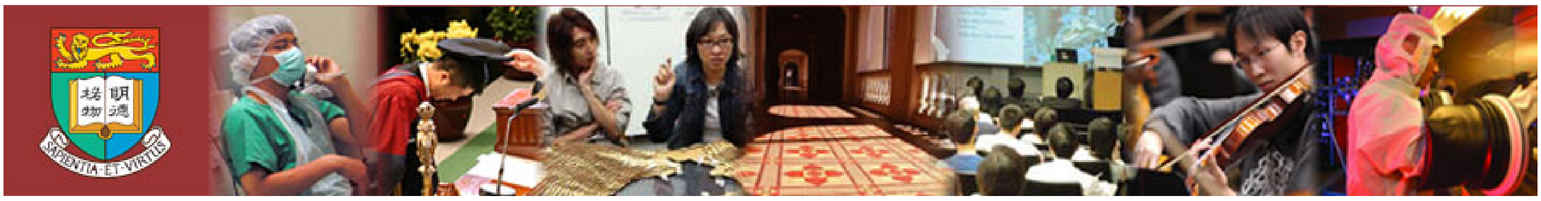


Title	Adaptive live VM migration over a WAN: modeling and implementation
Author(s)	Zhang, W; Lam, KT; Wang, CL
Citation	The 7th IEEE International Conference on Cloud Computing (CLOUD 2014), Anchorage, AK., 27 June-2 July 2014. In Conference Proceedings, 2014, p. 1-8
Issued Date	2014
URL	http://hdl.handle.net/10722/203646
Rights	IEEE International Conference on Cloud Computing (CLOUD). Copyright © IEEE Computer Society.

Adaptive Live VM Migration over a WAN: Modeling and Implementation

Weida Zhang, King Tin Lam, Cho-Li Wang
Department of Computer Science
The University of Hong Kong
Hong Kong, China
{wdzhang, ktlam, clwang}@cs.hku.hk

***Abstract*—Recent advances in virtualization technology enable high mobility of virtual machines (VMs) and resource provisioning at a data-center level. Various strategies have been proposed for fast VM live migration over a local-area network (LAN). The most common solution uses memory pre-copying and assumes storage is shared on the LAN. When applied to a wide-area network (WAN), a new design philosophy in VM live migration algorithms is necessary to address like challenges of long latency, limited or unstable bandwidth and storage relocation. This paper proposes a three-phase, fractional, hybrid pre-copy and post-copy solution for both memory and storage to achieve highly adaptive and responsive WAN-wide migration. Our strategy is to selectively migrate an important fraction of memory and storage in the pre-copy and freeze-and-copy phases, while the rest (non-critical data set) is post-copied or demand-paged. We propose a new metric called performance restoration agility, which considers both the downtime and VM speed degradation during the post-copy phase, to evaluate the migration process. We also develop a profiling framework and a novel probabilistic prediction model to adaptively find a predictably optimal combination of the memory and storage fractions to migrate. Our solution is implemented on Xen and evaluated in an emulated WAN environment. Experimental results show that the solution achieves better adaptiveness than others for various applications over a WAN while retaining the responsiveness of post-copy algorithms.**

Keywords—cloud computing; virtualization; live migration; hybird-copy; wide-area networks; performance modeling; Xen

I. Introduction

Cloud computing is emerging as an important paradigm shift in how computing demands are being met in future. It is transforming the role of IT in businesses in recent years. With cloud architecture, computing resources can be rapidly provisioned or scaled-out by live virtual machine (VM) migration with minimal management effort or service provider interaction [1]. Such elastic infrastructure marks the beauty of cloud computing for enhancing IT delivery's efficiency and cost-effectiveness. With VM migration technology, one can pool or shrink compute resources as desired by moving VM instances around over a cluster or even a wide area network (WAN), facilitating dynamic load distribution, fault resilience, and improved system administration (e.g. server consolidation). The importance of VM mobility is evidenced by paramount work like VMWare vMotion [2] and Xen [3].

Most VM systems implement *live migration* which moves the memory image while the VM is still running to minimize downtime. By *pre-copying* memory pages and iteratively copying dirty pages, Xen and vMotion achieve sub-second downtime for non-write-intensive applications, albeit with tens of seconds of total migration time. Other solutions [4] adopt a *post-copy* strategy that defers memory transfer until after the VM's CPU state gets resumed on the target. Essentially, post-copying ensures that each memory page is transferred at most once, thus avoiding duplicate transmission overhead of pre-copying. Whichever strategy is used, it will penalize the normal execution for a substantial period throughout the migration process. Common solutions assume the disk image is put on some shared networked storage (SAN/NAS) so that it can be immediately accessible to all destination hosts. VM migration also hinges on the assumption that the hosts are in a common network segment for seamless migration of network connections. Thus, the uses of VM migration have been largely restricted to local area networks (LANs).

Nevertheless, *wide-area VM migration*, which is parallel to the vision of global resource scheduling and disaster recovery, has been gaining interest and adoption. Several projects—Shrinker [5], VM Turntable [6] and ABSS [7]—have been steering towards this goal. They have to cope with challenges like how to maintain existing network connections despite a change in the VM's IP address [8], and how to transfer the persistent state of the big local storage efficiently over long-haul networks with bandwidth bottlenecks. Pure pre- or post-copy and basic hybrid-copy strategies have been proposed for WAN situations. However, the proposed solutions are usually inflexible and perform poorly in the changing network environment. Due to much longer latency and limited bandwidth of a WAN, a pure pre-copy algorithm could spend longer time on repeatedly transferring some *dirtied* pages which turn out to be unneeded by the VM resumed on the remote site. Worst still, it fails to converge to zero downtime if the dirtying rate outruns the link bandwidth. Therefore, pure pre-copying on a WAN could result in long migration time as well as long downtime. In some cases, the pre-copy iterations are repeated even when the network performance has fallen behind the memory dirtying rate. Thus, a vanilla pre-copy solution (e.g. [2], [3]) is usually hard coded with ending conditions to force migration at a predetermined stage to avoid generating additional dirty pages and unnecessary network traffic. Tuning the ending condition however does not improve the performance of pre-copy algorithms in many situations (e.g. write-intensive workloads) [9]. A pure post-copy approach in bad network conditions performs poorly upon resumption due to the large amount of remote data access after the VM resumes execution at the destination. Therefore, full-system post-copying was researched restrictively on high-speed LANs only [4]. Hybrid-copy approaches have been proposed to reduce the downtime and remote uptime. But the way they hybridize is rather static, e.g. fully pre-copy memory and fully post-copy storage.

By studying the limitations of wide-area VM migration, this work advances the state of the art to a new level—a profile-guided *fractional* hybrid-copy strategy—so as to achieve better migration performance in various aspects, particularly *adaptiveness* and *responsiveness*, in a WAN situation. While existing solutions adopt WAN-wide post-copying for storage but not for memory, we do apply pre- and post-copying to both memory and storage, yet in a fractional manner. Our proposed hybrid solution relies on two fractional variables M and S, representing the critical portions of memory and storage images to be migrated during the pre-copy and freeze-and-copy phases, while the remaining are migrated in the post-copy phase. By adjusting M and S, we can achieve adaptiveness in response to the network conditions and application behaviors. The contributions of this work are elaborated as follows.

- We investigate the algebraic relation between *total migration time* (T), *downtime* (D), *remote uptime* (U), *performance degradation* (P_D) and other parameters, such as dirtying rate of memory/storage and bandwidth to make a thorough performance model. In particular, we propose a new metric—*performance restoration agility* (Γ)—as a quantitative measurement of how fast the performance gets restored to a nearly full-speed level under different M and S settings. The definition of Γ has embedded D and P_D[1] which are in some inverse relationship. In essence, Γ models the wrestling between pre-copy and post-copy effects which favor P_D and D respectively.
- This is the first work to propose and analyze a fractional, hybrid (pre/post-copy) approach to WAN-wide live migration of both memory (M) and storage (S). We see that finding the best portion of (M,S) that achieves an agile migration while still minimizing the degradation during the post-migration execution is an important research gap to fill, and we provided a non-trivial prediction model and its implementation at page level on Xen accordingly. We devise a profiling-based and model-based solution to determine M and S that are adaptive to the underlying network condition and the application's memory/storage access patterns. They could be carefully determined to ensure Γ is above a certain level. This can achieve reasonable performance even when vanilla pre-copy algorithms cannot converge to small downtime, and can counteract degradation during the post-copy phase.

The motivation behind this work is an attempt to make WAN-wide live migration more useful in real life. For instance, one may need to continue some work during the travel time from office to home. Then a mobile working environment could be enabled by migrating a VM instance from his office PC to his laptop, providing the same view of software applications as if he were working in office. Though a fully post-copy scheme gives the shortest remote uptime U that allows his work to resume as soon as possible, it entails the longest low-performance period after the VM gets resumed. A balance between the data amounts to pre-copy and post-copy is thus important to let the user perceive an acceptable VM performance after the migration. So our solution is useful here, by trading off some access profiling overheads and slightly longer downtime for better post-copy performance.

For the rest of this paper, we review the existing solutions in Section II. Section III and Section IV present the modeling and implementation of our proposed fractional hybrid-copy live VM migration for a WAN, respectively. In section V, we evaluate the performance of Xen with our model-based solution implemented. Section VI concludes this paper.

II. Literature Review

Pre-copy migration strategies [2], [3] aim at reducing downtime. In a LAN environment, only memory is to be migrated and is transmitted in an iterative way. In the first iteration, all the memory pages are transmitted. In each subsequent iteration, all the memory pages that are dirtied during the previous iteration are transmitted, assuming that the number of dirtied pages in each subsequent iteration will decrease and converge to a small amount at some point.

Clark *et al.* [3] characterize the downtime of pre-copy migration by the *writable working set* (WWS) of the application. The WWS is a set of pages being frequently updated to an extent that it is unwise to transmit them before the last iteration since they will be updated again in every short period, and any previous transmissions would be wasted. However, Clark *et al.* did not analyze the relationship between WWS and the network condition. A later piece of work by Akoush *et al.* [10] provides simulation models that can predict the total migration time and downtime based on the pre-copy scheme. They conclude that the ratio between the page dirtying rate and the link speed has a strong impact on VM migration behaviors. They model the page dirtying rate with two approaches: AVG models it as an average, and HIST models it as a history log. They run simulations over the dirtying rate model to predict the downtime and total migration time of a migration. We find that the AVG model is too simple while HIST is too heavyweight for our usage since we are going to run the simulation for many times. In this work, we model the dirtying rate as the probabilistic expected value of the access frequencies of every page within an observation window.

As far as we know, Bradford *et al.* [9] proposed the first work that accomplished VM migration over a WAN. Both memory and storage are migrated using pre-copying. They applied write-throttling to slow down some disk writes and decrease the dirtying rate so as to ensure convergence. However, disk write operations cannot tolerate aggressive throttling. For memory pre-copying, write-throttling is difficult to implement and with high overheads. Thus we have not seen any existing work on memory write-throttling. Recent work—ABSS [7]—done by Akoush *et al.* streamlines repeated transmission of storage blocks during the pre-copy stage. They analyze the storage access patterns to predict whether a data block is to be rewritten in near future. If it is the case, the system postpones

[1]Basically, P_D and D are relatively more important than T and U because they affect not only the experience perceived by the system administrator who triggers the migration but also the user of the VM.

the transmission of that block, thus reducing the number of blocks to retransmit and the total migration time.

Post-copy strategies transmit all the processor state to the destination and intend to resume the VM execution as fast as possible, while actively pushing the VM's memory pages from source to destination. Any memory pages that are faulted at the destination but not yet pushed, are demand-paged over the network. Post-copying thus ensures that each memory page is transferred at most once, thus avoiding the duplicate transmission overheads of pre-copying. Hines *et al.* [4] implemented a post-copy solution called *dynamic self-ballooning (DSB)* for live VM migration across a gigabit LAN. While the DSB technique works well on a LAN, there will be problems if we extend it for the hybrid migration on WANs. First, fetching of memory pages through the guest storage driver will in turn goes through the Xen's storage migration driver. This will introduce long latency. Second, getting the guest OS memory shrunk well before the pre-copy phase means performance degradation gets started sooner. To avoid these drawbacks, we implement another method, namely *page tracking*, proposed by Hines. Hirofuchi *et al.* [11] introduce post-copying for live storage migration over a WAN. They also propose background copying, which can transfer the blocks without affecting the performance of the migrated VM. Overall, post-copy migration can achieve small remote uptime and improve interactivity. It however cannot guarantee the performance of the migrated VM. As network speed is slower than local memory, a migrated VM could suffer severe degradation at the destination.

Noack [12] mentioned about a hybrid-copy algorithm composed of a single pre-copy round plus post-copying of dirty pages. Luo *et al.* [13] and Hirofuchi *et al.* [11] adopt pre-copying for memory pages and post-copying for storage. Zheng *et al.* [14] add data locality-based prediction to pre-copy, post-copy and hybrid-copy storage migrations over a WAN. To the best of our knowledge, this work is the first to propose a VM migration strategy using a fractional hybrid-copy approach for both memory and storage.

III. Modeling

A. Hybrid Migration Framework

As shown in Fig. 1, we divide the whole migration process into three phases, namely *pre-copy phase*, *freeze-and-copy phase* and *post-copy phase*. After the pre-copy and freeze-and-copy phases, the percentage of transferred memory image is M, while the percentage of transferred storage is S.

The hybrid migration scheme works as follows:

1) In the pre-copy phase, storage is pre-copied first.
2) Once S of storage is migrated, the system switches to move storage in background and starts memory pre-copying to move M of memory. However, if further iterations during the pre-copy phase are not expected to improve the situation (*i.e.*, page dirtying rate is greater than transmission rate), the system could choose to enter the next phase.
3) During the freeze-and-copy phase, we continue the migration until M of memory and S of storage are fully migrated.
4) In the post-copy phase, the remaining memory $(1-M)$ and storage $(1-S)$ continue to be migrated. When the $(1-M)$ memory migration is not finished, the storage migration only maintains a small background transmission rate.
5) Once the memory migration is finished, the storage migration is performed at full speed.

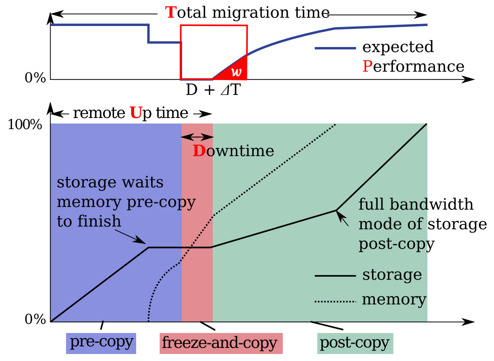


Fig. 1. Hybrid migration of memory and storage

We model T, D and U based on the work of Akoush *et al.* [10]. However, Akoush's prediction on dirtying rate is either too simple (AVG model) or too heavyweight (HIST model). Instead we propose prediction of the dirtying rate based on as few as ten samples per page. We model the *page dirtying rate* as a function $\texttt{dirty}(\tau)$, where τ could be any predicted length of a pre-copy iteration. To calculate the total migration time (T), we add U to the predicted time of the post-copy phase, during which the remaining memory $(1-M)$ and storage $(1-S)$ are transmitted. Since during the post-copy phase, a page or a block will not be transmitted twice, the total elapsed time is simply determined by the transmission rate.

The *performance restoration agility*, $\mathit{\Gamma}$, is defined as a ratio of the execution speed of the virtual machine with migration to that without migration. Intuitively, it symbolizes how fast the migrated VM regains the full speed since the VM suspension. Graphically, it refers to the ratio between the triangular area and the rectangular area as shown in Fig. 1 (the graph at the top). Suppose w is a constant workload, which takes the VM δT time to finish it at full processing speed (*i.e.* without migration). Suppose the full processing speed of the VM is a constant c, then $c = w/\delta T$. To accomplish the workload w during the VM migration process, the VM stops its service during the downtime D and can only resume its execution at sub-optimal speed during the first few seconds (ΔT) at the beginning of post-copy phase. It is likely that $\Delta T > \delta T$ since the performance is degraded while fetching memory or storage left on the source host. The workload completed at this stage equals the triangular area. The average speed of the VM during the period of $(D + \Delta T)$ is $w/(D + \Delta T)$. So we define $\mathit{\Gamma}$ as the ratio of these two average speeds:

$$\mathit{\Gamma} = \frac{\delta T}{D + \Delta T} \tag{1}$$

Using an automobile as a metaphor, we are measuring the speed of a broken car when we do not have a mileometer. So we first drive the car in good condition for δT and mark its distance traveled. Later, when the car is broken, we drive it for the same distance above, and measure the time elapsed as ΔT. Then we can understand the performance degradation of the car. δT could be any reasonable length of time considering the precision of the 'stopwatch' and 'distance marker'. In our experiment, we find 20 seconds a good choice. We also use this δT as the unit time for profiling and the model to simplify the algebraic calculation.

B. Data Access Modeling

The model of T, D, U and Γ relies on two intermediate variables $\texttt{dirty}(\tau)$ and ΔT. Predicting $\texttt{dirty}(\tau)$ is equivalent to telling how many pages are expected to be written within τ time. As for ΔT, its implication is about how many pages will be fetched on demand during the post-copy phase, thus penalizing the performance before a workload of w is completed. Both $\texttt{dirty}(\tau)$ and ΔT are modeled as the mathematical expectations of some random variables in the following statistical model.

We model the access to memory or storage as a Poisson distribution. Suppose X is a random variable suggesting how many times a page will be accessed during a period of time τ. Each page is associated with a parameter λ (the expected value and also variance of a Poisson distribution). Then the probability of accessing a page k times per unit time is

$$\Pr(X = k) = \frac{e^{-\lambda}\lambda^k}{k!} \tag{2}$$

Typically, we are interested in whether a page i will be accessed during any time period, *e.g.* τ. The probability of no access to a page within τ is $\Pr_{i,\tau}(X = 0) = e^{-\lambda\cdot\tau}(\lambda\tau)^0/(0!) = e^{-\lambda\cdot\tau}$.

C. Measurement of Frequency of Access

Before we can predict $\texttt{dirty}(\tau)$ and ΔT, we first need to estimate λ. This is done by profiling the access frequencies of each page/trunk of the memory and storage, and calculating the *maximum likelihood estimation (MLE)* of λ from the profiling samples.

1) Storage Profiling: The storage profiling is relatively easy. We simply extend the disk driver backend to track all the I/O requests from the VM without introducing high overheads. To minimize the overhead of storing profiling data, each trunk of 64 contiguous 512-byte blocks is the basic unit for profiling. Suppose we perform profiling for n time slots during which the profiler recorded that a trunk is accessed $k_1, k_2, \cdots, k_n$ times. Thus,

$$\hat{\lambda}_{\text{MLE}} = \frac{1}{n}\sum_{i=1}^{n} k_i = \frac{\Sigma k}{n} \tag{3}$$

where Σk is the abbreviated form of $\sum_{i=1}^{n} k_i$. In the following contexts, we use Σk to denote the sum of the samples of a storage trunk.

2) Memory Profiling: To obtain a more realistic cost model, our memory profiling involves a localized migration (source and destination being the same machine) with all the network protocol bypassed. We run the pre-copy and post-copy mechanisms during memory profiling to sample the memory access behaviors in a fast manner, inducing a small amount of memory copying and 'minor' page faults. A minor page fault refers to a fault that can be resolved quickly without swapping. Reads and writes are sampled separately: read operations are observed in post-copy phases while write operations in pre-copy phases. However, differing from storage profiling, it introduces high overheads to record every memory access. For both kinds of access, we can only observe if there is one access event during a period τ. But this measurement is sufficient to estimate λ by applying the following mathematics.

The probability of a Poisson distribution is defined in (2). So $\Pr(X = 0) = e^{-\lambda}$, $\Pr(X \neq 0) = 1 - e^{-\lambda}$; suppose we have n samples $j_1, j_2, \cdots j_n$, indicating whether a page is accessed ($j = 1$) or not ($j = 0$) during each unit time. Let $f(j|\lambda)$ denote the probability of observing j in each unit time, then $f(0|\lambda) = \Pr(X = 0)$, $f(1|\lambda) = \Pr(X \neq 0)$, and $f(j|\lambda) = (1 - 2e^{-\lambda})j + e^{-\lambda}$. To find the maximum likelihood estimation (MLE) of λ, we solve the maximum likelihood function: likelihood$(\lambda) = \prod_{i=1}^{n} f(j_i|\lambda)$. Suppose $\Sigma j = \sum_{i=1}^{n} j_i$, and consider that j_i has only two values 0 and 1, the likelihood function has its maximum value when $\lambda = \log n/(n - \Sigma j)$, *i.e.*,

$$\hat{\lambda}_{\text{MLE}} = \log\frac{n}{n - \Sigma j} \tag{4}$$

When $\Sigma j = 0$, $\hat{\lambda}_{\text{MLE}} = 0$; when $\Sigma j = n$, without loss of generality, $\hat{\lambda}_{\text{MLE}}$ is $+\infty$. However, either $\hat{\lambda} = 0$ or $\hat{\lambda} = +\infty$ suggests that the profiling window is not appropriate for the specified page. If many pages have such a situation, adjustment of the sampling time (unit time) should be considered. By shortening the sampling time, we can observe $j_i = 0$ even for frequently accessed pages. On the contrary, lengthening the sampling time leads to $j_i = 1$ observable even for rarely accessed pages.

D. Dirtying Rate Modeling

In our model, one critical task is to calculate the dirtying rate of a set of memory pages or storage trunks, given the observation window τ. While traditional approaches ([3] and AVG model in [10]) use a simplified version $\texttt{dirty}(\tau) = d\cdot\tau$, we use the probabilistic expected value (unit: $\texttt{pages}/\tau$). For page/trunk i during a period of time τ, X is a random variable of the number of writes, $\Pr_{i,\tau}(X \neq 0)$ is the probability that page i is written during τ. λ is already estimated in (3) and (4). Summing up the probabilistic values of all pages/trunks, we can get the expected number of dirty pages/trunks during τ:

$$\text{dirty}(\tau) = \sum_{i=1}^{m} \Pr_{i,\tau}(X \neq 0) \tag{5}$$

where m equals the number of the pages/trunks concerned.

The quantity $\texttt{dirty}(\tau)$ would be calculated for different τ's, but it is not necessary to enumerate every page each time. The pages/trunks with the same Σj or Σk are aggregated to speed up this calculation.

E. Performance Restoration Agility

We calculate the *performance restoration agility* (Γ) as defined in (1). That is, we need to find δT, D and ΔT. Note that we already fix δT as a given argument. D is the result of a combination of the methodology by Akoush *et al.* [10] and our $\texttt{dirty}(\tau)$ function (5).

$$D = \text{Result of Akoush simulation using dirty}\,(\tau) \tag{6}$$

ΔT is calculated as follows. At the beginning of the post-copy phase, suppose K_{mem} memory pages are expected to be fetched on demand when handling the workload of w. We have:

$$K_{\text{mem}} = \sum_{i=1}^{m} \text{Pr}_{\text{mem},i,\delta T}(X \neq 0) \tag{7}$$

where m here is the number of pages to be post-copied. In the same way, suppose K_{storage} storage trunks are expected to be fetched on demand, then we have: $K_{\text{storage}} = \sum_{i=1}^{m} \text{Pr}_{\text{storage},i,\delta T}(X \neq 0)$, where m is the number of trunks to be post-copied.

In addition, we know that such delayed memory and storage operations will affect the overall performance of the migrated VM during ΔT. Note that the migrated VM has to complete a workload of w within ΔT, *i.e.* $(\Delta T - \delta T)$ CPU time is wasted in waiting for some memory or storage to be fetched remotely from the source. Suppose the penalty is denoted by ϕ, we have (8):

$$\Delta T = \delta T + \phi_{\text{mem}} \cdot K_{\text{mem}} + \phi_{\text{storage}} \cdot K_{\text{storage}} \tag{8}$$

With δT, D and ΔT, we can apply (1) to find $\varGamma$.

IV. Implementation

A. Model-based Search for Optimal M and S

We define the optimal solution as one that can achieve the highest $\varGamma$ (1). If two migrations have similar $\varGamma$'s (their difference is less than 1%), the one with shorter downtime, remote uptime, and then total migration time are preferred (in the above order). We argue $\varGamma$ can truly reflect the user perceived performance during the migration process.

To find the optimal M and S, we first isolate the two variables and find the best S by ignoring memory. Then the calculated S is used as a parameter to search for the best M. Suppose we are given S and are now searching for the best M. If the predicted downtime at $M = 100\%$ is zero, then the downtime is zero for any $M < 100\%$. So $\varGamma$ is monotonically increasing until it reaches 100%. If we can pre-copy $a_1\%$ and $a_2\%$ $(a_1 < a_2)$ of memory with convergence, then both situations have zero downtime, and ΔT for $a_2\%$ must not be larger than ΔT for $a_1\%$ because the $a_2\%$ case has more prepared memory pages at the resuming time. In this case, we will consider U and T, which favor a small M. We can use binary search to find the minimum M whose $\varGamma$ is 100%.

On the other hand, if the memory pre-copying cannot converge at $M = 100\%$, then the downtime is non-zero. $\varGamma$ is not 100% for $M = 100\%$. The best $\varGamma$ is obtained for some M between 0% and 100%. For any percentage smaller than that value of M, the downtime is smaller; yet performance degradation is heavier. While for any percentage larger than that M, performance degradation gets lighter but the downtime becomes longer. By ignoring the noises caused by discreteness and computation errors, it can be assumed that the whole curve of $(M, \varGamma)$ is a unimodal curve. By this assumption, we can use the ternary search to find the best value of $\varGamma$. We can combine the binary search and the ternary search into a novel search algorithm listed in Algorithm IV.1.

Algorithm IV.1: TernarySearch($\mathit{left}, \mathit{right}$)

```
if $right - left$ is small enough
  then return $((left + right)/2)$
  else
    $l \leftarrow left + (right - left)/3$
    $r \leftarrow right - (right - left)/3$
    if $\varGamma(r) = 100\%$ or $\varGamma(l) > \varGamma(r)$
      then TernarySearch($left, r$)
      else TernarySearch($l, right$)
```

When the same algorithm is applied to storage to find S, as mentioned before, memory is ignored. If storage does not converge, memory is unlikely to converge as well since all the storage access actually goes through memory. If we choose an S value that cannot converge, then the high update rate of storage would make the pre-copied blocks updated again while waiting for memory pre-copying to finish. So for storage, we only find the best $\varGamma$ with $D = 0$.

B. System Implementation on the Xen VM

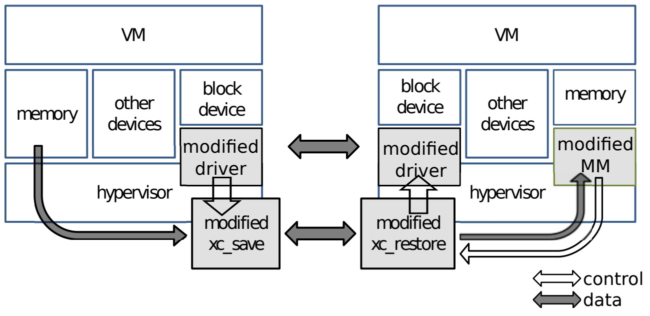


Fig. 2. Our system design overview

1) System Design: Fig. 2 shows an overview of our system architecture. We extend the existing Xen migration manager (i.e. *xc_save* and *xc_restore*, which run on the source and destination hosts sending and receiving memory pages respectively). In vanilla Xen, *xc_save* and *xc_restore* only have the pre-copy and freeze-and-copy phases. They are extended to support post-copying. *xc_save*'s pre-copy phase is modified to copy only $M\%$ of the memory. The remaining memory pages are marked in a bitmap (*to_postcopy*) for moving during the post-copy phase. We also extend the QEMU disk backend system with a new layer for traditional storage drivers to support migration storage migration over a WAN. During the pre-copy phase, the dirtied trunks in the $S\%$ of storage are maintained in a priority queue based on the write frequencies of each trunk, which are collected since profiling and continuously maintained during the pre-copy phase. The least frequently updated trunks will be migrated first.

2) Page Tracking: The *page tracking* method is used to handle the pages marked in the *to_postcopy* bitmap. We ensure each page table entry (PTE) pointing to such pages has a specified reserved-bit marked. During the post-copy phase, access to a marked PTE will trigger what we call a post-copy fault. A page-copy fault is handled by the hypervisor's handler, and will not be passed to the guest's handler. For performance reasons, when a page is removed from the *to_postcopy* bitmap, the update of its corresponding PTE is deferred. So there might be situations that a marked PTE points to a page which is no longer in *to_postcopy*. In this situation, the handler just clears the reserved bit and returns immediately. In other situations, an on-demand request is sent from *xc_restore* to the *xc_save*. The VM then keeps retrying the same instruction until the page is resolved. A scheduler "yield" is used to reduce such retrying and to wake up *xc_restore* to send the request sooner. Besides this, if the guest is faulting at a non-critical location, for example at user level or along a preemptable kernel path, then the page-faulting process will run out of time slices and be scheduled out by the guest OS itself soon. However, this might cause another post-copy fault, so the post-copy handler must support parallel page requests.

3) Whole Page Overwriting: A method called *whole page overwriting* is used to improve the performance of the post-copy phase. When the hypervisor handles a write post-copy fault that overwrites a *to_postcopy* page entirely (*e.g.* when the VM calls Intel instructions *rep movsb* and *rep stosb* with *edx* equal to a multiple of page size), it overwrites the page immediately without sending an on-demand request. Storage migration could also use whole page overwriting since storage operations are always in units of blocks. In this way, we not only save the bandwidth for more critical data, but also reduce the performance degradation caused by on-demand fetching. For example, the v8 benchmark suite (v8-bench) [15], in which nearly 20% of the memory pages are overwritten, can exploit this technique well.

V. Performance Evaluation

In this section, v8-bench [15] and SysBench [16] are used to evaluate the proposed migration framework. We installed two machines A and B in a cluster to act as the source machine and destination machine. To emulate a WAN environment, another machine W is used to redirect packets between A and B, restricting the bandwidth between them as well as adding delays to packets. The *tun/tap* driver overhead in A and B is the same as in any overlay network solution of wide-area migration such as our previous work WAVNet [8]. All the machines are equipped with Intel Core2 Duo CPU E6750 running at 2.66GHz with 2GB memory each. Host and guest OSes are both Linux 3.3.4. The hypervisor is Xen version 4.1.2 with memory management and QEMU backend modified.

A. Evaluation of Access Frequency Profiling

We run the benchmark programs on the VM, and carry out profiling for 10 time slots. After each time slot, we randomly restart the VM and delay some time before the next time slot. These 10 time slots are used to estimate λ and predict the probability of access to each page/trunk. Another time slot is profiled to be the test case. j^* and j_{actual} represent the predicted and actual values of whether a page/trunk is accessed respectively. The prediction is based on (2), (3) and (4).

We first use v8-bench to evaluate the memory profiling. v8-bench is a suite of Javascript-based benchmarks including *cryto*, *raytrace*, *regexp*, *etc*. which are executed on the Google V8 Javascript engine [15]. Each benchmark takes about two seconds and reports a score. We configure it to repeatedly execute each of the benchmarks so that a sequence of scores and timestamps are collected afterward. The results are listed in Table I. We find that v8-bench is quite memory-intensive. Within 20 seconds, 36.4% and 45.5% of the memory pages are read and written respectively. In general, the accuracy of memory profiling is good. We found accuracy$_R$ and accuracy$_W$ are as high as 86.1% and 96.1% respectively.

TABLE I. Overall evaluation of the memory prediction

Read			Write		
j^* \ j_{actual}	0	1	j^* \ j_{actual}	0	1
0	59.8%	10.1%	0	54.1%	3.5%
1	3.8%	26.3%	1	0.4%	42.0%
accuracy$_R$	86.1%		accuracy$_W$	96.1%	

TABLE II. Overall evaluation of the storage prediction

Read			Write		
j^* \ j_{actual}	0	1	j^* \ j_{actual}	0	1
0	75.1%	0.9%	0	96.6%	3.4%
1	2.2%	21.9%	1	0.0%	0.0%
accuracy$_R$	96.9%		accuracy$_W$	96.6%	

We also evaluate the prediction of storage access based on an I/O-intensive program—SysBench [16]. In this test, we run SysBench in *fileio* test-mode with read/write ratio equal to 15. SysBench will issue random read/write requests to one of the benchmark files (total size is about 2 gigabytes). The results are summarized in Table II. Both read and write accuracies (accuracy$_R$ and accuracy$_W$) are above 96% in this case.

B. Evaluation of the Prediction Model

We use our modeling method to predict the migration performance of v8 benchmarks. We validate our model by comparing the predicted results with the real-world experimental results. The experiments are conducted in a virtualized network with 5-MBps bandwidth and 5-ms round-trip time (RTT).

We first apply estimated λ to predict $\texttt{dirty}(\tau)$, and then use the method as in Akoush *et al.* [10] to predict the total migration time, remote uptime and downtime. For the case of $(M, S) = (60\%, 50\%)$, as shown in Table III, the predicted downtime is $49.2s$ while the actual downtime is $53.7s$, which has an error of 8.4% only.

After D is predicted, from (7) and (8), we use λ_R to predict ΔT so as to get Γ. δT is set to 20 seconds. In our experiments, we could achieve $11ms$ of on-demand fetching delay. Therefore, we assume the performance penalty of each memory miss (ϕ_{mem}) is $11ms$. The storage fetching delay is a little longer as it is with higher latency. We assume $\phi_{\text{storage}} = 20ms$. For v8-bench, the storage activity is so low that most storage access is cached by the VM memory.

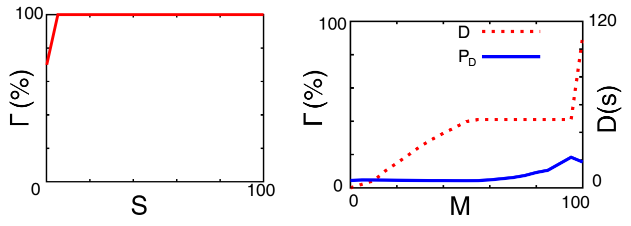
For the case of $(M, S) = (60\%, 50\%)$, we predict $\Delta T = 366.28s$, which means that it will take the VM $D + \Delta T = 415.48s$ to finish the w workload in 20 seconds at full speed. The performance, as Γ suggests, is only 5%. Thus, we can tell $M = 60\%$ is not a good choice. We plot the predictions for different combinations of S and M in Fig. 3. In Fig. 3 (a), only storage is considered, so ϕ_{memory} is temporarily considered zero. The storage reads are more frequent than storage writes,

TABLE III. Prediction of T, U and D

	Predicted (s)	Actual (s)
Total migration time (T)	1063.3	988
Remote uptime (U)	554.3	493
Downtime (D)	49.2	53.7

so the downtime D is not severe when S grows. However, ΔT increases when S falls below 5%. When $S >= 5\%$, Γ approaches 100% and the downtime is nearly zero. Our search algorithm will then consider the remote uptime and total migration time, which favor smaller S.

In Fig. 3 (b), S is given, *e.g.* 50%. As long as the storage could converge, the Γ curve of M would not vary too much because storage will not introduce extra downtime and storage on-demand fetching. Because v8-bench is a memory-intensive benchmark, the best Γ (at $M = 95\%$) in the figure does not exceed 20%. By applying our search algorithm, we can look up the solution at a finer resolution. The combination of (M, S) at $(95\%, 3\%)$ was thus found.



(a) Consider only storage (b) $S = 50\%$

Fig. 3. The predicted Γ with under different (M, S)'s

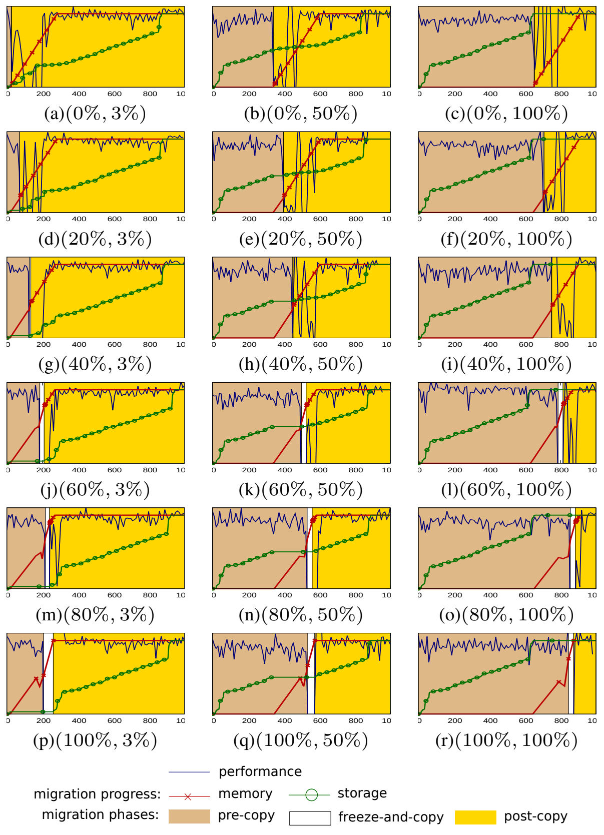


Fig. 4. Comparison of 18 migrations using different (M, S)'s

To validate our model, 18 combinations of M and S are experimented; their results are shown in Fig. 4. The pre-copy, freeze-and-copy and post-copy phases are separately shown in rose, white and yellow background. The green curve (with circle markers) represents the migration progress of storage, increasing from zero to 100%. The red curve (with cross markers) refers to the migration progress of memory, starting from somewhere when S of the storage is migrated. The blue curve is the score of v8-bench, normalized to the range $[0, 100\%]$. From these 18 figures, we can validate the predictions we have made:

1) Storage converges even for $S = 100\%$: in Fig. 4 (c)(f)(i)(l)(o)(r), we can see that the percentage of migrated storage reaches 100% before the memory starts the pre-copy phase in all cases;
2) Storage does not introduce performance degradation when $S \geq 3\%$: in figures (p)(q)(r), the blue curve stays above 90% during the post-copy phase;
3) The predicted downtime shown in Fig. 3 (b) matches the width of the freeze-and-copy phases (white areas) in figures (b)(e)(h)(k)(n)(q).

By comparing the simulated results and the experimental results, we conclude that the proposed model can predict the real migration behavior accurately. We can see the $(95\%, 3\%)$ case lies between Fig. 4(m) and (q). That means the optimal solution selected by our model-based solution performed better than the vanilla pre-copy algorithm (Fig. 4 (r)) and pure post-copy algorithms (Fig. 4 (a)).

C. *Migration Performance Comparison*

We compare six different configurations of v8-bench migrations, as shown in Table IV. We tried to compare the model-based solution with different combinations of pre-copy and post-copy settings, and try to evaluate the adaptiveness of the model-based solution.

TABLE IV. SIX CONFIGURATIONS OF MIGRATIONS

Memory (M)	**Storage (S)**	**Name shown in the figures**
pre-copy (100%)	pre-copy (100%)	pre-copy
pre-copy (100%)	post-copy (0%)	100-0
post-copy (0%)	pre-copy (100%)	0-100
hybrid-copy (50%)	hybrid-copy (50%)	50-50
post-copy (0%)	post-copy (0%)	post-copy
fractional hybrid-copy		FHC

1) Performance of CPU-bound Workloads: For v8-bench, after adopting the whole-page overwriting technique, the model-based solution outputs $M = 48\%, S = 0\%$. In Fig. 5, we can see that the model-based results, denoted by FHC in the table, are among the best configurations in every metric. Notably, the model-based solution can avoid long downtime as in any configurations with large M, and can also maintain a reasonable performance.

2) Performance of I/O-bound Workloads: In this test, we run SysBench. It accesses the files randomly with both read and write operations, where the read/write ratio is about 4:1 and the cache usage is limited to be within 50 megabytes.

In this case, the model-based solution outputs $(M = 98\%, S = 25\%)$. The results are compared with other configurations and shown in Fig. 6. As we can see, the FHC solution achieves 87% Γ and at the same time, unlike pure pre-copy, the remote uptime stays within 677 seconds. Compared to a post-copy solution, FHC avoids the severe performance degradation at the cost of one extra second of downtime,

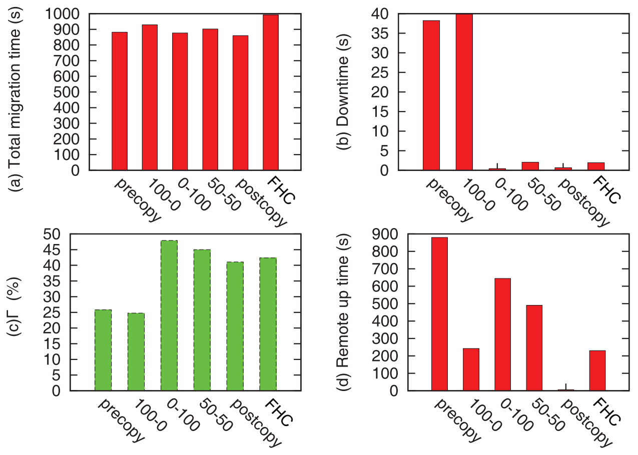


Fig. 5. Overall performance of v8-bench

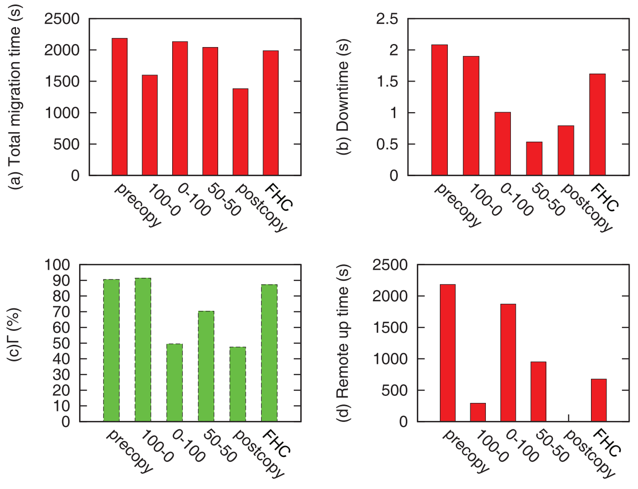


Fig. 6. Overall performance of SysBench

which is reasonable. This again demonstrates a good overall performance of the model-based solution.

VI. Conclusion

This work explores the solutions of live VM migration over a WAN. We propose a flexible and adaptive migration framework, which seamlessly integrates pre-copy and post-copy strategies for migrating both memory and storage. By adjusting the fraction of memory and storage (*i.e.*, M and S) to be migrated before the VM is resumed at the destination, we are able to model the migration behaviors and predict four aspects of migration performance, including total migration time, downtime, remote uptime and performance degradation. We propose a model-based method to find the best M and S combination that adapts well to the current network condition and application behavior, and results in the fastest performance restoration to full speed. The solution includes the memory and storage profiling methods, simulation of pre-copying, performance prediction of post-copying, and the ternary search for the best (M, S) pair. We implement the solution into the Xen VM and validate it step by step. To support the hybrid migration, the implementation employs a unique page tracking technique to perform memory post-copying, and a modified QEMU to support pre- and post-copy storage migration. Our results are compared with traditional pre-copy and post-copy solutions. The all-round and adaptive design of our system helps migrate a VM with balanced and reasonably good performance in all the four aspects.

Acknowledgment

This research is supported by Hong Kong RGC grant HKU 7180/11E.

References

[1] P. Mell and T. Grance, "The NIST definition of cloud computing," National Institute of Standards and Technology, Information Technology Laboratory, Tech. Rep., 2009. [Online]. Available: http://csrc.nist.gov/groups/SNS/cloud-computing/cloud-def-v15.doc

[2] M. Nelson, B.-H. Lim, and G. Hutchins, "Fast transparent migration for virtual machines," in *Proceedings of the USENIX Annual Technical Conference*, 2005, pp. 391–394.

[3] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines," in *Proceedings of the 2nd USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, vol. 2, 2005, pp. 273–286.

[4] M. R. Hines, U. Deshpande, and K. Gopalan, "Post-copy live migration of virtual machines," *SIGOPS Oper. Syst. Rev.*, vol. 43, pp. 14–26, July 2009.

[5] P. Riteau, C. Morin, and T. Priol, "Shrinker: Efficient wide-area live virtual machine migration using distributed content-based addressing," INRIA, Rennes, France, Tech. Rep., February 2010.

[6] F. Travostino, P. Daspit, L. Gommans, C. Jog, C. de Laat, J. Mambretti, I. Monga, B. van Oudenaarde, S. Raghunath, and P. Y. Wang, "Seamless live migration of virtual machines over the MAN/WAN," *Future Gener. Comput. Syst.*, vol. 22, pp. 901–907, October 2006.

[7] S. Akoush, R. Sohan, B. Roman, A. Rice, and A. Hopper, "Activity based sector synchronisation: Efficient transfer of disk-state for WAN live migration," in *Proceedings of the 19th IEEE International Symposium on Modeling, Analysis Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2011.

[8] Z. Xu, S. Di, W. Zhang, L. Cheng, and C.-L. Wang, "WAVNet: Wide-area network virtualization technique for virtual private cloud," in *Proceedings of the 40th International Conference on Parallel Processing (ICPP)*, 2011, pp. 285–294.

[9] R. Bradford, E. Kotsovinos, A. Feldmann, and H. Schiöberg, "Live wide-area migration of virtual machines including local persistent state," in *Proceedings of the 3rd ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, 2007, pp. 169–179.

[10] S. Akoush, R. Sohan, A. Rice, A. W. Moore, and A. Hopper, "Predicting the performance of virtual machine migration," in *Proceedings of the 18th IEEE International Symposium on Modeling, Analysis Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2010, pp. 37–46.

[11] T. Hirofuchi, "A live storage migration mechanism over WAN and its performance evaluation," *Science and Technology*, pp. 67–74, 2009.

[12] M. Noack, "Comparative evaluation of process migration algorithms," Ph.D. dissertation, Dresden University of Technology, 2003.

[13] Y. Luo, B. Zhang, X. Wang, Z. Wang, Y. Sun, and H. Chen, "Live and incremental whole-system migration of virtual machines using block-bitmap," in *Proceedings of 2008 IEEE International Conference on Cluster Computing*, 2008, pp. 99–106.

[14] J. Zheng, T. S. E. Ng, and K. Sripanidkulchai, "Workload-aware live storage migration for clouds," in *Proceedings of the 7th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE)*, vol. 46, no. 7, 2011, pp. 133–144.

[15] "V8 benchmark suite - version 7." [Online]. Available: http://v8.googlecode.com/svn/data/benchmarks/v7/run.html

[16] "Sysbench: a system performance benchmark." [Online]. Available: http://sysbench.sourceforge.net/