Live Migration of Virtual Machines

Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen[†],
Eric Jul[†], Christian Limpach, Ian Pratt, Andrew Warfield

University of Cambridge Computer Laboratory — *15 JJ Thomson Avenue, Cambridge, UK* — `firstname.lastname@cl.cam.ac.uk`

[†] *Department of Computer Science* — *University of Copenhagen, Denmark* — `{jacobg,eric}@diku.dk`

Abstract

Migrating operating system instances across distinct physical hosts is a useful tool for administrators of data centers and clusters: It allows a clean separation between hardware and software, and facilitates fault management, load balancing, and low-level system maintenance.

By carrying out the majority of migration while OSes continue to run, we achieve impressive performance with minimal service downtimes; we demonstrate the migration of entire OS instances on a commodity cluster, recording service downtimes as low as $60ms$. We show that that our performance is sufficient to make live migration a practical tool even for servers running interactive loads.

In this paper we consider the design options for migrating OSes running services with liveness constraints, focusing on data center and cluster environments. We introduce and analyze the concept of *writable working set*, and present the design, implementation and evaluation of high-performance OS migration built on top of the Xen VMM.

1 Introduction

Operating system virtualization has attracted considerable interest in recent years, particularly from the data center and cluster computing communities. It has previously been shown [1] that paravirtualization allows many OS instances to run concurrently on a single physical machine with high performance, providing better use of physical resources and isolating individual OS instances.

In this paper we explore a further benefit allowed by virtualization: that of live OS migration. Migrating an entire OS and all of its applications as one unit allows us to avoid many of the difficulties faced by process-level migration approaches. In particular the narrow interface between a virtualized OS and the virtual machine monitor (VMM) makes it easy avoid the problem of 'residual dependencies' [2] in which the original host machine must remain available and network-accessible in order to service certain system calls or even memory accesses on behalf of migrated processes. With virtual machine migration, on the other hand, the original host may be decommissioned once migration has completed. This is particularly valuable when migration is occurring in order to allow maintenance of the original host.

Secondly, migrating at the level of an entire virtual machine means that in-memory state can be transferred in a consistent and (as will be shown) efficient fashion. This applies to kernel-internal state (e.g. the TCP control block for a currently active connection) as well as application-level state, even when this is shared between multiple cooperating processes. In practical terms, for example, this means that we can migrate an on-line game server or streaming media server without requiring clients to reconnect: something not possible with approaches which use application-level restart and layer 7 redirection.

Thirdly, live migration of virtual machines allows a separation of concerns between the users and operator of a data center or cluster. Users have 'carte blanche' regarding the software and services they run within their virtual machine, and need not provide the operator with any OS-level access at all (e.g. a root login to quiesce processes or I/O prior to migration). Similarly the operator need not be concerned with the details of what is occurring within the virtual machine; instead they can simply migrate the entire operating system and its attendant processes as a single unit.

Overall, live OS migration is a extremelely powerful tool for cluster administrators, allowing separation of hardware and software considerations, and consolidating clustered hardware into a single coherent management domain. If a physical machine needs to be removed from service an administrator may migrate OS instances including the applications that they are running to alternative machine(s), freeing the original machine for maintenance. Similarly, OS instances may be rearranged across machines in a cluster to relieve load on congested hosts. In these situations the combination of virtualization and migration significantly improves manageability.

DOCKET ALARM Find authenticated court documents without watermarks at docketalarm.com.

We have implemented high-performance migration support for Xen [1], a freely available open source VMM for commodity hardware. Our design and implementation addresses the issues and tradeoffs involved in live local-area migration. Firstly, as we are targeting the migration of active OSes hosting live services, it is critically important to minimize the *downtime* during which services are entirely unavailable. Secondly, we must consider the *total migration time*, during which state on both machines is synchronized and which hence may affect reliability. Furthermore we must ensure that migration does not unnecessarily disrupt active services through resource contention (e.g., CPU, network bandwidth) with the migrating OS.

Our implementation addresses all of these concerns, allowing for example an OS running the SPECweb benchmark to migrate across two physical hosts with only 210ms unavailability, or an OS running a Quake 3 server to migrate with just 60ms downtime. Unlike application-level restart, we can maintain network connections and application state during this process, hence providing effectively seamless migration from a user's point of view.

We achieve this by using a *pre-copy* approach in which pages of memory are iteratively copied from the source machine to the destination host, all without ever stopping the execution of the virtual machine being migrated. Page-level protection hardware is used to ensure a consistent snapshot is transferred, and a rate-adaptive algorithm is used to control the impact of migration traffic on running services. The final phase pauses the virtual machine, copies any remaining pages to the destination, and resumes execution there. We eschew a 'pull' approach which faults in missing pages across the network since this adds a residual dependency of arbitrarily long duration, as well as providing in general rather poor performance.

Our current implementation does not address migration across the wide area, nor does it include support for migrating local block devices, since neither of these are required for our target problem space. However we discuss ways in which such support can be provided in Section 7.

2 Related Work

The Collective project [3] has previously explored VM migration as a tool to provide mobility to users who work on different physical hosts at different times, citing as an example the transfer of an OS instance to a home computer while a user drives home from work. Their work aims to optimize for slow (e.g., ADSL) links and longer time spans, and so stops OS execution for the duration of the transfer, with a set of enhancements to reduce the transmitted image size. In contrast, our efforts are concerned with the migration of live, in-service OS instances on fast neworks with only tens of milliseconds of downtime. Other projects that have explored migration over longer time spans by stopping and then transferring include Internet Suspend/Resume [4] and μDenali [5].

Zap [6] uses partial OS virtualization to allow the migration of process domains (pods), essentially process groups, using a modified Linux kernel. Their approach is to isolate all process-to-kernel interfaces, such as file handles and sockets, into a contained namespace that can be migrated. Their approach is considerably faster than results in the Collective work, largely due to the smaller units of migration. However, migration in their system is still on the order of seconds at best, and does not allow live migration; pods are entirely suspended, copied, and then resumed. Furthermore, they do not address the problem of maintaining open connections for existing services.

The live migration system presented here has considerable shared heritage with the previous work on NomadBIOS [7], a virtualization and migration system built on top of the L4 microkernel [8]. NomadBIOS uses pre-copy migration to achieve very short best-case migration downtimes, but makes no attempt at adapting to the writable working set behavior of the migrating OS.

VMware has recently added OS migration support, dubbed *VMotion*, to their VirtualCenter management software. As this is commercial software and strictly disallows the publication of third-party benchmarks, we are only able to infer its behavior through VMware's own publications. These limitations make a thorough technical comparison impossible. However, based on the VirtualCenter User's Manual [9], we believe their approach is generally similar to ours and would expect it to perform to a similar standard.

Process migration, a hot topic in systems research during the 1980s [10, 11, 12, 13, 14], has seen very little use for real-world applications. Milojicic *et al* [2] give a thorough survey of possible reasons for this, including the problem of the *residual dependencies* that a migrated process retains on the machine from which it migrated. Examples of residual dependencies include open file descriptors, shared memory segments, and other local resources. These are undesirable because the original machine must remain available, and because they usually negatively impact the performance of migrated processes.

For example Sprite [15] processes executing on foreign nodes require some system calls to be forwarded to the home node for execution, leading to at best reduced performance and at worst widespread failure if the home node is unavailable. Although various efforts were made to ameliorate performance issues, the underlying reliance on the availability of the home node could not be avoided. A similar fragility occurs with MOSIX [14] where a deputy process on the home node must remain available to support remote execution.

DOCKET ALARM Find authenticated court documents without watermarks at docketalarm.com.

We believe the residual dependency problem cannot easily be solved in any process migration scheme – even modern mobile run-times such as Java and .NET suffer from problems when network partition or machine crash causes class loaders to fail. The migration of entire operating systems inherently involves fewer or zero such dependencies, making it more resilient and robust.

3 Design

At a high level we can consider a virtual machine to encapsulate access to a set of physical resources. Providing live migration of these VMs in a clustered server environment leads us to focus on the physical resources used in such environments: specifically on memory, network and disk.

This section summarizes the design decisions that we have made in our approach to live VM migration. We start by describing how memory and then device access is moved across a set of physical hosts and then go on to a high-level description of how a migration progresses.

3.1 Migrating Memory

Moving the contents of a VM's memory from one physical host to another can be approached in any number of ways. However, when a VM is running a live service it is important that this transfer occurs in a manner that balances the requirements of minimizing both *downtime* and *total migration time*. The former is the period during which the service is unavailable due to there being no currently executing instance of the VM; this period will be directly visible to clients of the VM as service interruption. The latter is the duration between when migration is initiated and when the original VM may be finally discarded and, hence, the source host may potentially be taken down for maintenance, upgrade or repair.

It is easiest to consider the trade-offs between these requirements by generalizing memory transfer into three phases:

Push phase The source VM continues running while certain pages are pushed across the network to the new destination. To ensure consistency, pages modified during this process must be re-sent.

Stop-and-copy phase The source VM is stopped, pages are copied across to the destination VM, then the new VM is started.

Pull phase The new VM executes and, if it accesses a page that has not yet been copied, this page is faulted in ("pulled") across the network from the source VM.

Although one can imagine a scheme incorporating all three phases, most practical solutions select one or two of the three. For example, *pure stop-and-copy* [3, 4, 5] involves halting the original VM, copying all pages to the destination, and then starting the new VM. This has advantages in terms of simplicity but means that both downtime and total migration time are proportional to the amount of physical memory allocated to the VM. This can lead to an unacceptable outage if the VM is running a live service.

Another option is *pure demand-migration* [16] in which a short stop-and-copy phase transfers essential kernel data structures to the destination. The destination VM is then started, and other pages are transferred across the network on first use. This results in a much shorter downtime, but produces a much longer total migration time; and in practice, performance after migration is likely to be unacceptably degraded until a considerable set of pages have been faulted across. Until this time the VM will fault on a high proportion of its memory accesses, each of which initiates a synchronous transfer across the network.

The approach taken in this paper, *pre-copy* [11] migration, balances these concerns by combining a bounded iterative push phase with a typically very short stop-and-copy phase. By 'iterative' we mean that pre-copying occurs in *rounds*, in which the pages to be transferred during round n are those that are modified during round $n-1$ (all pages are transferred in the first round). Every VM will have some (hopefully small) set of pages that it updates very frequently and which are therefore poor candidates for pre-copy migration. Hence we bound the number of rounds of pre-copying, based on our analysis of the *writable working set* (WWS) behavior of typical server workloads, which we present in Section 4.

Finally, a crucial additional concern for live migration is the impact on active services. For instance, iteratively scanning and sending a VM's memory image between two hosts in a cluster could easily consume the entire bandwidth available between them and hence starve the active services of resources. This *service degradation* will occur to some extent during any live migration scheme. We address this issue by carefully controlling the network and CPU resources used by the migration process, thereby ensuring that it does not interfere excessively with active traffic or processing.

3.2 Local Resources

A key challenge in managing the migration of OS instances is what to do about resources that are associated with the physical machine that they are migrating away from. While memory can be copied directly to the new host, connections to local devices such as disks and network interfaces demand additional consideration. The two key problems that we have encountered in this space concern what to do with network resources and local storage.

DOCKET ALARM Find authenticated court documents without watermarks at docketalarm.com.

For network resources, we want a migrated OS to maintain all open network connections without relying on forwarding mechanisms on the original host (which may be shut down following migration), or on support from mobility or redirection mechanisms that are not already present (as in [6]). A migrating VM will include all protocol state (e.g. TCP PCBs), and will carry its IP address with it.

To address these requirements we observed that in a cluster environment, the network interfaces of the source and destination machines typically exist on a single switched LAN. Our solution for managing migration with respect to network in this environment is to generate an unsolicited ARP reply from the migrated host, advertising that the IP has moved to a new location. This will reconfigure peers to send packets to the new physical address, and while a very small number of in-flight packets may be lost, the migrated domain will be able to continue using open connections with almost no observable interference.

Some routers are configured not to accept broadcast ARP replies (in order to prevent IP spoofing), so an unsolicited ARP may not work in all scenarios. If the operating system is aware of the migration, it can opt to send directed replies only to interfaces listed in its own ARP cache, to remove the need for a broadcast. Alternatively, on a switched network, the migrating OS can keep its original Ethernet MAC address, relying on the network switch to detect its move to a new port[1].

In the cluster, the migration of storage may be similarly addressed: Most modern data centers consolidate their storage requirements using a network-attached storage (NAS) device, in preference to using local disks in individual servers. NAS has many advantages in this environment, including simple centralised administration, widespread vendor support, and reliance on fewer spindles leading to a reduced failure rate. A further advantage for migration is that it obviates the need to migrate disk storage, as the NAS is uniformly accessible from all host machines in the cluster. We do not address the problem of migrating local-disk storage in this paper, although we suggest some possible strategies as part of our discussion of future work.

3.3 Design Overview

The logical steps that we execute when migrating an OS are summarized in Figure 1. We take a conservative approach to the management of migration with regard to safety and failure handling. Although the consequences of hardware failures can be severe, our basic principle is that safe migration should at no time leave a virtual OS more exposed

[1]Note that on most Ethernet controllers, hardware MAC filtering will have to be disabled if multiple addresses are in use (though some cards support filtering of multiple addresses in hardware) and so this technique is only practical for switched networks.

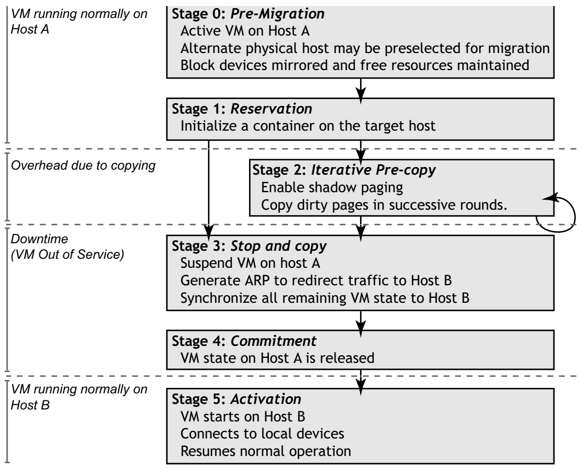
VM running normally on Host A
Stage 0: Pre-Migration
Active VM on Host A
Alternate physical host may be preselected for migration
Block devices mirrored and free resources maintained
Stage 1: Reservation
Initialize a container on the target host
Overhead due to copying
Stage 2: Iterative Pre-copy
Enable shadow paging
Copy dirty pages in successive rounds.
Downtime (VM Out of Service)
Stage 3: Stop and copy
Suspend VM on host A
Generate ARP to redirect traffic to Host B
Synchronize all remaining VM state to Host B
Stage 4: Commitment
VM state on Host A is released
VM running normally on Host B
Stage 5: Activation
VM starts on Host B
Connects to local devices
Resumes normal operation

Figure 1: Migration timeline

to system failure than when it is running on the original single host. To achieve this, we view the migration process as a transactional interaction between the two hosts involved:

Stage 0: Pre-Migration We begin with an active VM on physical host A. To speed any future migration, a target host may be preselected where the resources required to receive migration will be guaranteed.

Stage 1: Reservation A request is issued to migrate an OS from host A to host B. We initially confirm that the necessary resources are available on B and reserve a VM container of that size. Failure to secure resources here means that the VM simply continues to run on A unaffected.

Stage 2: Iterative Pre-Copy During the first iteration, all pages are transferred from A to B. Subsequent iterations copy only those pages dirtied during the previous transfer phase.

Stage 3: Stop-and-Copy We suspend the running OS instance at A and redirect its network traffic to B. As described earlier, CPU state and any remaining inconsistent memory pages are then transferred. At the end of this stage there is a consistent suspended copy of the VM at both A and B. The copy at A is still considered to be primary and is resumed in case of failure.

Stage 4: Commitment Host B indicates to A that it has successfully received a consistent OS image. Host A acknowledges this message as commitment of the migration transaction: host A may now discard the original VM, and host B becomes the primary host.

Stage 5: Activation The migrated VM on B is now activated. Post-migration code runs to reattach device drivers to the new machine and advertise moved IP addresses.

DOCKET ALARM Find authenticated court documents without watermarks at docketalarm.com.

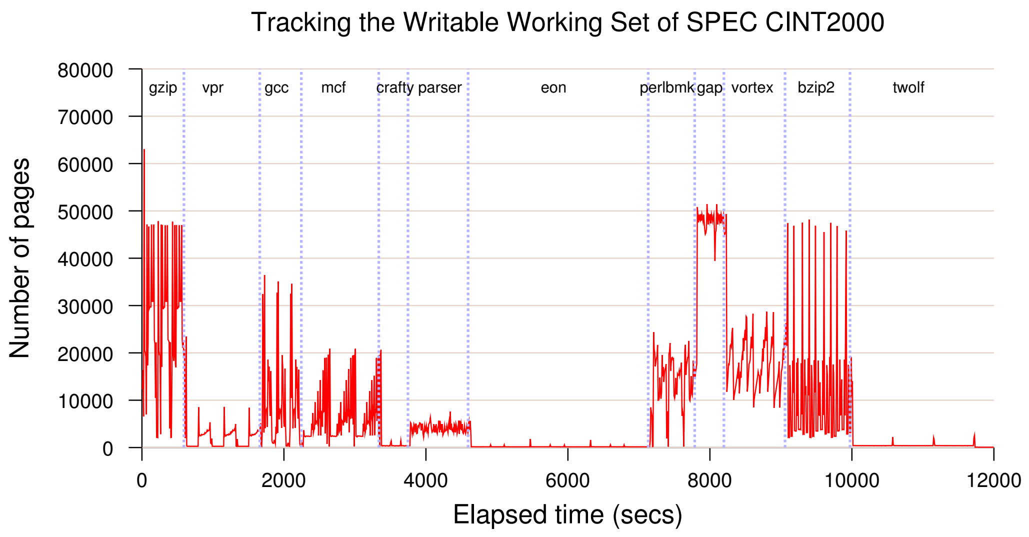

Figure 2: WWS curve for a complete run of SPEC CINT2000 (512MB VM)

This approach to failure management ensures that at least one host has a consistent VM image at all times during migration. It depends on the assumption that the original host remains stable until the migration commits, and that the VM may be suspended and resumed on that host with no risk of failure. Based on these assumptions, a migration request essentially attempts to move the VM to a new host, and on any sort of failure execution is resumed locally, aborting the migration.

4 Writable Working Sets

When migrating a live operating system, the most significant influence on service performance is the overhead of coherently transferring the virtual machine's memory image. As mentioned previously, a simple stop-and-copy approach will achieve this in time proportional to the amount of memory allocated to the VM. Unfortunately, during this time any running services are completely unavailable.

A more attractive alternative is pre-copy migration, in which the memory image is transferred while the operating system (and hence all hosted services) continue to run. The drawback however, is the wasted overhead of transferring memory pages that are subsequently modified, and hence must be transferred again. For many workloads there will be a small set of memory pages that are updated very frequently, and which it is not worth attempting to maintain coherently on the destination machine before stopping and copying the remainder of the VM.

The fundamental question for iterative pre-copy migration is: how does one determine when it is time to stop the pre-copy phase because too much time and resource is being wasted? Clearly if the VM being migrated never modifies memory, a single pre-copy of each memory page will suffice to transfer a consistent image to the destination. However, should the VM continuously dirty pages faster than the rate of copying, then all pre-copy work will be in vain and one should immediately stop and copy.

In practice, one would expect most workloads to lie somewhere between these extremes: a certain (possibly large) set of pages will seldom or never be modified and hence are good candidates for pre-copy, while the remainder will be written often and so should best be transferred via stop-and-copy – we dub this latter set of pages the *writable working set* (WWS) of the operating system by obvious extension of the original working set concept [17].

In this section we analyze the WWS of operating systems running a range of different workloads in an attempt to obtain some insight to allow us build heuristics for an efficient and controllable pre-copy implementation.

4.1 Measuring Writable Working Sets

To trace the writable working set behaviour of a number of representative workloads we used Xen's shadow page tables (see Section 5) to track dirtying statistics on all pages used by a particular executing operating system. This allows us to determine within any time period the set of pages written to by the virtual machine.

Using the above, we conducted a set of experiments to sam-

DOCKET ALARM
Find authenticated court documents without watermarks at docketalarm.com.

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS

Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS

Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS

Sync your system to PACER to automate legal marketing.