Live Migration of Virtual Machine Based on Full-System Trace and Replay[*]

Hai Jin, Haikun Liu, Xiaofei Liao, Liting Hu, Peng Li
Services Computing Technology and System Lab
Cluster and Grid Computing Lab
School of Computer Science and Technology
Huazhong University of Science and Technology, Wuhan, 430074, China
hjin@mail.hust.edu.cn

Abstract

Live migration of *virtual machines* (VM) across distinct physical hosts provides a significant new benefit for administrators of data centers and clusters. However, previous memory-to-memory approaches only make the live VM migration be applied in *local area networks* (LAN), and their improved methods for *wide area networks* (WAN) environment bring the VM migration with a long period of downtime. This paper describes the design and implementation of a novel approach that adopts checkpointing/recovery and trace/replay technology to provide fast, transparent VM migration for both LAN and WAN environments. With execution trace logged on the source host, a synchronization algorithm is performed to orchestrate the running source and target VM until they get a consistent state. This scheme can greatly reduce the migration downtime and network bandwidth consumption. Theoretic analysis indicates that the service downtime to migrate a VM across WANs is equivalent to the downtime cost while migrating a VM in a LAN. Experimental measurements show that for a variety of workloads migrated in a LAN or across WANs, the migration downtime is only 200 milliseconds with a little difference.

1. Introduction

The use of virtual machine (VM) migration technology for data centers management has attracted significant attention in the recent years [3, 4, 14, 21]. The capability of migrating live virtual machine among distinct physical hosts provides a significant new benefit for multiple VM-based environments in many key scenarios: **1)** ***Load balancing***, VMs may be rearranged across physical machines in a cluster to relieve load on congested hosts; **2)** ***Online maintenance and proactive fault tolerance*** [13, 23], sometimes, a physical machine may need upgrades or servicing for upcoming system fault, an administrator should migrate the running VMs to alternative machine(s), freeing the original machine for maintenance. So live VM migration improves system serviceability and availability; **3)** ***Power management*** [15], usually, the load and throughput of servers are uneven but statistically regular at different periods. When some applications running on distributed physical machines are working at light load, we can rearrange those applications with their underlying VMs onto a single host, and then the original hosts may be decommissioned once migration is completed. This strategy of power management help corporations to reduce IT operation costs and benefit the natural environment.

In such situations, the combination of virtualization and migration significantly improves manageability of data centers and clusters. For the above requirements, there are many literatures that related to implementing high-performance migration methods [3, 4, 7, 14, 21]. The most influential approaches are VMotion [14] and XenMotion [4] which were shipped by VMware and XenSource as parts of their products respectively. Their implementation mechanism is similar, because they

[*]This work is supported by National 973 Basic Research Program of China under grant No.2007CB310900.

have the same applying scenarios (in a LAN) and analogical scheme for migrating physical memory and network connections. In their solutions, there are mainly three kinds of states that should be solved when a VM is being migrated in a LAN: the VM's physical memory; the network connections and virtual device state; the SCSI storage.

The most intractable issue is migrating physical memory, because it is the main factor that affects the migration *downtime*, *i.e.*, the time during which the services on the VM are entirely unavailable. VMotion and XenMotion adopted pre-copy algorithm [14, 4] to address this issue. Although the memory pre-coping algorithm is able to decrease the downtime to the magnitude of millisecond, there are still some unsolved issues which should be considered further. First, when the rate that memory pages dirtied is faster than the rate of pre-copy procedure, all pre-copy work will be inefficient and one should immediately stop the VM and copy the entire memory pages to the target host. So some memory-intensive workloads would get no benefit from pre-copy algorithm and the downtime may rise to several seconds. This limitation also makes the algorithm only applicable in high-speed LANs. Second, memory-to-memory approach is only applicable to LAN but unfit for low-bandwidth WAN because of the difficulty of live migrating the local persistent state (for example, file system) and on-going network connections among different networks. Third, some paravirtualized optimization scheme, such as stunning rogue processes and freeing unallocated pages that are mentioned in XenMotion [4] may cause some negative effect to users' experience, especially for some latency-sensitive interactive services. At last, pre-copy algorithm did not recover the CPU's cache data. Although it may not lead to any mistake on the target host, massive cache and TLB missing may cause evident performance degradation once the VM takes over the service.

In this paper, we propose a novel live VM migration approach. The target is to minimize the migration downtime and network bandwidth consumption. We implement our prototype based on a full-system trace and replay system - ReVirt [5]. Checkpointing/recovery and trace/replay technology is adopted to provide fast, transparent VM migration in both LAN and WAN environments. A trace daemon continuously logs the non-deterministic events of the VM while sacrificing very little performance. The execution trace file logged at the original host is iteratively transferred to the target host and used to synchronize the migrated VM's execution state. Experimental measurements show the migration downtime is almost only 200 milliseconds for most of applications, which is really unperceivable for most of client users.

The contribution of this paper is mainly in two aspects: our preliminary work shows that the full-system replay technique can re-create VM state without any deviation on x86 uni-processor, thus provide a novel VM migration approach to minimize the downtime and network bandwidth consumption; the proposed scheme makes the migration downtime equivalent for both LAN and WAN environments, so our approach makes live migration become practical and efficient in WAN environment.

The remainder of the paper is organized as follows. Section 2 describes the related work about VM migration. Section 3 gives a brief introduction about the groundwork - ReVirt. Section 4 presents the design and implementation of our approach. Section 5 gives a theoretical analysis about our approach's performance. Section 6 presents the experiments undertaken and results obtained, demonstrating that it provides an effective solution. Finally, we conclude our work in section 7.

2. Related Works

Recent virtual machine management tools [4, 14] allow live migration of servers within a local area network environment. These technologies have proven to be a very effective tool to enable data center management in a non-disruptive fashion. Pre-copying algorithm is widely used for process-level [23] and VM-level [4, 14] live migration in a memory-to-memory approach. During migration, physical memory pages are pushed across network to the new destination while the source host continues running. Pages modified during the replication must be re-sent to ensure consistency. After a bounded iterative push phase, a very short stop-and-copy phase is executed to transfer the remaining dirty pages. This COW (copy-on-write) mechanism achieves very short best-case migration downtimes. To greatly reduce memory replication during migration, D.K. Panda et al proposed a high performance VM migration scheme by using RDMA (Remote Direct Memory Access), which is a feature provided by many modern high speed interconnects (InfiniBand). Their approach drastically reduced the VM migration overheads.

In a LAN environment, since the migrated virtual machine retains the same network address as before, any ongoing network level interactions are not disrupted. Similarly, storage requirements are normally met via either network attached storage (NAS) or storage area network (SAN), which is still reachable from the migrated VM location to allow for continued storage access. Unfortunately, in a WAN environment, live VM migration is not as easily achievable for intractable network connectivity holding and bulk storage replication.

Current virtual machine software with a suspend and resume feature that can be used to support WAN migration includes Collective [19], Internet Suspend/Resume [10] and μDenali [22], but those projects have explored migration over longer time spans by stopping the source VM and then transferring the VM image file and local block devices to the target host. To archive live migration of virtual machine across WANs, recent approaches using dynDNS and IP tunnels to guarantee network connections was demonstrated in [3, 21], where an IP tunnel between the source and target host is set up to transparently forward packets to and from the client applications. But the edge router's support is required for those approaches. To replicate the large size of local disk storage with less disruption, R. Bradford et al [3] described a block level solution combining pre-coping with write throttling. K. K. Ramakrishnan et al [18] proposed a flexible mode combining asynchronous replication with synchronous replication that can be dictated by the migration semantics.

To optimize the transfer of large amounts of disk and memory state during migration, M. Satyanarayanan et al [20] proposed a solution based on opportunistic replay, which captures user interactions with applications at the GUI level, resulting in very small replay logs that economize network utilization. This approach is somewhat similar with our mechanism, but its applicable scenario requires that the target hosts should have initially identical replica of a suspended VM with source host before the source VM is migrated to the same host again. Another difference with our approach is that it implemented logging-and-replay only at the GUI level but not full-system, thus resulted in divergent VM state that should be dealt with cryptographic hashing techniques. Moreover, their VM migration scheme was not implemented in a live fashion.

Transparent and efficient access to I/O devices during VM migration needs to dynamically change the mappings of virtual to physical devices. Netchannel [11] presented a VMM-level abstraction that transparently handles pending I/O transactions, thus provided a novel mechanism for continuous and seamless device access during VM migration and device hot-swapping for networked as well as locally attached devices.

3. Deterministic Replay with Execution Trace

Our implementation is based on a full-system trace and replay tool - ReVirt, so it is necessary to give a brief introduction about this system. ReVirt was ported on UMLinux [6] and deigned for intrusion detection. It logs enough information to replay a long-term execution of the virtual machine instruction-by-instruction. This enables it to provide arbitrarily detailed observations about what transpired on the system, even in the presence of non-deterministic attacks and executions. Moreover, ReVirt adds reasonable time and space overhead. Overheads due to virtualization are imperceptible for interactive use and CPU-bound workloads, and 13-58% for kernel-intensive workloads. Logging adds only 0-8% performance overhead, and log growth rates range from 0.04GB per day to 1.2GB per day for the related workloads [5].

Checkpoint/recovery [2, 8, 12] and trace/replay [5, 16, 25] technology is used widely for recovering state. Replaying a process requires logging the non-deterministic events that affect the process's computation. These log records guide the process as it re-executes (rolls forward) from a checkpoint. Non-deterministic events fall into two categories: time and external input. Time refers to the exact point in the execution stream at which an event takes place. External input refers to data received from a user or another computer via a peripheral device, such as a keyboard, mouse, or network interface card. ReVirt only logs asynchronous virtual interrupts and sufficient information to re-deliver the signal at the same point during replay. ReVirt plays back the original asynchronous virtual interrupts using the combination of the hardware counters and host kernel hooks that are used during logging.

Replay can be conducted on any host with the same processor type as the source host. There is not any deviation generated during the replay process, so it is very suitable for us to recover the migrated VM's state during live migration. Figure 1 shows our migration system structure with logging-and-replay component of ReVirt.

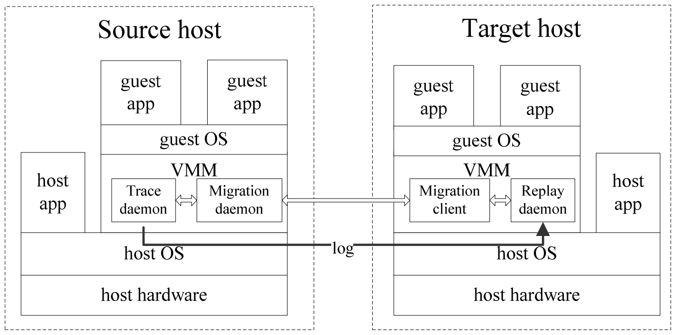

Figure 1. Migration system structure

4. System Design and Implementation

This section describes our design motivation and how we apply the ReVirt to implement VM live migration on top of UMLinux for x86 platform. The system is implemented as a set of modifications to the host kernel.

4.1 Design Objectives

Live virtual machine migration takes a running VM and moves it from one physical machine to another. This process must be transparent to the applications running on the operating system, and remote clients connected to the virtual machine. VM-resident services do not need to be migration-aware in any way. Previous memory-to-memory approaches can only fit this requirement in a LAN, but not in WANs. Our target is to make the migration unaware both in LAN and WANs.

Before we discuss our design and implementation details, some metrics need be introduced to evaluate the performance characteristics of the VM migration scheme: **1)** ***downtime***, the time when no CPU cycle is devoted to any of the VM-resident applications, neither at the source nor at the target system, it consists of the time necessary to quiesce the VM on the source, transfer the system state to the destination, load the device state, and activate the suspended VM on the remote host; **2)** ***total migration time***, the interval between the time migration is initiated and the time the migrated VM gets a consistent state with the original one, *i.e.*, during which the state of two VMs is synchronized; **3)** ***total data transmitted***, the total data is transferred in the migration duration.

When a VM is running live services, it is necessary to ensure that the migration occurs in a manner that balances the requirements of minimizing all the three metrics. Our motivation is to design a live VM migration scheme with negligible downtime and lowest network bandwidth consumption. Furthermore, we should ensure that the migration would not disrupt other active services residing in the same host through resource contention (*e.g.*, CPU, network bandwidth).

4.2 Live Migration Process

This section describes the implementation of our live VM migration approach combining with instructions execution trace and replay. Unlike memory pre-coping algorithms, our method employs the target host's computation capability to synchronize the migrated VM's state. What we copied is the execution log of the source VM but not the dirtied memory pages, and this may greatly decrease the data transferred while synchronizing the two VM's running state. Our approach reduces the downtime by combining a bounded iterative log transferring phase with a typically short stop-and-copy phase. By *iterative* we mean that synchronization occurs in rounds, in which the log file to be transferred during round *n* are those that generated during round *n*-1 (the checkpoint file is transferred in the first round). After several rounds of iteration, the last log file transferred in the stop-and-copy phase may be reduced to a negligible size so that the downtime can be decreased to an unperceived degree. Our approach is suitable for both LAN and WANs, but with more excellent performance when migration is performed in a LAN.

Like the limitation of pre-copy algorithm, the dirty memory pages must be transferred faster than that they are dirtied, there are also some prerequisites for our approach. It is obvious that the log transfer rate should be faster than the log growth rate. Otherwise, our algorithm would be useless because log files will quickly accumulate on the source host. Fortunately, the log data grows far more slowly than it is transferred even when the source VM is running an OS-intensive or I/O-intensive workload [5]. The same outcome has also been presented in other analogical works [16, 25]. Our experiments also show the same conclusion for typical server workloads which we present in table 1. For most workloads, the log growth rate is not more than 1 MB/sec, which is much less than the network transfer rate in a Gbit/s network. Another requirement of our approach is that the log replay rate must be faster than the log growth rate. If this condition is not satisfied, the downtime may be even much longer than the elapsed time transferring the checkpoint file from the source host to the destination. Generally, the replay speed can be faster than the original execution with logging, because during normal execution a process may block waiting for I/O events, while during replay all events can be immediately replayed due to the ability of skipping over the idle time of HLT instructions [5, 25]. We suppose the two above prerequisites are satisfied in the following description and discussion.

Figure 2 shows the whole process while migrating a running VM from host *A* to host *B*. We view the migration process as a transactional interaction between the two hosts involved the following phases:

1) Initialization: a target host with sufficient resources is selected to guarantee the requirement of receiving migration. A good choice may speed the upcoming migration and boost up the server's QoS.

2) Reservation: host *A* makes a request of migrating a VM to host *B*. A VM container of the source VM's size should be reserved to guarantee the necessary resources are available on host *B*.

3) Checkpointing: the VM on top of host *A* freezes, the system state (virtual main memory, CPU registers, memory from external devices and virtual disk) at the current instant are saved to an image file in a copy-on-write fashion. After checkpointing, the source VM continues to run as though nothing had happened.

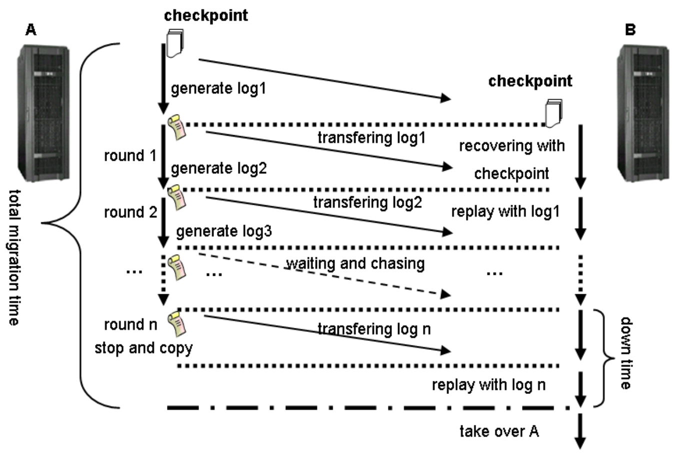

Figure 2. Process of live VM migration

4) Iterative Log Transferring: during the first round of transferring, the checkpoint file is copied from host *A* to *B*, while the VM on host *A* is continuously running and non-deterministic system events are recorded in a log file. Subsequent iterations copy the log file generated during the previous transfer round. At the same time, host *B* is replaying with the received log files once it had recovered from the checkpoint. As the log is transferred much faster than the log generated, this iterative process is convergent.

5) Waiting-and-Chasing: after several rounds of iteration, when the log file generated during the previous transfer round is reduced to a specified size (we defined this threshold value as V_{thd} in section 5 and set it to 1KB in our experiments), host *A* inquires *B* whether the stop-and-copy phase can be executed soon, if the resumed VM on host *B* does not replay fast enough, *i.e.*, the cumulative unused log file size on host *B* is still bigger than V_{thd} at this time, host *B* should inform host *A* to postpone the stop-and-copy phase until the log is used up on host *B*. The iterative log transferring should be continuously performed till the size of unconsumed log at host *B* reduced to V_{thd}. As the log replay speed on host *B* is faster than log generated speed on host *A*, the migrating VM on host *B* would chase up the running state of the source VM finally.

6) Stop-and-Copy: the source VM is suspended and the remaining log file was transferred to host *B*. After the last log file is replayed, there is a consistent replica of the VM at both *A* and *B*. The VM at *A* is still considered to be primary and may be resumed in case of failure.

7) Commitment: host *B* informs *A* that it has successfully synchronized their running states. Host *A* acknowledges this message as commitment of the migration transaction, and then all its network traffic is redirected to host *B*. Now the source VM may be discarded.

8) Service Taking Over: the migrated VM on host *B* is activated now, and the new VM advertises its moved IP address. Host *B* becomes the primary host and takes over host *A*'s service.

This approach should be a fault-tolerant process. The source host should remain a stable state no matter what sort of failure occurs during migration. This guarantees the service continuously running on the source VM with no risk of failure until the migration commits.

4.3 Implementation Challenges

A key challenge of migrating a running VM is how to hold the connections to local device including SCSI disks and network interfaces. There are different solutions to address such issues when a VM is migrated in a LAN or WANs. The remainder of this section describes the detail issues about migrating the two most important components.

SCSI Devices In a cluster environment, most modern data centers consolidate their storage requirements with network-attached storage (NAS) or storage area networks (SAN). The NAS can be accessed uniformly from all host machines in the cluster, and this advantage avoids the need to migrate disk storage. Unfortunately, it may cause some conflicts while synchronizing the migrated VM's execution state in our migration approach. For instance, the source VM may first read a block of the disk and then write the same block, if this process is replayed on the target host without any intervention, this may cause some mistakes, because at this time the reading data is what the source VM had written. We should avoid such style of replay that can be named as WAR (write after read) for short. There are two common approaches to address this issue. The first is to track the disk changes in a redo/undo log [9] during the synchronization phase, but it is very difficult to cope with the time sequence of disk access, because the logging and replay are performed synchronously that the two VM may compete to read or write the same block at the same time. Another approach is chosen to address this issue in our system. All the disk read operation on the source VM are intercepted and the bytes are recorded in a log file, during replay on the target host, the disk read are prohibited and redirected to the log file. All the disk writing operations are also prohibited during the replay process. Although this approach causes some space penalty and network bandwidth consumption, but it works well and doesn't cause any mistakes. Furthermore, to make the solution much more robust and efficient in a high-speed LAN, a little network bandwidth consumption is worthwhile.

Although NAS and SAN are popular in the modern data centers, some environments may still make extensive use of local disks. In some cases (*e.g.*, data migration between data centers for maintenance or fault-tolerance), the physical machine needs servicing or upgrades, the original host should be powered off after the VMs had been migrated to remote data centers, the local storage migration across WANs becomes necessary. These cases present a significant challenge to migrate the usually considerable large size of disk storage. A block level solution combining pre-copying with write throttling was described in [3]. As our main purpose is not to solve the complicated issue of migrating large storage across WAN, in the following discussion, we do not consider local disk migration if not mentioned.

Network Connections To ensure the transparency of VM migration, it is essential to guarantee all the network connections that were opened before migration keeping open after the migration finished. In a cluster environment, the network interfaces of the source and target hosts typically attached to the same switched LAN. VMware and Xen address this issue with similar mechanism of ARP broadcasting [14, 4], and we adopt such analogous method to keep ongoing network connections while migrating a VM in a LAN.

When VM migration takes place in different networks, a new IP address should be allocated to the migrated VM, and thus existing network connections may break. We employ temporary network redirection scheme to address such issue by combining IP tunneling [17] with dynamic DNS [24], which also had been adopted in the literature [3]. With such technology, existing connections can continue transparently while new ones are redirected to the new network location.

5. Analysis

Before we discuss the performance of our approach for live VM migration, some important notations and their corresponding definitions should be introduced in advance:

R_{log}: Log growth rate, which denotes the average growth rate of the source VM execution trace for a special workload.

R_{trans}: Log transfer rate, which mainly lies on the network bandwidth between the two hosts. It is also an average value.

R_{replay}: Log replay rate, which denotes the average rate of replay with the log files on target host.

V_{thd}: The threshold value of log data size at which the iterative log transfer should be terminated.

We define the log file list transferred at each rounds as L=<log_1, log_2, …, log_n>, and their file size as $V=<V_{log_1},V_{log_2},\cdots,V_{log_n}>$ correspondingly. The elapsed time sequence at each transferring rounds is defined as T=<t_0, t_1, t_2, …, t_n>, while t_0 presents the elapsed time to transfer the checkpoint of the source VM and V_{ckpt} denotes the data size of the checkpoint file.

In the following analysis and experiments, the V_{thd} is set to a specified value (1KB). And to make our model simpler, we deem that the R_{trans} is a constant value. We mainly concern the three performance evaluation metrics in two scenarios: *fast synchronization* and *slow synchronization with waiting-and-chasing*. The discriminate of the two scenarios is that there is an additional waiting-and-chasing phase performed in the second scenario comparing with fast synchronization.

5.1 Fast Synchronization

In this scenario, the log replay rate is much higher than the log growth rate (R_{replay}>>R_{log}). For instance, if the VM is running only for daily use, the R_{replay} is 33 times bigger than R_{log} according to the ReVirt's experiment [5]. In this condition, replay is executed on the target host so fast that there is no need to do the *waiting-and-chasing* phase. The R_{log} and R_{trans} are the main factors which affect the process of migration. The detail migration process is shown in Figure 3.

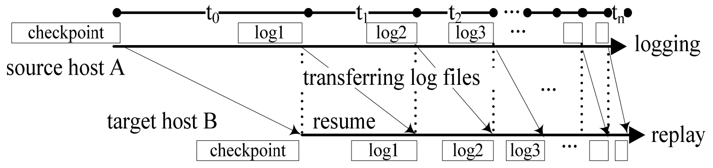

Figure 3. Migration process of fast synchronization

Because R_{trans}>R_{log}, the iteration of log transfer process is convergent. After several rounds of iteration, the last log file size may reduce to the fixed value V_{thd}. The elapsed time in each round can be calculated like this: $t_0=\frac{V_{ckpt}}{R_{trans}}$, $t_1=\frac{R_{log}t_0}{R_{trans}}$, …, $t_n=\frac{R_{log}t_{(n-1)}}{R_{trans}}=\frac{V_{ckpt}R_{log}^{(n-1)}}{R_{trans}^{n}}$, where t_0 presents the time cost to transfer the data of checkpoint, and t_n presents the time cost to transfer the log file generated during previous round. Let φ ($0<\varphi<1$) denote the ratio of R_{log} to R_{trans}:

$$\varphi = R_{\log} / R_{trans} \tag{1}$$

The elapsed time during the round n is presented as:

$$t_n = t_0\varphi^{(n-1)} = \frac{V_{ckpt}\varphi^{(n-1)}}{R_{trans}} \tag{2}$$

Then the *total migration time* (TMT) can be calculated as:

$$TMT=\sum_{\mathrm{i=0}}^{\mathrm{n}} t_i = \frac{V_{ckpt}(1-\varphi^n)}{R_{trans}(1-\varphi)} \tag{3}$$

With the equation (3), the *total data transmitted* (TDT) during a migration becomes:

$$TDT=V_{ckpt}+\sum_{i=1}^{n}V_{\log_i}=TMT * R_{trans}=\frac{V_{ckpt}(1-\varphi^n)}{1-\varphi} \tag{4}$$

Now, we analyze the *downtime* caused in the whole migration process. It is composed of two parts: t_n, the time the last log file (only 1KB) is transferred during the stop-and-copy phase, it is very short both for LAN and WAN environments; and the time the last log file is replayed on the target host. All the two parts can be done within a very short time interval. The total downtime can be represented as:

$$T_{downtime} = t_n + V_{log_n} \big/ R_{replay} \tag{5}$$

To evaluate the convergence rate of our algorithm, we can calculate the total rounds of the iteration by the inequality (6):

$$t_{(n-1)} R_{log} \le V_{thd} \tag{6}$$

It is the condition when the iterative log transfer should be terminated. Combining with equation (1) and (2), inequality (6) can be transformed to $V_{ckpt}\varphi^{(n-1)} \le V_{thd}$, *i.e.*, $n \le 1 + \log_{\varphi}\frac{V_{thd}}{V_{ckpt}}$, so the iteration round is:

$$n = 1 + \left\lceil \log_{\varphi}\frac{V_{thd}}{V_{ckpt}} \right\rceil \tag{7}$$

From the above equations, we can easily make the following conclusions: a smaller size of checkpoint file and faster network transfer rate would greatly improve the convergence rate of our algorithm, and also reduce the total migration time and the total data transmitted; the log growth rate generate a little effect on the iteration rounds, resulting much less network packages transmitting and shorter total migration time. A simple instance shows that when the network throughput is at a rate of 400Mbit/sec and the checkpoint file is 512MB, the iteration rounds is not more than 5 times even when the log growth rate has raised to 10MB/sec.

5.2 Slow Synchronization with Waiting and-chasing

Here, all the steps are the same as the above scenario, but append a waiting-and-chasing phase before stop-and-copy phase. This synchronization process makes it much more complicated in this case. When the log file size on the source host has reduced to the specified value V_{thd}, there is still much unused log that should be replayed on the target host, so a waiting-and-chasing phase is needed to postpone the stop-and-copy phase until the two VMs get a consistent state. Figure 4 shows the detail process of this scenario. The following inequality presents this condition:

$$\frac{\sum_{i=1}^{m} V_{log_i}}{R_{log}} - \frac{\sum_{i=1}^{m} V_{log_i}}{R_{replay}} <= \frac{V_{ckpt}}{R_{trans}} \tag{8}$$

As $\sum_{i=1}^{m} V_{log_i} \big/ R_{log}$ denotes the time cost when the log file is reduced to the threshold value V_{thd}, it can also be presented with equation (3), as log replay rate can be normalized to log growth rate:

$$R_{replay} = \partial R_{log} \tag{9}$$

Where ∂ denotes the ratio of log replay rate to log growth rate. Combining with equation (3) and (9), inequality (8) can be transformed to:

$$(1-\frac{1}{\partial})\sum_{i=0}^{m} t_i <= t_0 \tag{10}$$

$$i.e., \ \frac{1-\varphi^m}{1-\varphi} <= \frac{\partial}{\partial - 1} \tag{11}$$

Analyzing the whole process of migration from overall perspective, we can derive that the difference of the two VMs' runtime during the migration is just the time cost transferring the checkpoint. This conclusion can be expressed as:

$$\frac{\sum_{i=1}^{n} V_{log_i}}{R_{log}} - \frac{\sum_{i=1}^{n} V_{log_i}}{R_{replay}} = \frac{V_{ckpt}}{R_{trans}} \tag{12}$$

With equation (12) and (9), we can calculate the total data volume of generated log files as:

$$\sum_{i=1}^{n} V_{log_i} = \frac{\partial\varphi}{\partial - 1} V_{ckpt} \tag{13}$$

So the *total data transferred* can be expressed as:

$$TDT = V_{ckpt} + \sum_{i=1}^{n} V_{log_i} = (\frac{\partial\varphi}{\partial - 1} + 1)V_{ckpt} \tag{14}$$

Figure 5 shows that while migrating a VM with 512MB checkpoint file in a high-speed LAN (400Mbit/sec bandwidth), the total data should be transferred varying with the parameter ∂ and φ.

The *total migration time* can be calculated as:

$$TMT = \frac{V_{ckpt} + V_{log_1}}{R_{trans}} + \frac{\sum_{i=1}^{n} V_{log_i}}{R_{replay}} = (\varphi + \frac{\partial}{\partial - 1})\frac{V_{ckpt}}{R_{trans}} \tag{15}$$

Note that we should not calculate the *TMT* with equation (3) or simply using the expression TDT / R_{trans}, because during the waiting-and-chasing phase, the log files is not being transferred all the time, but they are being replayed on the target host all along.

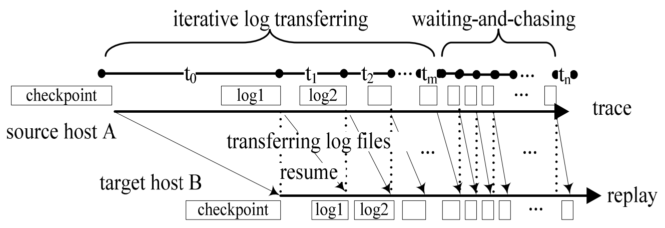

Figure 4. Migration process with a waiting-and-chasing synchronization phase

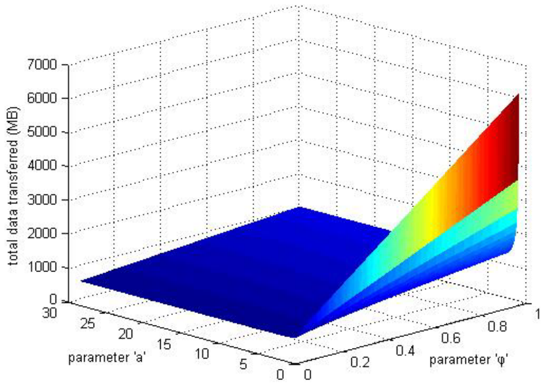

Figure 5. The total data transferred with different parameter values of ∂ and φ during a migration

6. Performance Evaluation

This section investigates the performance characteristics of the VM migration scheme described above. It presents measurements of migration downtime, total migration time and the total data transferred when a VM is migrated in a LAN or across WANs. With a variety of workloads, our approach shows that VM migration can be fast and transparent to applications and operating systems.

6.1 Experimental Setup

Our experiments are performed on identical hosts with AMD Athlon 3500+ processor and 1GB DDR RAM. Storage is accessed via iSCSI protocol from a NetApp F840 network attached storage server. Each host has an Intel Pro/1000 Gbit/s NIC to transfer the state of the VMs. To reduce the effect on other ongoing network service hosted on the source host, we limit the network bandwidth to 500Mbit/sec for the migration daemon. The guest kernel is Linux 2.4.18 ported to UMLinux, and the host kernel for UMLinux is a modified version of Linux 2.4.20. The virtual machine is configured to use 512MB of RAM. To emulate a wide-area network for the experiments, we use the same approach mentioned in [3]. The Linux traffic shaping interface is used to limit network bandwidth to 5Mbps between source and target hosts. To highlight the benefit of our approach for VM migration over WANs by comparing to in a LAN, the storage is also accessed by networked storage server connected to the LAN, we do not consider the disk storage migration across WANs.

The VM being migrated is the only VM running on the source machine and there are no VMs running on the target machine. Each experiment migrates a single VM five times between two physical hosts. The results reported are the average of the five migrations. The experiments use the following VM workloads:

1) ***Daily use***: an idle Linux OS for daily use.

2) ***Kernel-build***: the complete Linux 2.4.18 kernel compilation is a system-call intensive workload, which is expensive to virtualization.

3) ***Static web application***: we use the Apache 2.0.63 to measure static-content web server performance. Both two clients are configured with 50 simultaneous connections and repetitively downloading a 256KB file from the web server.

4) ***Dynamic web application***: a more challenging Apache workload is presented by SPECweb99, a complex application-level benchmark for evaluating web servers and the systems that host them. The workload is a complex mix of page requests: 30% require dynamic content generation, 16% are HTTP POST operations, and 0.5% executes a CGI script. As the server runs, it generates access and POST logs, contributing to disk (and therefore network) throughput.

5) ***Unixbench***: it is a benchmark suite for Linux that integrates CPU, file I/O, process spawning and other workloads. The following tails are performed: Dhrystone2 using register variables, arithmetic, system call overhead, pipe throughput, pipe-based context switching, process creation, execl throughput, file system throughput, concurrent shell scripts, compiler throughput, and recursion.

As our migration approach is based on checkpointing/recovery and trace/replay technology, we name this scheme as CR/TR-Motion for short. To compare CR/TR-Motion with previous migration schemes in LAN environments, we port pre-copy algorithm implemented in XenMotion to UMLinux and make an experiment on the above workloads. The test environment is the same as CR/TR-Motion.

6.2 Logging and Replay Overheads

Table 1 shows the time and space overhead of logging and replay on daily use, kernel-build, static web application, dynamic web application, unixbench workloads. Log growth rate shows the average rate of growth of the log during the workload. Log replay rate is normalized to the log growth rate of UMLinux with logging. It is denoted by the value of φ. Workloads with little non-determinism (*e.g.*, kernel-build) generate very little log traffic. The log growth rate for static web app and SPECweb99 is higher because of the need to log incoming network packets. However, it is still small enough comparing to the network transmitting rate in a Gbit/s LAN. The log replay rate for kernel-build and unixbench is only a little faster than log growth rate because such applications are compute-intensive and generate little non-determinism events.

Table 1. Time and space overhead of logging and replay

Workloads	**Log growth rate (KB/s)**	**φ (Replay rate normalized to logging)**
daily use	7.9	36.8
kernel-build	1.2	1.05
static web app	247	1.63
dynamic web app	722	1.20
unixbench	61	1.08

6.3 Migration Time

Each workload migration is tested in LAN and emulated WAN environment. We mostly concern about the downtime during which the VM is unavailable. This interval must be short enough to avoid any noticeable delay from the VM.

To compare the migration downtime of our scheme (CR/TR-Motion) with pre-copy scheme, the same workloads are migrated in a high-speed LAN. The test result in figure 6 shows that our approach gets less downtime than pre-copy algorithm for most workload (even for some I/O-intensive workloads, *e.g.*, web applications). Figure 7 shows that the downtime to migrate a VM across high-speed LAN. This experiment result verifies the truth of the theoretic analysis for our approach.

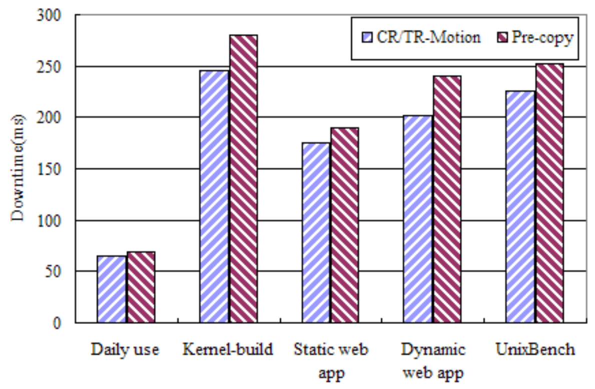

Figure 6. The downtime of CR/TR-Motion and Pre-copy for different workloads migrated in a LAN

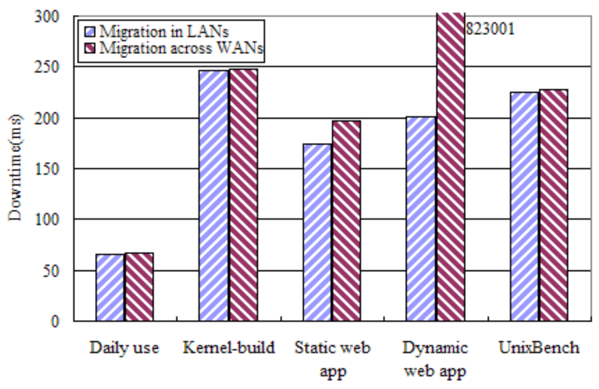

Figure 7. The downtime of CR/TR-Motion for Different workloads migrated across LANs and WANs

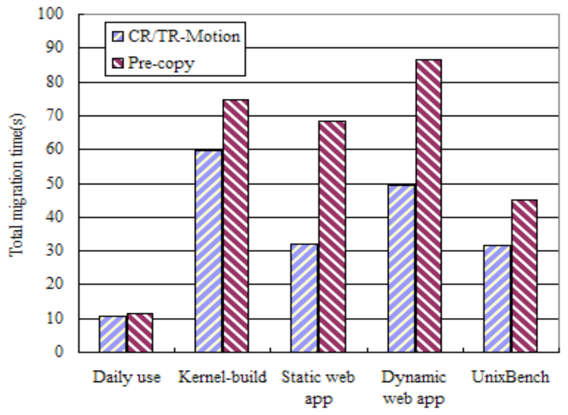

Figure 8. Total migration time of CR/TR-Motion and Pre-copy for different workloads migrated in a LAN

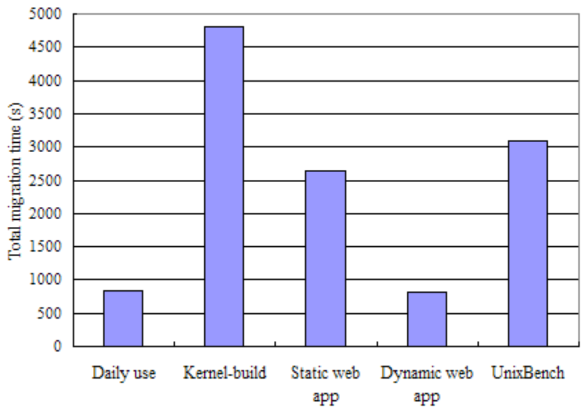

Figure 9. Total migration time of CR/TR-Motion for different workloads migrated across WANs

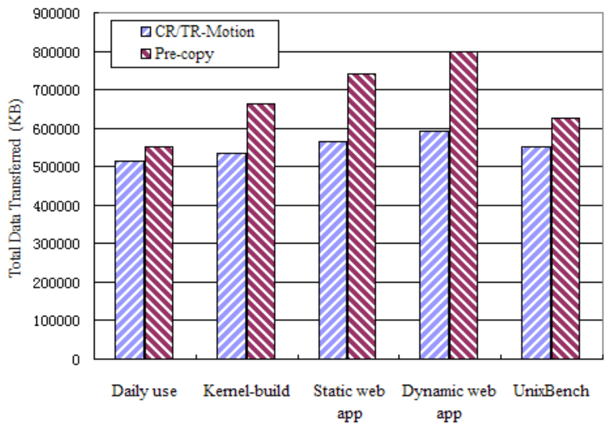

Figure 10. Total data transferred with CR/TR-Motion and Pre-copy for different workloads migrated in LAN

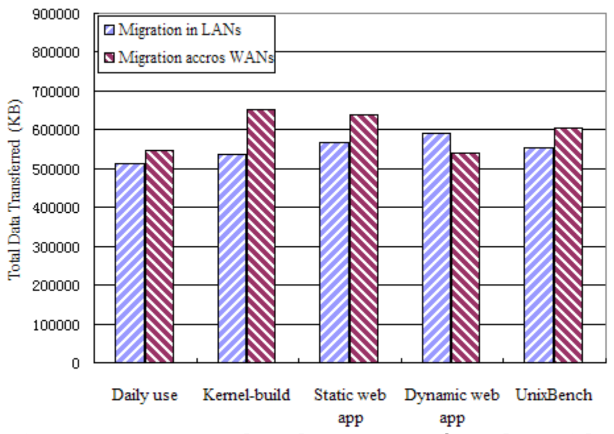

Figure 11. Total data transferred with CR/TR-Motion scheme for different workloads migrated across LANs and WANs

The migration downtime of dynamic web application is extremely long while migrating across WANs, the matter is that the log transfer rate is less than the log growth rate in our test environment and our migration scheme is not suitable for such case, so that the migration downtime equals to the total migration time, which is the time to transfer the VM's checkpoint file from the source host to target host. The same outcome is also demonstrated in Figure 9.

We also pay attention to the total migration time during which machine resources are consumed to perform the migration. Figure 8 shows the total migration time is less than one minute for various workloads while migrating in a fast LAN with our scheme, while pre-copy algorithm takes much more time for the same workloads. This improvement may give a great benefit for cluster administrators. Figure 9 shows that it costs a long period of time to migrate a VM in low-bandwidth WANs environment. The migration elapsed time seems a little long for Linux kernel-building and unixbench. The reason is that the log replay rate is more close to the log growth rate for such workloads, so our algorithm executes many times of iterations to perform the *waiting-and-chasing* phase, which produce much longer total migration time. As our approach is not suitable for migrating dynamic web application in our emulated WAN, the total migration time is just the elapse time to transfer the VM's checkpoint file. This conclusion can be verified in Figure 7 and Figure 9.

6.4 Network Throughput of Migration

Figure 10 shows our migration approach gets less network throughput than pre-copy algorithm in a LAN. For those workloads, the total data transmitted is no more than 600MB in a LAN with our approach (Overhead is less than 10%). Figure 12 and 13 show the network throughout of migration in fast and slow synchronization scenarios. We can find that the source host achieves a consistent throughput of approximately 400Mbit/sec when the checkpoint file is being transmitted, and then iterative log transferring phase is executed, resulting in the network throughput dropping to only 25Mbit/sec. During the waiting-and chasing phase, the bandwidth even drops to 4Mbit/sec. Those results demonstrate that the migrations for those workloads cause reasonable network traffic and bandwidth consumption.

Figure 11 illustrates that the total data transferred does not increase much when the workloads are migrated across WANs. This result indicates that our approach makes live migration become practical and efficient even in WAN environments. We find that the workload dynamic web application even requires more network traffic to migrate in a LAN than migrate across WANs. That is because our approach is not applicable to migrate this workload across low-bandwidth WANs and a stop-and-copy algorithm is executed soon, so there is no log file transferred during the migration.
We make a comparative analysis between the theoretical mathematical results and the actual measured data. For most workloads (daily use, kernel-build, and static web application), the experimental data is consistent with the mathematical models with a little difference. But for the SPECweb99 and unixbench workloads, the test data has some warp comparing to the theoretical results. That may be because our model is built on the three parameters R_{log}, R_{replay}, and R_{trans}, which are average values. While SPECweb99 and unixbench are benchmark suites integrating different applications, the R_{log} and R_{replay} are uneven at different period in our 5 times experiments, and the network bandwidth also change a lot for different loads.

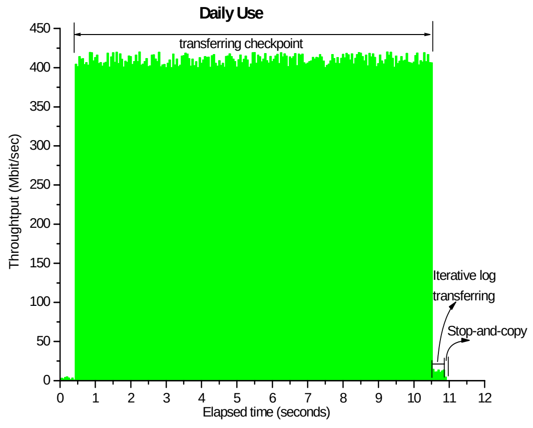

Figure 12. Network throughput of Migrating daily use workload in fast synchronization scenario

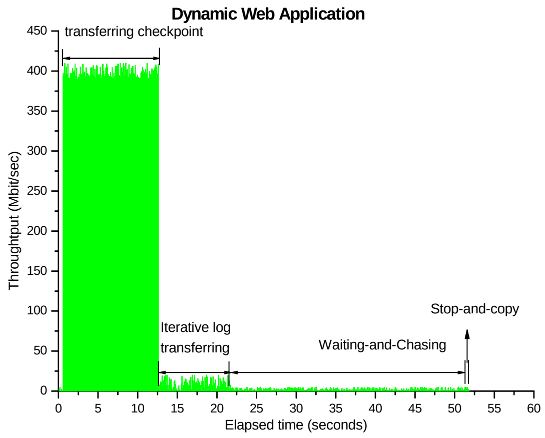

Figure 13. Network throughput of Migrating dynamic web application in slow synchronization scenario

7. Conclusion

In this paper we presented the design, implementation, and evaluation of a novel approach of live VM migration. It shows how we adopt checkpointing/recovery and trace/replay technology to provide fast, transparent VM migration for both LAN and WAN environments. This approach makes unperceived VM migration downtime and reasonable network bandwidth consumption. It allows VM migration in a way which is essentially invisible both for users of this VM and for external clients using network services located inside the VM. The service downtime can be reduced to the order of milliseconds for most of applications even in WAN environments. The experiment results show that our preliminary work makes live migration become practical and efficient in WAN environment. Experimental measurements also show our scheme get better average performance comparing to XenMotion when they are migrated in high-speed LANs. In the future, we will address the issues of migrating the usually considerable large size of disk storage across WAN, and the migration privacy and security challenges related to WAN will also be studied.

8. References

[1] 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”, *Proceedings of the nineteenth ACM symposium on Operating Systems Principles (SOSP’03)*, October 19-22, 2003, Lake George, New York, USA, pp.164-177

[2] Greg Bronevetsky, Rohit Fernandes, Daniel Marques, Keshav Pingali and Paul Stodghill, “Recent Advances in Checkpoint/Recovery Systems”, *Proceedings of 20th International Parallel and Distributed Processing Symposium (IPDPS‘06)*, April 25-29, 2006

[3] R. Bradford, E. Kotsovinos, A. Feldmann, and H. Schioeberg, “Live Wide-Area Migration of Virtual Machines Including Local Persistent State”, *Proceedings of the third International Conference on Virtual Execution Environments (VEE’07)*, ACM Press, June 13-15, 2007, San Diego, California, USA, pp.169-179

[4] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, “Live Migration of Virtual Machines”, *Proceedings of 2nd Symposium on Networked Systems Design and Implementation (NSDI’05)*, May 2-4, 2005, Boston, MA, USA, pp.273-286

[5] G. W. Dunlap, S. T. King, S. Cinar, M. Basrai, and P. M. Chen, “ReVirt: Enabling Intrusion Analysis through Virtual-Machine Logging and Replay”, *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI’02)*, ACM Press, December 8-11, 2002, Boston, MA, USA, pp.211-224

[6] Kerstin Buchacker and Volkmar Sieh, “Framework for Testing the Fault-Tolerance of Systems Including OS and Network Aspects”, *Proceedings of 6th IEEE International High Assurance Systems Engineering Symposium (HASE’01)*, October 22-24, 2001, pp.95-105

[7] W. Huang, Q. Gao, J. Liu, and D.K. Panda, “High Performance Virtual Machine Migration with RDMA over Modern Interconnects”, *IEEE International Conference on Cluster Computing (Cluster'07)*, September 17-20, 2007, Austin, Texas, USA

[8] Joshua Hursey, Jeffrey M. Squyres, Timothy Mattox, Andrew Lumsdaine, “The Design and Implementation of Checkpoint/Restart Process Fault Tolerance for Open MPI”, *Proceedings of 21th International Parallel and Distributed Processing Symposium (IPDPS‘07)*, March 26-30, 2007, pp.1-8

[9] S. T. King, G. W. Dunlap, and P. M. Chen. “Debugging Operating Systems with Time-Traveling Virtual Machines”, *Proceedings of the USENIX Annual Technical Conference (USENIX'05)*, April 10-15, 2005, Anaheim, CA, USA

[10] M. Kozuch and M. Satyanarayanan. “Internet Suspend/Resume”, *Proceedings of the IEEE Workshop on Mobile Computing Systems and Applications (HotMobile’02),* June 20-21, 2002, Callicoon, NY, USA, p.40

[11] Sanjay Kumar, Karsten Schwan, “Netchannel: A VMM-level Mechanism for Continuous, Transparent Device Access During VM Migration”, *Proceedings of the 2008 International Conference on Virtual Execution Environments (VEE’08),* March 5-7, 2008, Seattle, WA, USA, pp.31-40

[12] Yudan Liu, Chokchai Leangsuksun, Hertong Song, Stephen L. Scott, “Reliability-aware Checkpoint/Restart Scheme: A Performability Trade-off”, *IEEE International Conference on Cluster Computing (Cluster'05),* September 26-30, Boston, Massachusetts, USA, pp.1-8

[13] Arun Babu Nagarajan, Frank Mueller, Christian Engelmann and Stephen L. Scott, “Proactive Fault Tolerance for HPC with Xen Virtualization”, *Proceedings of 21st ACM International Conference on Supercomputing (ICS’07),* June 16-20, 2007, Seattle, WA, USA, pp.23-32

[14] M. Nelson, B. H. Lim, and G. Hutchins, “Fast Transparent Migration for Virtual Machines”, *Proceedings of USENIX Annual Technical Conference (USENIX'05),* April 10-15, 2005, Marriott Anaheim, Anaheim, CA, USA, pp.391-394

[15] Ripal Nathuji and Karsten Schwan, “Virtual Power: Coordinated Power Management in Virtualized Enterprise Systems”, *Proceedings of the 22st ACM Symposium on Operating Systems Principles (SOSP’07),* October 14-17, 2007, Skamania Lodge Stevenson, WA

[16] D. A. S. de Oliveira, J. R. Crandall, G. Wassermann, S. F. Wu, Z. Su, and F. T. Chong, “ExecRecorder: VM-Based Full-System Replay for Attack Analysis and System Recovery”, *Proceedings of The 9th Asian Symposium on Information Display (ASID’06),* October 21, 2006, San Jose, California, USA, pp.66-71

[17] C. Perkins, “IP Encapsulation within IP”, RFC 2003, 1996

[18] K. K. Ramakrishnan, P. Shenoy, and J. Van der Merwe, “Live Data Center Migration across WANs: A Robust Cooperative Context Aware Approach”, *Proceedings of The 2nd ACM Workshop on Internet Network Management (INM'07),* Kyoto, Japan, August 31, 2007

[19] C. P. Sapuntzakis, R. Chandra, B. Pfaff, J. Chow, M. S. Lam, and M. Rosenblum, “Optimizing the Migration of Virtual Computers”, *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI’02),* December 8-11, 2002, Boston, MA, USA

[20] Ajay Surie, H. Andr´es Lagar-Cavilla, Eyal de Lara, M. Satyanarayanan, “Low-Bandwidth VM Migration via Opportunistic Replay”, *The Ninth Workshop on Mobile Computing Systems and Applications (HotMobile’08),* February 25-26, 2008, Napa Valley, CA, USA

[21] 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 Computer Systems,* Vol.22, No.8, October 2006

[22] A. Whitaker, R. S. Cox, M. Shaw, and S. D. Gribble, “Constructing Services with Interposable Virtual Hardware”, *Proceedings of the First Symposium on Networked Systems Design and Implementation (NSDI '04),* March 29-31, 2004, San Francisco, USA, pp.169-182

[23] Chao Wang, Frank Mueller, Christian Engelmann and Stephen L. Scott, “Proactive Process-Level Live Migration in HPC Environments”, in *Supercomputing (SC’08),* November 15-21, 2008 (accepted)

[24] B. Wellington. Secure DNS Dynamic Update, RFC 3007

[25] M. Xu, V. Malyugin, J. Sheldon, G. Venkitachalam, and B. Weissman, “ReTrace: Collecting Execution Trace with Virtual Machine Deterministic Replay”, *Proceedings of the Third Annual Workshop on Modeling, Benchmarking and Simulation*, June 10, 2007, California, USA