the average throughput raises to 175 req/sec in both MO and VMbuddies because the capacity of the VMs increases by a factor of 4.

As VMbuddies is designed to mitigate application performance degradation caused by splitting VMs between data centers, it's entential to measure the synchronized and unsynchronized periods during VM migrations. As shown in Fig. 12, although sequential migration approaches including NS and MO show relatively shorter migration completion time, they suffer long-term unsynchrionized periods when App-VM and web-VM are migrated. The application performance is significantly degraded during this period, as shown in Fig. 11. In contrast, LLL, OBA and their optimizations can guarantee the application performance through synchronization protocol during VM migrations, at the cost of reasonable increase of migration completion times. Compared to NS, VMbuddies (OBA+MO) reduces the migration completion time from 1,478 seconds to 880 seconds, a 40 percent reduction of time that the application suffer from service level degradation during VM migrations.

5.4 Migration Cost

As a result of the performance penalty during the migration, it is important to minimize the migration completion time in order to reduce this period of lower performance. Fig. 13 shows the migration completion times of D&H and S&L workloads using different migration approaches in real

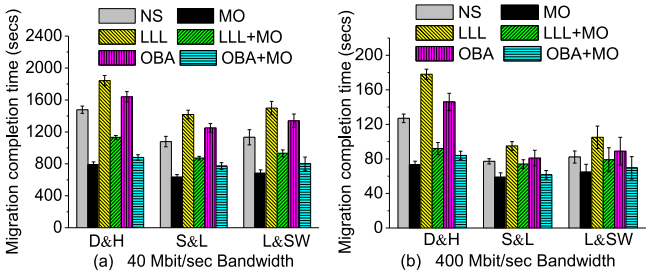

Fig. 13. The migration completion time of D&H, S&L and L&SW workloads over networks with 40 and 400 Mbit/sec bandwidth.

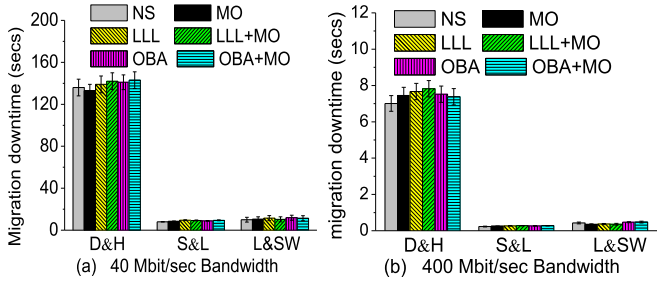

Fig. 15. Total migration downtime of migrating D&H, S&L and L&SW workloads over networks with 40 and 400 Mbit/sec bandwidth.

WAN and in-lab testbeds. The maximum available network throughput in the in-lab testbed is 400 Mbit/sec, representing a high-speed network. Fig. 14 shows the corresponding network traffic generated during the migrations and Fig. 15 shows the total migration downtime.

We have the following observations: (1) OBA causes less synchronization cost than LLL. Specifically for D&H workload on high-speed network, OBA can reduce 18 percent total migration completion time and 16 percent network traffic compared to LLL. It nearly avoid application performance degradation during migration while only experiencing 10 percent increase of migration completion time compared to NS; (2) By using MO strategies, all migration approaches show significant reduction of migration completion time and network traffic. The reduction is more significant for memory intensive workloads or low-speed network environments. For D&H workload on the real WAN testbed, it can reduce the migration cost by 47 percent . In summary, VMbuddies reduces the migration cost by 36 percent on average compared to NS. (3) For the same migration approach, S&L workload shows much less migration cost than D&H workload in both high-speed and low-speed networks. The reason is that higher memory dirtying rate in D&H results in much more data to be sent in each round of pre-copying; (4) Although the average memory dirtying rate of L&SW workload is almost equal to that of S&L, L&SW shows a little higher migration costs and higher standard deviations than S&L due to errors of workload pattern predictions and overhead of frequent bandwidth re-allocation. (5) LLL algorithm always causes higher migration cost than OBA algorithms as it causes higher synchronization cost for the web-VM and App-VM migrations; (6) The lower speed network implies more network traffic, as shown in Fig. 13. As a result, the reduction of migration completion time is not linearly proportional to the increase of network bandwidth; (7) All migration approaches show slight difference of total migration downtime between each other. Although VMbuddies leads to less than 4 percent increases of total migration downtime compared to NS, it avoids the significant application performance degradation during live migration of correlated VMs.

We further examine the log of live migration in NS and VMbuddies. In NS, the live migration of DB-VM always generates more than 3 GB network traffic for D&H workloads whether it is migrated over 40 or 400 Mbps networks. The reason is that the migration process always terminates the pre-copying phase only when the network traffic already exceeds 3 times of its memory size. Similarly, the migration process of App-VM terminates the pre-copying always after 30 rounds, but most of iterations are unnecessary, simply causing more network traffic and migration time. In VMbuddies, all VMs accelerate the progress of live migration through our intelligently terminating the iterative pre-copying. For S&L workload migrated over the network with 400 Mbps bandwidth, we find that NS works well because the memory dirtied in these VMs relatively slowly. In this case, the ballooning mechanism takes effect on the migration cost reduction because the VM shows smaller working set when it experiences light memory load.

5.5 Algorithms Accuracy and Runtime Overhead

We study the workload modeling errors and algorithm overhead by comparing L&SW with S&L, as shown in Fig. 16. We use S&L as a baseline and illustrate the normalized migration cost of L&SW, both are measured in the real WAN environment using "OBA + MO" approach. L&SW show reasonable higher migration cost than S&L due to runtime overhead and model errors. When the time window of memory dirtying rate prediction increases, the frequency of

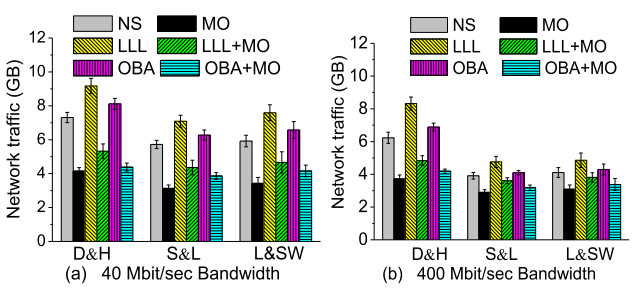

Fig. 14. Network traffic of migrating D&H, S&L and L&SW workloads over networks with 40 and 400 Mbit/sec bandwidth.

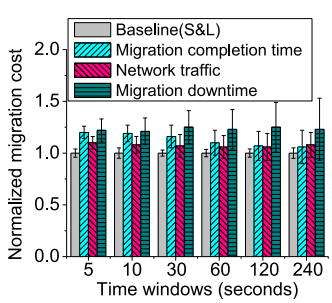

Fig. 16. The normalized migration cost of L&SW compared to S&L.

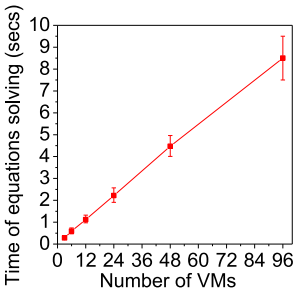

Fig. 17. The compute over-head of Algorithm 2.

bandwidth re-allocation decreases and thus the migration completion time is reduced. The network traffic is not sensitive to the changes of time window because live migration of VMs usually last long in WANs and positive/negative predication errors counteract each other. However, migration downtime show larger standard deviations when the time window increases. This is because the stop-and-copy phases are performed at different time windows and a larger window implies higher prediction error.

We further study the runtime overhead of Algorithm 2 on a host machine configured with 2 CPU and 4 GB memory. We exponentially increase the number of VMs from 3 to 96, i.e., number of equations to be solved. Fig. 17 shows that the compute cost of Algorithm2 is linearly proportional to the number of VMs. This demonstrates that VMbuddies cause acceptable runtime overhead. In practice, we never need to solve a large number of equations because our VM grouping scheme guarantees only a small set of VMs is migrated concurrently.

In the end, we summarize the evaluation results of different migration approaches. MO always shows the lowest cost of VM migrations, however, it cost significant application performance degradation due to splitting VMs in different data centers. LLL and VMbuddies (OBA+MO) both can significantly reduce application performance degradation through our synchronization protocol. However, LLL results in much more synchronization cost than that of VMbuddies. VMbuddies guarantee the application performance during migration, at the cost of slight increase of migration cost.

6 Related Work

We present the related work in the following three categories: single-VM migrations, multiple VM migrations and multi-tier application migration in the cloud.

Single-VM Live migration over LANs or WANs. There have been many approaches proposed for VM live migration, such as pre-copying [10], [30], post-copying [16], full-system log/replay [24]. Additionally, there have been a number of optimizations to further reduce the overhead of pre-copying algorithm, including memory compression [20], page de-duplication [12], [14], [38], partial migration [5], parallelizing migration [33], shared storage accelerating [22] and others [19]. Those optimizations are also applicable to VMbuddies. Unlike VM migrations in LANs where the disk is shared and network connections are sustained, the disk and network connections need special care in WANs. For disks, block-level disk pre-copying and write throttling mechanisms was proposed to migrate the persistent storage [8], [42]. Mashtizadeh et al. evaluated snapshotting, dirty block tracking and IO mirroring for storage migration supported by VMware ESX [29]. For network connections, VPN [38], IP tunneling and dynDNS [8] were proposed to guarantee the network switching transparent to VM applications. VM Turntable demonstrated VM live migration over WAN using dedicated optical links for Grid computing [34]. CloudNet built a private network using MPLS-based VPN for VM live WAN migration [38]. Xen-Blanket provided an abstract layer on top of Xen to support VM live migration across multiple cloud providers [37]. These works all focused on migrating a single VM in LANs or over WANs. None has considered the problem of correlated VM migration in multi-tiered applications across distributed data centers.

Multi-VM Migration. There have been limited proposals on reducing the cost of multiple concurrent VM migrations. Deshpande et al. investigated live gang migration of VMs in LAN environments [14]. They proposed page and sub-page level memory de-duplication among co-located VMs and compression strategies to optimize memory migration of multiple VMs. As the network bandwidth is sufficiently high in LANs, their work did not consider the correlated VM migration problem. Al-Kiswany et al. were also motivated by data de-duplication, and proposed VMFlocks to optimize cross-datacenter transfer and instantiation of disk images of multiple correlated VMs [4]. Their work focused on exploiting the similarity among the VM images and didn't consider the memory live migration. These works are orthogonal to the theme of this paper. They focus on reducing the data transfer by de-duplication or fine-grained data sharing. In contrast, VMbuddies employ network bandwidth allocation strategies to coordinate the correlated VM migrations so that the total synchronization cost is minimized.

Multi-tier Application Migration in the Cloud. A number of studies had investigated the live migration performance of multi-tier applications for both intra- and inter-datacenter scenarios. Voorsluys et al. [35] evaluated the performance degradation of 2-tier web 2.0 applications running within VMs. The similar evaluations were conducted in [17], [23]. Their experimental results all showed significant performance degradation in terms of request response time, even when the multi-tier applications are migrated in the same data center. While those studies had given preliminary results on the performance problem of correlated VM migrations in a multi-tier application, they had not formulated or solved the problem. Recently, Zheng et al. proposed Pacer [43], a progress management system for VM migration in the cloud. Pacer controls VM migration completion time based on analytic models of progress prediction and online adaptation. Pacer is perhaps the most similar work with VMbuddies. However, Pacer focuses on progress management of single-VM migration, and the evaluation result on correlated VM migrations is very limited. This paper concentrates on the correlated VM migration problem. We formulate it as a DCOP problem, and propose an efficient algorithm to solve it. The evaluation studies provide more insight on live migration of complex applications.

7 Conclusion

As more and more multi-tier applications are hosted in virtualization environments across geographically distributed data centers, the efficiency of live migration of VMs in a multi-tier application becomes important. In this paper, we formulate the VMs live migration of multi-tier applications as a correlated VM migrations problem, and identify the low efficiency of the basic approach in current virtualization platforms such as Xen. Therefore, we propose a coordination system called VMbuddies for correlated VM migrations. To minimize the migration cost, VMbuddies coordinates VM migrations with a synchronization protocol and an optimal network bandwidth allocation algorithm. Experiments using a public benchmark show that VMbuddies achieves lower average response time by 77 percent and higher throughput by 18 percent, and reduces average total migration cost by 36 percent in comparison with Xen.

We also note that the performance models of VM live migration would facilitate data center administrators in many scenarios, such as, scheduling long-term VM migration with deadline constraints, making tradeoffs amongst different migration performance metrics through network resource control, making decision for grouping and ordering correlated VMs to reduce migration cost. In our future work, we plan to explore VMBuddies for more complicated scenarios, such as workflow [40], MapReduce and MPI applications that have tree or directed acyclic graph topologies.

Acknowledgments

This work was supported by the Singapore National Research Foundation under its Environmental & Water Technologies Strategic Research Programmes and administered by the Environment & Water Industry Programme Office (EWI), under project 1002-IRIS-09. The work of Haikun Liu was done when he visited Nanyang Technological University.

References

[1] RUBBoS, (2005) [Online]. Available: http://jmob.ow2.org/rubbos.html

[2] Secant Method, (2003) [Online]. Available: http://math.fullerton.edu/mathews/n2003/SecantMethodMod.html

[3] Cloud Workload Patterns, (2010) [Online]. Available: http://watdenkt.veenhof.nu/2010/07/13/workload-patterns-for-cloud-computing

[4] S. Al-Kiswany, D. Subhraveti, P. Sarkar, and M. Ripeanu, "VMFlock: Virtual machine co-migration for the cloud," in *Proc. ACM Symp. High Perform. Parallel Distrib. Comput.*, Jun. 2011, pp. 159–170.

[5] N. Bila, E. Lara, K. Joshi, H. A. Lagar-Cavilla, M. Hiltunen, and M. Satyanarayanan, "Jettison: Efficient idle desktop consolidation with partial VM migration," in *Proc. ACM Eur. Conf. Comput. Syst.*, Apr. 2012, pp. 211–224.

[6] 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 *Proc. ACM Symp. Operating Syst. Principles*, Oct. 2003, pp. 164–177.

[7] J. Billaud and A. Gulati, "hClock: Hierarchical QoS for packet scheduling in a hypervisor," in *Proc. ACM Eur. Conf. Comput. Syst.*, Apr. 2013, pp. 309–322.

[8] R. Bradford, E. Kotsovinos, A. Feldmann, and H. Schioeberg, "Live wide-area migration of virtual machines including local persistent state," in *Proc. Third Conf. Virtual Execution Environ.*, Jun. 2007, pp. 169–179.

[9] H. Ballani, K. Jang, T. Karagiannis, C. Kim, D. Gunawardena, and G. O'Shea, "Chatty tenants and the cloud network sharing problem," in *Proc. 10th USENIX Symp. Netw. Syst. Des. Implementation*, Apr. 2013, pp. 171–184.

[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 *Proc. 2nd Symp. Netw. Syst. Des. Implementation*, May 2005, pp. 273–286.

[11] C. Chen, B. He, and X. Tang, "Green-aware workload scheduling in geographically distributed data centers," in *Proc. IEEE 4th Cloud Comput. Technol. Sci.*, 2012, pp. 82–89.

[12] J. Chiang, H. Li, and T. Chiueh, "Introspection-based memory deduplication and migration," in *Proc. ACM Int. Conf. Virtual Execution Environ.*, Mar. 2013, pp. 51–61.

[13] G. Daniel, R. Jerry, C. Ludmila, and K. Alfons, "Workload analysis and demand prediction of enterprise data center applications," in *Proc. IEEE 10th Symp. Workload Characterization*, Sep. 2007, pp. 171–180.

[14] U. Deshpande, X. Wang, and K. Gopalan, "Live gang migration of virtual machines," in *Proc. Int. Symp. High Perform. Distrib. Comput.*, Jun. 2011, pp. 135–146.

[15] C. M. Fiduccia and R. M. Mattheyses, "A linear-time heuristic for improving network partitions," in *Proc. 19th Des. Autom. Conf.*, Jun. 1982, pp. 175–181.

[16] M. Hines and K. Gopalan, "Post-copy based live virtual machine migration using adaptive pre-paging and dynamic self-ballooning," in *Proc. ACM Conf. Virtual Execution Environ.*, Mar. 2009, pp. 51–60.

[17] H. Hlavacs and T. Treutner, "Predicting web service levels during VM live migrations," in *Proc. DMTF Workshop Syst. Virtualization Manage.*, Oct. 2011, pp. 1–10.

[18] D. Huang, B. He, and C. Miao, "A Survey of resource management in multi-tier web applications," *IEEE Commun. Surveys Tutorials*, vol. PP, no. 99, pp. 1–17, Jan. 2014, DOI: 10.1109/SURV.2014.010814.00060

[19] K. Z. Ibrahim, S. Hofmeyr, C. Iancu, and E. Roman, "Optimized pre-copy live migration for memory intensive applications," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, Nov. 2011, pp. 1–11.

[20] H. Jin, L. Deng, S. Wu, X. Shi, and X. Pan, "Live virtual machine migration with adaptive memory compression," in *Proc. IEEE Conf. Cluster Comput.*, Aug. 2009, pp. 13–22.

[21] V. Jeyakumar, M. Alizadeh, D. Mazi'eres, B. Prabhakar, C. Kim, and A. Greenberg, "EyeQ: Practical network performance isolation at the edge," in *Proc. 10th USENIX Symp. Netw. Syst. Des. Implementation*, Apr. 2013, pp. 297–311.

[22] C. Jo, E. Gustafsson, J. Son, and B. Egger, "Efficient live migration of virtual machines using shared storage," in *Proc. ACM Conf. Virtual Execution Environ.*, Mar. 2013, pp. 41–50.

[23] S. Kikuchi and Y. Matsumoto, "Impact of live migration on multi-tier application performance in clouds," in *Proc. IEEE 5th Conf. Cloud Comput.*, Jun. 2012, pp. 261–268.

[24] H. Liu, H. Jin, X. Liao, L. Hu, and C. Yu, "Live migration of virtual machine based on full system trace and replay," in *Proc. 18th Int. Symp. High Perform. Distrib. Comput.*, Jun. 2009, pp. 101–110.

[25] H. Liu, H. Jin, X. Liao, C. Yu, and C. Xu, "Live virtual machine migration via asynchronous replication and state synchronization," *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, no. 12, pp. 1986–1999, Dec. 2011.

[26] H. Liu, C. Xu, H. Jin, J. Gong, and X. Liao, "Performance and energy modeling for live migration of virtual machine," in *Proc. ACM Int. Symp. High Perform. Parallel Distrib. Comput.*, Jun. 2011, pp. 171–182.

[27] K. Z. Meth and J. Satran, "Design of the iSCSI protocol," in *Proc. IEEE Symp. Mass Storage Syst.*, Apr. 2003, pp. 116–122.

[28] V. Mann, A. Gupta, P. Dutta, A. Vishnoi, P. Bhattacharya, R. Poddar, and A. Iyer, "Remedy: Network-aware steady state VM management for data centers," in *Proc. 11th IFIP Conf. Netw.*, May 2012, pp. 190–204.

[29] A. Mashtizadeh, E. Celebi, T. Garfinkel, and M. Cai, "The design and evolution of live storage migration in VMware ESX," in *Proc. USENIX Ann. Tech. Conf.*, Jun. 2011, pp. 1–14.

[30] M. Nelson, B. H. Lim, and G. Hutchins, "Fast transparent migration for virtual machines," in *Proc. USENIX Ann. Tech. Conf.*, Apr. 2005, pp. 391–394.
[31] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, "Root finding and nonlinear sets of equations importance sampling," in *Numerical Recipes: The Art of Scientific Computing*. 3rd ed., New York, NY, USA: Cambridge Univ. Press, 2007, ch. 9.
[32] A. Qureshi, R. Weber, and H. Balakrishnan et al., "Cutting the electric bill for internet-scale systems," in *Proc. ACM Conf. Data Commun.*, Aug. 2009, pp. 123–134.
[33] X. Song, J. Shi, R. Liu, J. Yang, and H. Chen, "Parallelizing live migration of virtual machines," in *Proc. ACM Int. Conf. Virtual Execution Environ.*, Mar. 2013, pp. 85–95.
[34] F. Travostino, P. Daspit, L. Gommans, C. Jog, C. de Laat, J. Mambretti, I. Monga, B. van Oudenaarde, S. Raghunath, and P. Wang, "Seamless live migration of virtual machines over the MAN/WAN," *Future Generations Comput. Syst.*, vol. 22, no. 8, pp. 901–907, Oct. 2006.
[35] W. Voorsluys, J. Broberg, S. Venugopal, and R. Buyya, "Cost of virtual machine live migration in clouds: A performance evaluation," in *Proc. Int. Conf. Cloud Comput.*, Dec. 2009, pp. 254–265.
[36] C. A. Waldspurger, "Memory resource management in VMware ESX server," in *Proc. 5th Symp. Operating Syst. Des. Implementation*, Dec. 2002, pp. 181–194.
[37] D. Williams, H. Jamjoom, and H. Weatherspoon, "The xen-blanket: Virtualize once, run everywhere," in *Proc. 7th ACM Eur. Conf. Comput. Syst.*, Apr. 2012, pp. 113–126.
[38] T. Wood, P. Shenoy, K. K. Ramakrishnan, and J. V. Merwe, "CloudNet: Dynamic pooling of cloud resources by live wan migration of virtual machines," in *Proc. 7th Int. Conf. Virtual Execution Environ.*, Mar. 2011, pp. 121–132.
[39] D. Xie, N. Ding, Y. C. Hu, and R. Kompella, "The only constant is change: Incorporating time-varying network reservations in data centers," in *Proc. ACM Conf. Data Commun.*, Aug. 2012, pp. 199–210.
[40] A. Zhou and B. He, "Transformation-based monetary cost optimizations for workflows in the cloud," *IEEE Trans. Cloud Comput.*, vol. 2, no. 1, pp. 85–98, Mar. 2014.
[41] H. Zheng and X. Tang, "On server provisioning for distributed interactive applications," in *Proc. 33rd Int. Conf. Distrib. Comput. Syst.*, Jul. 2013, pp. 500–509.
[42] J. Zheng, T. S. Eugene Ng, and K. Sripanidkulchai, "Workload-aware live storage migration for clouds," in *Proc. 7th Int. Conf. Virtual Execution Environ.*, Mar. 2011, pp. 133–144.
[43] J. Zheng, T. S. Eugene Ng, K. Sripanidkulchai, and Z. Liu, "Pacer: A progress management system for live virtual machine migration in cloud computing," *IEEE Trans. Netw. Serv. Manage.*, vol. 10, no. 4, pp. 369–382, Dec. 2013.
[44] R. Zivan and H. Peled, "Max/min-sum distributed constraint optimization through value propagation on an alternating DAG," in *Proc. 11th Int. Conf. Auton. Agents Multi-Agent Syst.*, Jun. 2012, pp. 265–272.

Dr. Haikun Liu received the PhD degree from the Huazhong University of Science & Technology (2007-2012). He is currently a research fellow at the School of Computer Engineering, Nanyang Technological University. His current research interests include virtualization technologies, cloud computing, and distributed systems.

Bingsheng He received the bachelor's degree in computer science from Shanghai Jiao Tong University (1999-2003), and the PhD degree in computer science from the Hong Kong University of Science and Technology (2003-2008). He is currently an assistant professor in the Division of Networks and Distributed Systems, School of Computer Engineering of Nanyang Technological University, Singapore. His research interests are high performance computing, cloud computing, and database systems. He has been awarded with the IBM PhD fellowship (2007-2008) and with NVIDIA Academic Partnership (2010-2011).