2010 18th Annual IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems

Predicting the Performance of Virtual Machine Migration

Sherif Akoush, Ripduman Sohan, Andrew Rice, Andrew W. Moore and Andy Hopper
University of Cambridge Computer Laboratory
firstname.lastname@cl.cam.ac.uk

***Abstract*—With the ability to move virtual machines between physical hosts, live migration is a core feature of virtualisation. However for migration to be useful, deployable feature on a large (datacentre) scale, we need to predict migration times with accuracy. In this paper, we characterise the parameters affecting live migration with particular emphasis on the Xen virtualisation platform. We discuss the relationships between the important parameters that affect migration and highlight how migration performance can vary considerably depending on the workload. We further provide 2 simulation models that are able to predict migration times to within 90% accuracy for both synthetic and real-world benchmarks.**

I. Introduction

Virtualisation has become a core facility in modern computing installations. It provides opportunities for improved efficiency by increasing hardware utilisation and application isolation [1] as well as simplifying resource allocation and management. One key feature that makes virtualisation attractive is that of live migration.

Live migration platforms (e.g. XenMotion [2] and VMotion [3]) allow administrators to move running virtual machines (VMs) seamlessly between physical hosts. This is of particular benefit for service providers hosting high availability applications. The level of service to which a service provider commits when hosting and running an application is described by a Service Level Agreement (SLA) and is typically couched in terms of application availability. Policies such as 99.999% availability (common in the telecommunication industry) permit only 5 minutes of downtime a year. Routine activities such as restarting a machine for hardware maintenance are extremely difficult under such a regime and so service providers invest considerable resources in high-availability, fault-tolerant systems. Live migration mitigates this problem by allowing administrators to move VMs with little interruption. This permits regular maintenance of the physical hardware, supports dynamic reconfiguration enabling cluster compaction in times of low demand, and shifts computing load to manage cooling *hotspots* within a datacentre [4].

However, short interruptions of service are still unavoidable during live migration due to the overheads of moving the running VM. Previous studies have demonstrated that this can vary considerably between applications due to different memory usage patterns, ranging from 60 milliseconds when migrating a Quake game server [5] to 3 seconds in the case of High-performance computing benchmarks [6]. This variation means that predicting the duration of any interruption is necessary on a per-workload basis in order to effectively plan resource allocation and maintenance schedules.

In this paper we examine migration times in detail and propose models which can be used to make workload specific predictions of service interruptions. In Section II we discuss live migration approaches with detailed emphasis on the Xen migration architecture. In Section III we investigate migration performance and confirm that the page dirty rate and link speed are the significant determining factors. We quantitatively show how particular combinations of these factors can markedly extend migration time and VM unavailability. Finally, in Section IV we utilise our measurements to develop and validate 2 simulation models which are useful for predicting the total migration and service interruption times when migrating any workload.

II. Live Migration of VMs

Live migration is a technology with which an entire *running* VM is moved from one physical machine to another. Migration at the level of an entire VM means that active memory and execution state are transferred from the source to the destination. This allows seamless movement of online services without requiring clients to reconnect. As migration designs to-date require universally accessible network attached storage (NAS), migrations are reduced to copying in-memory state and CPU registers. On migration completion, virtual I/O devices are disconnected form the source and re-connected on the destination physical host.

There are several techniques for live migration that trade-off two important parameters—*total migration time* and *downtime*. Total migration time refers to the total time required to move the VM between physical hosts while downtime is the portion of that time when the VM is not running.

Pure *stop-and-copy* [7], [8], [9] designs halt the original VM and copy its entire memory to the destination. This technique minimises total migration time but suffers from high downtime as the VM is suspended during the entire transfer. Pure *on-demand* [10] migration on the other hand operates by stopping the VM to copy only essential kernel data to the destination. The remainder of the VM address space is transferred when it is accessed at the destination. While this technique has a very short downtime, it suffers from high total migration time.

It has previously been established that both stop-and-copy and on-demand migrations have poor performance [5]. The former may lead to significant service disruption especially

1526-7539/10 $26.00 © 2010 IEEE
DOI 10.1109/MASCOTS.2010.13

if the VM is running highly used applications while the latter incurs a longer total migration time and degraded performance during the synchronisation of on-demand pages between hosts.

A. Pre-copy Migration

Pre-copy migration tries to tackle problems associated with earlier designs by combining a bounded iterative *push* step with a final and typically very short *stop-and-copy* phase [5].

The core idea of this design is that of iterative convergence. It involves iterating through multiple rounds of copying in which the VM memory pages that have been modified during the previous copy are resent to the destination on the assumption that at some point the number of modified pages will be small enough to halt the VM temporarily, copy the (small number of) remaining pages across, and restart it on the destination host. Such a design minimises both total migration time and downtime.

Pre-copy migration involves 6 stages [5], namely:

1) *Initialisation*: a target is pre-selected for future migration.
2) *Reservation*: resources at the destination host are reserved.
3) *Iterative pre-copy*: pages modified during the previous iteration are transferred to the destination. The entire RAM is sent in the first iteration.
4) *Stop-and-copy*: the VM is halted for a final transfer round.
5) *Commitment*: the destination host indicates that it has received successfully a consistent copy of the VM.
6) *Activation*: resources are re-attached to the VM on the destination host.

Unless there exist stop conditions, the iterative pre-copy stage may continue indefinitely. Thus, the definition of stop conditions is critical in terminating this stage in a timely manner. These conditions are usually highly dependent on the design of both the hypervisor and the live migration sub-system but are generally defined to reduce the amount of data copied between physical hosts while minimising VM downtime. However, the existence of these stop conditions has a significant effect on migration performance and may cause non-linear trends in the total migration time and downtime experienced by VMs (as will be shown in Section III).

B. Defining Migration Performance

Migration performance may be evaluated by measuring total migration time and total downtime. The former is the period when state on both machines is synchronised which may affect reliability while the latter is the duration in which the VM is suspended thus seen by clients as service outage.

Using the pre-copy migration model, total migration time may be defined as the sum of the time spent on all 6 migration stages (Equation 1) from initialisation at the source through to activation at the destination. Total downtime, however, is the time required for the final 3 stages to complete (Equation 2).

While it is expected that the iterative pre-copy stage will dominate total migration time, our measurements found that for certain classes of applications (specifically those that do not have a high memory page modification rate) the initialisation, reservation, commitment and activation stages may add a significant overhead to total migration time and downtime. We classify the initialisation and reservation stages together as *pre-migration* overhead while the commitment and activation stages define *post-migration* overhead.

$$\begin{aligned} TotalMigrationTime = & \underbrace{Initialisation + Reservation}_{Pre\text{-}migrationOverhead} \\ & + \sum_{i} Pre\text{-}copy_i + Stop\text{-}and\text{-}copy \\ & + \underbrace{Commitment + Activation}_{Post\text{-}migrationOverhead} \end{aligned} \quad (1)$$

$$\begin{aligned} TotalDowntime = & Stop\text{-}and\text{-}copy \\ & + \underbrace{Commitment + Activation}_{Post\text{-}migrationOverhead} \end{aligned} \quad (2)$$

C. Migration Bounds

Given the stop conditions, it is possible to work out the upper and lower migration performance bounds for a specific migration algorithm. We will use a real-world case to characterise these boundaries.

While there exist a range of live migration platforms, for the remainder of this paper we will base our analysis on the Xen migration platform. Xen is already being used as the basis for large scale cloud deployments [11] and thus this work would immediately benefit these deployments. Moreover, Xen is open-source allowing us to quickly and efficiently determine the migration sub-system design and implementation. Note however that our measurement techniques, methodology and prediction models design basis are applicable to any virtualisation platform that employs the pre-copy migration mechanism.

The stop conditions that are used in the Xen migration algorithm are defined as follows:

1) Less than 50 pages were dirtied during the last pre-copy iteration.
2) 29 pre-copy iterations have been carried out.
3) More than 3 times the total amount of RAM allocated to the VM has been copied to the destination host.

The first condition guarantees a short downtime as few pages are to be transferred. On the other hand, the other 2 conditions just force migration into the stop-and-copy stage which might still have many modified pages to be copied across resulting in large downtime.

1) Bounding Total Migration Time (Equation 3): Consider the case of an idle VM running no applications. In this case the iterative pre-copy stage will terminate after the first iteration as there is no memory difference. Consequently, the migration sub-system needs only to send the entire RAM in the first round. The total migration lower bound is thus the time required to send the entire RAM coupled with pre- and post-migration overheads.

On the other hand, consider the case where the entire memory pages are being modified as fast as link speed. In this scenario, the iterative pre-copy stage will be forced to terminate after copying more than 3 times the total amount of RAM allocated to the VM. Migration then re-sends the entire modified RAM during the stop-and-copy stage. The total migration upper bound is thus defined as the time required to send 5 times the VM size less 1 page[1] plus pre- and post-migration overheads.

$$Overheads + \frac{VMSize}{LinkSpeed} \leq TotalMigrationTime \leq Overheads + \frac{5 * VMSize - 1 * page}{LinkSpeed} \quad (3)$$

2) Bounding Total Downtime (Equation 4): Similarly, the total downtime lower bound is defined as the time required for just the post-migration overhead, assuming that the final stop-and-copy stage does not transfer any pages. This occurs either when the VM is idle or the link speed is fast enough to copy all dirtied pages in the pre-copy stage. On the other hand, the total downtime upper bound is defined as the time required to copy the entire RAM in the stop-and-copy stage coupled with the post-migration overhead.

$$Post\text{-}migrationOverhead \leq TotalDowntime \leq Post\text{-}migrationOverhead + \frac{VMSize}{LinkSpeed} \quad (4)$$

3) Difference in Bounds: Modelling bounds is useful as it enables us to reason about migration times provided that we know the link speed and VM memory size. These bounds are the limits in which the total migration time and downtime are guaranteed to lie. Given a 1,024 MB VM and 1 Gbps migration link, for example, the total migration time has a lower bound of 13 and upper bound of 50 seconds respectively. Similarly, the downtime has a lower bound of .314 and upper bound of 9.497 seconds respectively.

Table I lists the migration bounds for some common link speeds. While the downtime lower limit is fixed (as it is dependent purely on the post-migration overhead), all other bounds vary in accordance to the link speed due to their correlation with the VM memory size. As the table indicates, the bounds vary significantly. For bigger VM memory sizes (which is common in current installations [12]) we have even larger differences.

Thus, using bounds is at best an imprecise exercise and does not allow for accurate prediction of migration times. Building better predictions requires understanding the relationship between factors that impact migration performance. We address this in the next section.

[1] $\underbrace{\sum_{i=0}^{n-1} Pre\text{-}copy_i}_{3VMSize-1page} + \underbrace{Pre\text{-}copy_n}_{1VMSize} + \underbrace{Stop\text{-}and\text{-}copy}_{1VMSize}$

TABLE I
MIGRATION BOUNDS. MT: TOTAL MIGRATION TIME (SECONDS). DT: TOTAL DOWNTIME (MILLISECONDS). LB: LOWER BOUND. UB: UPPER BOUND. VM SIZE= 1,024 MB.

$Speed$	MT_{LB}	MT_{UB}	DT_{LB}	DT_{UB}
100 Mbps	95.3 s	459.1 s	314 ms	91,494.5 ms
1 Gbps	13.3 s	49.9 s	314 ms	9,497.8 ms
10 Gbps	5.3 s	10.1 s	314 ms	1,518.7 ms

III. PARAMETERS AFFECTING MIGRATION

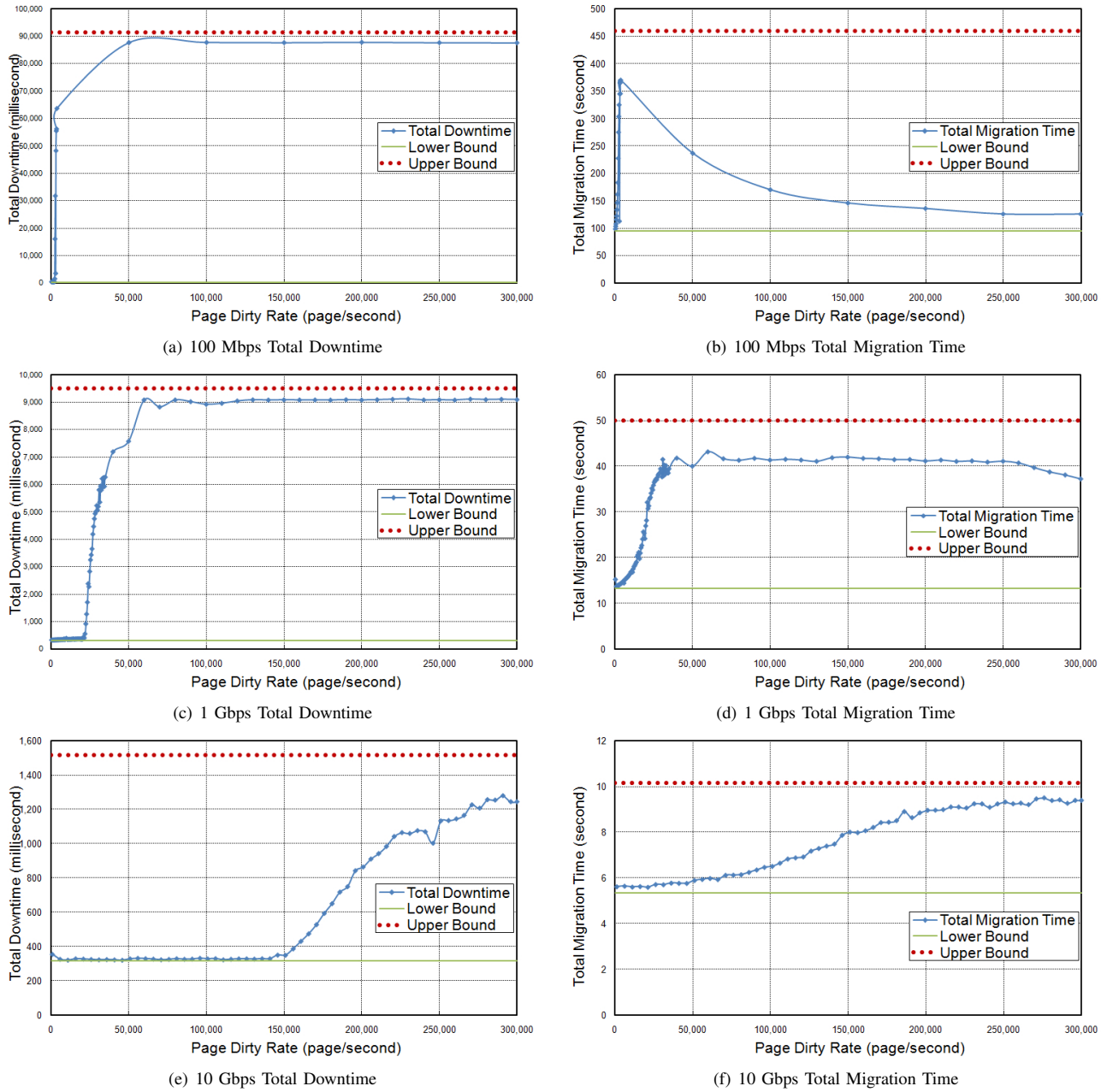
There are several factors that we need to study as a pre-requisite for accurate migration modelling. In this section, we explore these factors and their impact on total migration time and downtime. Moreover, stop conditions that may force migration to reach its final stages are generally what governs migration performance. Obviously, this is implementation specific which is exemplified by but not limited to the Xen support for live migration.

Migration link bandwidth is perhaps the most influential parameter on migration performance. Link capacity is inversely proportional to total migration time and downtime. Higher speed links allow faster transfers; thus migration requires less time to complete. Figure 1 illustrates migration performance for a 1,024 MB VM running a micro-benchmark that writes to memory pages with rates up to 300,000 pages/second on 100 Mbps, 1 Gbps and 10 Gbps links. It represents the impact of each link speed on total migration time and downtime. As link bandwidth increases, the point in the curve when migration performance starts to degrade rapidly shifts to the right roughly with the same ratio.

Page dirty rate is the rate at which memory pages in the VM are modified which, in turn, directly affects the number of pages that are transferred in each pre-copy iteration. Higher page dirty rates result in more data being sent per iteration which leads to longer total migration time. Furthermore, higher page dirty rates results in longer VM downtime as more pages need to be sent in the final transfer round in which the VM is suspended.

Figure 1 shows the effect of varying the page dirty rate on total migration time and downtime for each link speed. The relationship between page the dirty rate and migration performance is not linear because of the stop conditions defined in the algorithm. If the page dirty rate is below link capacity, the migration sub-system is able to transfer all modified pages in a timely fashion, resulting in a low total migration time and downtime. On the other hand, if the page dirty rate starts approaching link capacity, migration performance degrades significantly.

Total downtime at low page dirty rates is virtually constant and approximately equal to the lower bound (Equation 4). This is because the link has enough capacity to transfer dirty pages in successive iterations leading to a very short stop-and-copy stage. When the page dirty rate increases to the point that 29 iterations are not sufficient to ensure a short final copy round or when more than 3x the VM size have been transferred, migration is forced to enter its final stage with a large number



(a) 100 Mbps Total Downtime

(b) 100 Mbps Total Migration Time

(c) 1 Gbps Total Downtime

(d) 1 Gbps Total Migration Time

(e) 10 Gbps Total Downtime

(f) 10 Gbps Total Migration Time

Fig. 1. Effect of Link Bandwidth on Migration Performance. VM Size= 1,024 MB. Confidence intervals are omitted because of insufficient vertical resolution.

of dirty pages yet to be sent. Consequently, total downtime starts to increase in proportion to the increase in the number of modified pages that need to be transferred in the stop-and-copy stage. Total downtime further increases until the defined upper bound in which it has to send the entire VM memory.

Total migration time also increases with an increasing page dirty rate. This is attributable to the fact that more modified pages have to be sent in each pre-copy round. Moreover, the migration sub-system has to go through more iterations with the hope to have a short final stop-and-copy round. For page dirty rates near link speed, total migration time approaches its upper bound (Equation 3) as migration stops when 3x VM size has been transferred. Then, it starts to fall back towards its lower bound.

For *extremely* high page dirty rates (compared to link speed), migration is forced to reach its final transfer stage after 29 iterations having sent virtually no pages.[2] It then has to transfer the entire RAM in the final iteration. This is exemplified clearly for the 100 Mbps link (Figures 1(a) and 1(b)), in which total migration time drops back to its lower bound (almost all dirty pages are skipped in every iteration except the final one) while having total downtime at its upper bound (as the entire RAM has to be transferred in the stop-and-copy stage).

[2]A page is skipped if it is modified more than once in the same iteration.

The first pre-copy iteration tries to copy across the entire VM allocated memory. The duration of this first iteration is thus directly proportional to the **VM memory size** and subsequently impacts total migration time. On average, total migration time increases linearly with VM size. On the other hand, total downtime for low page dirty rates is almost the same regardless of the VM size as the migration sub-system succeeds in copying all dirtied pages between successive iterations resulting in a short stop-and-copy stage. When the link is unable to keep up with the page dirty rate, larger VMs suffer longer downtime (linearly proportional to the VM size) as there are more distinct physical pages that require copying in the stop-and-copy stage.

Pre- and post-migration overheads refer to operations that are not part of the actual transfer process. These are operations related to initialising a container on the destination host, mirroring block devices, maintaining free resources, reattaching device drivers to the new VM, and advertising moved IP addresses. As these overheads are static, they are significant especially with higher link speeds. For instance, pre-migration overhead constitutes around 77% of total migration time on a 10 Gbps link for a 512 MB idle VM. More importantly, post-migration overhead is an order of magnitude larger than the time required for the stop-and-copy stage.

To conclude this section, there are several parameters influencing migration performance. These parameters may be classified as having either a static or dynamic effect on migration performance. Parameters having static effect are considered as unavoidable migration overheads. On the other hand, parameters having dynamic effect on migration affect only the transfer process which are typically related to the VM specification and applications hosted inside it.

We show that the page dirty rate and link speed are the major factors defining migration times. We also show how particular combinations of these factors can extend expected total migration time and downtime. Finally, we observe that the pre- and post-migration overheads become significant compared to the iterative pre-copy and stop-and-copy stages, especially for VMs that have low page dirty rates and are being migrated over high speed links.

IV. Predicting Migration

We argued in Section II-C that the analytical bounds on migration are unacceptably wide. More accurate models are needed to help datacentre administrators intelligently provision and control their virtualised infrastructure. In Section III, we studied migration behaviour and the parameters impacting its performance. We now use these parameters to create accurate simulation models of the migration process.

We have implemented 2 simulation models, termed AVG (average page dirty rate) and HIST (history based page dirty rate), that may be used to predict pre-copy migration performance given all parameters affecting migration behaviour; namely the link speed, page dirty rate, VM memory size and migration overheads. The AVG model is based on pure simulation of the actual migration logic assuming a constant or average page dirty rate. It is useful when the page dirty rate of a given VM is fairly stable. On the other hand, the HIST model depends on a history log of page dirty rates that has been measured beforehand for a given time frame. It is used when the VM has a variable page dirty rate that cannot be approximated as an average.

Algorithm 1 The AVG Simulation Model

```
#These values are given
LINK_SPEED
PAGE_DIRTY_RATE
PRE_MIGRATION_OVERHEAD
POST_MIGRATION_OVERHEAD
VM_SIZE
#Start simulation
p2m_size = VM_SIZE / PAGE_SIZE
MAX_SENT = 80
MAX_ITERATIONS = 29
MAX_FACTOR = 3
MAX_BATCH_SIZE = 1,024
to_send ←ALL_PAGES
total_sent = 0
iteration = 0
last_iteration =False
loop
    iteration = iteration + 1
    N = 0
    while N ≤ p2m_size do
        sent_this_iteration = 0
        skip_this_iteration = 0
        if NOT last_iteration then
            to_skip =sim_peek()
        end if
        batch = 0
        for (batch ≤ MAX_BATCH_SIZE) AND (N ≤ p2m_size)) do
            if (NOT last_iteration) AND
            (N ∈ to_send) AND
            (N ∈ to_skip) then
                skip_this_iteration++
            end if
            if NOT ((N ∈ to_send AND N ∉ to_skip) OR (N ∈ to_send AND
            last_iteration)) then
                continue
            end if
            batch++
            N++
        end for
        migration_time = migration_time + batch / LINK_SPEED
        if last_iteration then
            downtime_time = downtime_time + batch / LINK_SPEED
        end if
        total_sent = total_sent + batch
        sent_this_iteration = batch
    end while
    if last_iteration then
        exit()
    end if
    if (iteration ≥ MAX_ITERATIONS) OR (total_sent > p2m_size *
    MAX_FACTOR) OR (sent_this_iteration + skip_this_iteration <
    MAX_SENT) then
        last_iteration =True
    end if
    to_send =sim_clean()
end loop


    total_migration_time =migration_time
                          +PRE_MIGRATION_OVERHEAD
                          +POST_MIGRATION_OVERHEAD
     total_downtime_time =downtime_time
                          +POST_MIGRATION_OVERHEAD
```

A. The AVG Simulation Model

The AVG simulation model assumes constant page dirty rates for applications that are running inside the VM. For certain workloads, it is sufficient to approximate the page dirty rate using an average. In those cases, the AVG model is useful in predicting total migration and downtime. Our AVG model simulation requires four input parameters: the link speed, average page dirty rate, VM memory size and migration overheads. The model is summarised in Algorithm 1.

Input parameters are easily obtained. The average page dirty rate is analytically determinable or measured using platform specific tools. The link speed is approximated using a standard throughput measurement tool (e.g. Iperf [13]) while the VM

memory size is a known value. Finally, pre- and post-migration overheads are determined by subtracting the time spent during the actual transfer from the total time required to migrate an idle VM.

The AVG (and the HIST) simulation model follow the core functionality of migration in Xen that is coded in `xc_domain_save.c` and `xc_domain_restore.c` files. The 2 key functions that we simulate are `sim_clean` and `sim_peek`. `sim_clean` returns the set of dirty pages and resets the state to *all clean*. `sim_peek` returns the dirty bitmap without resetting its state.

At each iteration in the iterative pre-copy stage, the simulated dirty bitmap is read and cleaned. Pages with the dirty bit set are candidates for transfer. We follow optimisations done in Xen: (1) pages that have been re-dirtied in the same iteration are skipped as they are likely to be transferred in the next iteration and (2) transfers occur in batches of 1,024 pages before the dirty bitmap is peeked again. Migration reaches the final stop-and-copy stage if conditions which force migration out of the iterative pre-copy stage are met. These conditions are illustrated in Section II-C.

Simulated total migration time is effectively the number of dirty pages (excluding re-dirtied pages in the same iteration) that should have been sent during the whole migration process divided by the available TCP bandwidth. We have also to account for migration overheads as they contribute to the migration time, especially for higher link speeds. On the other hand, simulated total downtime is the time required for the set of dirty pages that should have been sent in the stop-and-copy migration stage in addition to the post-migration overhead.

B. The HIST Simulation Model

In the cases where the AVG model is impractical or inadequate (e.g. where the page dirty rate is a function of time) we define the HIST simulation model. Our HIST model is a specialisation of the AVG model and is used for workloads that are approximately deterministic with similar behaviour between runs (e.g. a MapReduce workload).

The key idea is based on the observation that for deterministic processes the set of dirtied pages at any point in time will be approximately the same as for previous runs of the same workload running in a similar environment. Hence, for a migration initiated at time t, we are able to predict migration times based on a previously collected log of pages dirtied at time $t+1 \cdots t+N$. The HIST model is easily implemented by adjusting the `sim_clean` and `sim_peek` functions in Algorithm 1 to return the number of dirty pages at those points in time from the historical log.

C. Test-bed

Figure 2 illustrates the infrastructure used in our experiments. Citrix Xenserver 5.5.0 (Xen 3.3.1) is installed on 3 servers having each 2 Intel(R) Xeon(TM) E5506 2.13 GHZ CPUs, 6 GB DDR3 RAM, integrated dual Gigabit Ethernet (1 Intel PRO/1000PT PCI Express Single Port Desktop Adapter) and 1 PCI Express 8x slot. One of these servers is designated

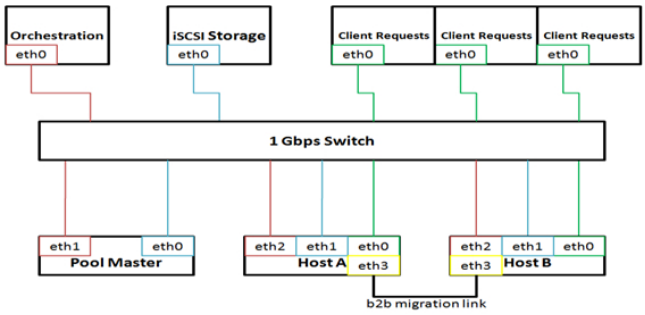


Fig. 2. Infrastructure

as the pool master while the others (Host A and B) are used for live migration runs.

The storage area network is configured using an IBM eserver xSeries 336 having Intel(R) Xeon(TM) X5470 3.00 GHZ CPU, 2 GB fully buffered DIMM modules, integrated dual Gigabit Ethernet and an Ultra320 SCSI controller. ISCSI support is provided by the Linux target framework (tgt) [14] installed on a Ubuntu server running the 2.6.27-7 kernel.

A number of client machines generate the load for SPECweb and SPECsfs while a desktop machine is used for automating the experiments and analysing the measurements. Three links running on separate subnets using a dedicated Netgear Gigabit switch provide management, storage and guest networking functions respectively. Migration is carried out over dedicated back-to-back connections between Host A and B.

D. Optimising Migration For 10 Gbps Links

We evaluate our prediction models by comparing simulation results with measurements obtained from actual migration runs executed over a pair of directly connected SolarFlare 10 Gbps network interconnects. However, our initial results showed that the default Xen migration platform is incapable of providing a migration throughput higher than 3.2 Gbps.

Profiling the migration sub-system highlighted a high overhead associated with mapping the guest domain (DomU) physical pages. The Xen migration sub-system carries out migration by continually mapping and un-mapping the physical pages of the migrating DomU in the control domain (Dom0) migration process. In doing this, Dom0 is able to access and send DomU page contents to the remote host.

The default Xen implementation maps and un-maps DomU pages in 4 MB segments (for batch transfer). These operations have a high temporal overhead due to the time required to setup and tear down page table entries in the migration process page table. As a result, Dom0 migration process spends a significant amount of time in page table management and is therefore unable to keep the link fully utilised.

We modified the Xen migration sub-system to eliminate this bottleneck by changing the design so that Dom0 migration process maps the *entire* guest VM physical address space at the start of the migration. Although the overhead of mapping the entire address space is in the order of hundreds of milliseconds

(as it is proportional to the VM size), this cost is amortised over the length of the migration process. We ensure we are able to map even the largest VMs by utilising a 64-bit Dom0.

Similarly, we observed that the per-packet overheads are high enough that utilising a larger maximum transmission unit (MTU) results in increased throughput. Thus, we modified the Xen migration platform to use *jumbo* (9 KB) frames for 10 Gbps links. As a result of these modifications, migration throughput increased to 7.12 Gbps, an enhancement of 122.5% over the original implementation.

E. Evaluation of the Simulation Models

In this section we provide results that evaluate the accuracy of both the AVG and HIST models on a bespoke page modification micro-benchmark designed to stretch migration performance, and industry-standard workloads. We use SPEC CPU [15] for CPU bound workloads, SPECweb [16] for web-server workloads, SPECsfs [17] for I/O intensive workloads and MapReduce tasks [18] for non-interactive workloads. All migrations are carried out on 10 Gbps links utilised the modifications outlined in Section IV-D.

1) Page Modification Micro-Benchmark: We required a deterministic application to measure the behaviour of the migration sub-system. For this purpose, we developed a synthetic userspace page modification micro-benchmark that writes to memory pages at fixed rates. This program has a resolution of 1 microsecond (determined by the `gettimeofday` system call), enabling a maximum page modification rate of 1 million pages/second.

While `gettimeofday` offers a resolution of 1 microsecond, implementation precisions varies. Therefore, we only use rates an order of magnitude larger in our experiments. We verified the accuracy of the benchmark by comparing the *set* rate with the *actual* rate computed from the dirty bitmap log.

At a given modification rate the program sequentially and circularly changes (writes to) the first byte of every memory page. The `valloc` library call is employed to ensure allocated memory is aligned to a page boundary.

We tested all the major parameters affecting migration performance identified in Section III. We performed more than 25,000 live migrations runs to evaluate and verify our 2 models. We have validated our prediction results with varying page modification rates (up to 300,000 pages per second), 3 link speeds (100 Mbps, 1 Gbps, and 10 Gbps) and 2 VM memory sizes (512 MB and 1,024 MB).

The AVG and HIST models are equally applicable to VMs running our page modification micro-benchmark. Because the program modifies memory pages at a fixed rate we can use this value as input to the AVG model. On the other hand, we are able to record page dirty bitmaps of the benchmark for use in the HIST model as well.

Figure 3 compares actual measurements with our predictions from the AVG and HIST models for different page modification rates up to 300,000 pages/second of a 1,024 MB VM migrated on 10 Gbps link. Points are averages of 15 measured data values while the solid line represents the simulation mean. Figure 3 illustrates that The AVG and HIST models closely follow the measured results. The AVG model has mean errors as low as 2.6% and 3.3% while the HIST model deviates from measured results by 3.3% and and 6.2% for total migration and total downtime respectively (Table II).

TABLE II
PREDICTION MEAN ERROR FOR DIFFERENT LINK SPEEDS AND VM SIZES. MT: TOTAL MIGRATION TIME. DT: TOTAL DOWNTIME.

$VMSize$	$Speed$	$Model$	MT_{err}	DT_{err}
1,024	100 Mbps	AVG	1.8%	7.5%
1,024	100 Mbps	HIST	3.5%	8.0%
1,024	1 Gbps	AVG	1.6%	9.3%
1,024	1 Gbps	HIST	2.5%	7.4%
1,024	10 Gbps	AVG	2.6%	3.3%
1,024	10 Gbps	HIST	3.3%	6.2%
512	10 Gbps	AVG	3.2%	7.1%
512	10 Gbps	HIST	3.8%	4.9%

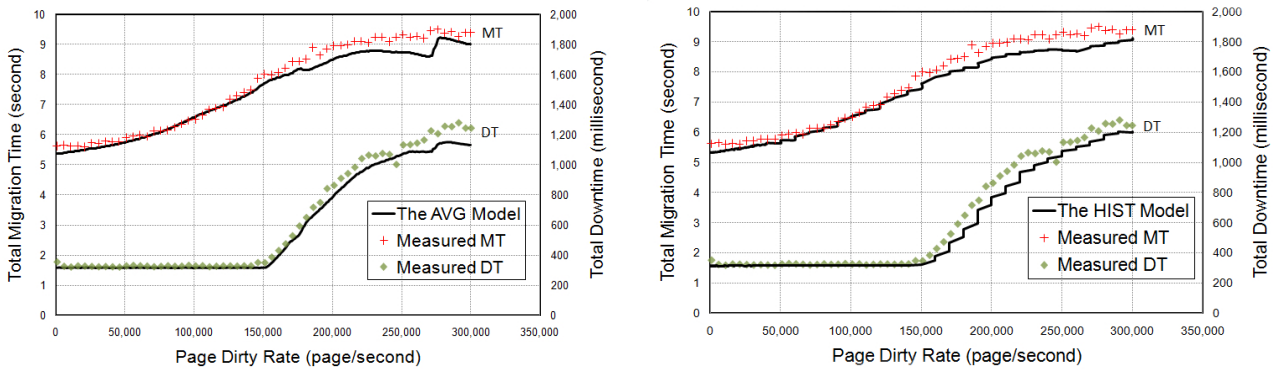
With regards to the link speed, we validated the AVG and HIST model predictions with 100 Mbps, 1 Gbps and 10 Gbps connections. We define link speed as the application level (migration) throughput or *goodput*, which is lower than the available hardware bandwidth due to protocol, OS and virtualisation overheads. We measure this obtainable goodput beforehand for use as input into our simulation. Our 2 models are accurate for all link speeds, as indicated by the low mean errors between the predicted and actual results provided in Table II.

So far we have shown that our models accurately predict migration performance for a 1,024 MB VM. Changing VM size to 512 MB, we verified the AVG and HIST models using a 10 Gbps link with varying page dirty rates. The AVG model has mean errors as low as 3.2% and 7.1% while the HIST model differs from measured results by 3.8% and 4.9% for total migration time and downtime respectively. Next, we examine the SPEC set of benchmarks.

2) SPEC CPU: We continued our evaluation by comparing the simulation predictions against a set of industry-standard workloads, specifically SPEC CPU, SPECweb and SPECsfs.

We begin by evaluating the CINT2000 (integer component of SPEC CPU2000). This benchmark stresses a system's processor, memory sub-system and compiler. CINT2000 is hosted inside a 1,024 MB guest VM that has been allocated 1 vCPU. It contains 12 applications written in C/C++. However, we omit details about individual application and only report average migration times for 15 entire runs of the benchmark. We evaluate the models by comparing predicted migration times with actual migration times (using the HIST model due to the non-uniform page dirty rates). We instrument migration to happen once during the run of each application.

3) SPECweb: Next, we evaluated a web workload, specifically SPECweb99. This benchmark comprises a webserver serving a complex mix of static and dynamic page (CGI scripts) requests among other features. We configured a VM to host the SPECweb server with 6 vCPUs and 1,024 MB RAM in order to have enough computing resources to saturate our client traffic links running at 100 Mbps. 3 separate client machines generate a load of 100 simultaneous connections to



(a) The AVG Model.

(b) The HIST Model.

Fig. 3. Evaluation for the AVG and HIST Models. Bandwidth= 10 Gbps. VM Size= 1,024 MB. MT: Total Migration Time. DT: Total Downtime. Confidence intervals are omitted because of insufficient vertical resolution.

TABLE III
INDUSTRY-STANDARD WORKLOADS. CPU: SPEC CPU. WEB: SPECWEB. SFS: SPECSFS. MR: MAPREDUCE TASKS. MT: TOTAL MIGRATION TIME (IN SECONDS). DT: TOTAL DOWNTIME (IN MILLISECONDS). BANDWIDTH= 10 GBPS. VM SIZE= 1,024 MB (FOR CPU AND WEB) AND 4,096 MB (FOR SFS AND MR). A: ACTUAL MEASUREMENT. P: THE HIST MODEL'S PREDICTION.

	MT_A	MT_P	Err	DT_A	DT_P	Err
CPU	5.8 s	5.7 s	2.4%	317.3 ms	314.1 ms	2.4%
WEB	7.5 s	7.4 s	2.0%	449.5 ms	420.4 ms	6.4%
SFS	14.8 s	14.9 s	1.5%	217.6 ms	217.7 ms	0.1%
MR	14.9 s	15.13 s	1.4%	348.9 ms	348.1 ms	0.2%

the webserver. We force a live migration every 2 minutes over the duration of 15 runs of SPECweb. Each run is 20 minutes long with an additional 5 seconds warmup time.

The page modification rates for SPEC CPU and SPECweb have previously been reported as being around 50,000 and 12,000 pages/second respectively [5]. A 10 Gbps migration link is capable of transferring approximately 250,000 pages/second. Due to the low page dirty rates the link is able to adequately transfer all dirty pages in each iteration.

Consequently, migration times are relatively constant. Table III shows the actual and modelled migration times for SPEC CPU and SPECweb. The HIST model is accurate for predicting migration performance as shown by the low mean error. SPECweb incurs a slightly higher error for downtime prediction due to the additional load on the network sub-system. We have noticed that this increased non-deterministically the time required to suspend and/or resume the VM during the final stages of migration.

4) SPECsfs: Next, we evaluate SPECsfs 3.0 (SFS97_R1). This benchmark measures NFS (version 3) file server throughput and response time for an increasing load of NFS operations (`lookup`, `read`, `write`, `getattr`, `readling`, `create`, `remove`, `fsstat`, `setattr`, `readdirplus`, `access`, and `commit`) against the server over file sizes ranging from 1 KB to 1 MB.

The NFS server runs in a guest VM with 4,096 MB RAM and 1 vCPU. We used 2 clients to generate an increasing load on the server, starting from 100 to 1,000 operations/second. Each run is 5 minutes long with an additional 5 seconds warmup time. We force a live migration every 2 minutes over the duration of the entire 10 runs.

The average measured total migration time and downtime are 14.9 seconds and 348 milliseconds respectively. We also tracked the total number of dirty pages for the benchmark run, which is illustrated in Figure 4. As the load increases, the dirty rate also increases. However, SPECsfs does not surpass 10,000 dirty pages/second, which is approximately 4% of the link capacity on a 10 Gbps network. Thus, load variation has negligible effects on migration results.

As the page dirty rates for the workload are not constant, we rely on the HIST model to simulate migration performance for SPECsfs. Predicted total migration time and total downtime differ from actual measurements by 1.5% and 0.1% respectively (Table III) which shows excellent accuracy for I/O bound workloads.

5) MapReduce Tasks: There is currently wide-spread interests in the MapReduce paradigm for large-scale data processing and analysis [19], [20]. This simple (but effective) model consists of two functions: *Map* and *Reduce*. The Map function reads, filters and/or transforms data from an input file, and then outputs a set of intermediate records. These intermediate records are typically split according to a hash function into disjoint buckets. The Reduce function processes or combines all intermediate records associated with the same hash value, and then writes new records to an output file. Programs written according to this model are automatically executed in parallel on a large scale cluster. MapReduce is used in web search, sorting, data mining, machine learning and many other systems [21].

We evaluate MapReduce tasks that involve HTML document processing [18]. The most popular open-source implementation of the MapReduce framework is the Hadoop system [22], which we use to run the tasks. We consider live migration of one node in the cluster. This node is configured to run both Map and Reduce functions. It has 4,096 MB RAM and 1 vCPU. We instrument live migration every 2 minutes over the course of 3 runs of all tasks. The average total migration time and downtime are 14.9 seconds and 348.9 milliseconds respectively. The total number of

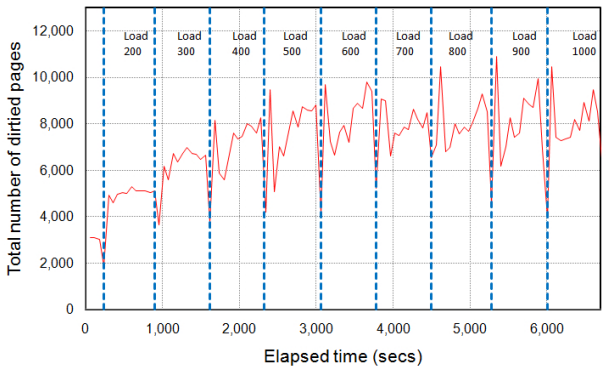


Fig. 4. Tracking the Total Number of Dirtied Pages for SPECsfs Workload

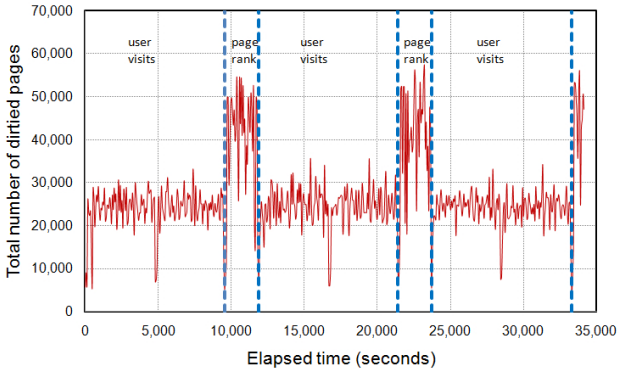


Fig. 5. Tracking the Total Number of Dirtied Pages for MapReduce Workload

dirty pages for 3 executions is shown in Figure 5. Tasks related to computing the "page rank" modify memory pages with a rate that is double that of "user visits" tasks. This is because "page rank" involves a complex calculation on two data sets. In all cases though, the page dirty rate is hardly over 50,000 pages/seconds (≃200 MB/s), which is approximately 20% of the link capacity. Consequently, migration times are approximately constant.

We use the HIST model to predict migration performance for a node running MapReduce tasks. The model is able to achieve good accuracy with less than 2% error (Table III) between actual and predicted migration results. For long running background MapReduce workloads, our simulation is a close predictor.

6) Lower Link Speeds: 10 Gbps link speed is too fast to show variability in migration times for MapReduce tasks despite having different page modification rates. By doing migration however on slower links we expect to get variable migration results dependant on the current page dirty rate. In Table IV we report minimum and maximum time values for migrating MapReduce tasks on 100 Mbps link. Results illustrate that a specific workload can have different migration times if the link cannot send modified pages fast enough. We also show the HIST model predictions which illustrate that our simulation can capture variability in migration performance.

TABLE IV
MIGRATION ON 100 MBPS LINK FOR MAPREDUCE TASKS. MT: TOTAL MIGRATION TIME (IN SECONDS). DT: TOTAL DOWNTIME (IN MILLISECONDS). VM SIZE= 4,096 MB. A: ACTUAL MEASUREMENT. P: THE HIST MODEL'S PREDICTION.

	MT_A	MT_P	DT_A	DT_P
Min	525.82 s	567.80 s	629.00 ms	499.61 ms
Max	1,109.88 s	1,219.37 s	42,534.00 ms	45,431.12 ms

In other words, the HIST model provides bounds in which migration times are expected to lie. Furthermore, we can use the HIST model to decide when it is best to do migration that will result in minimum total migration time or/and downtime.

V. RELATED WORK

Live migration performance under Xen has been studied before on various workloads such as simple webservers, SPECweb99, low latency Quake 3 servers, up to "diabolical" memory bound workloads [5]. We extend this work by characterising the effects of the link speed, page dirty rate and VM memory size. We also analyse SPECsfs and MapReduce tasks as additional workloads.

The REMUS project provides a high degree of fault tolerance using asynchronous virtual machine replication [23]. It documented enhancements to the migration code to optimise the final stop-and-copy stage. REMUS implements a quick filter of clean pages and maps the guest domain's entire physical memory to reduce mapping overhead. We follow similar optimisations to minimise migration overheads and increase throughput on 10 Gbps links but differ in doing this for the entire migration process.

An automatic and transparent mechanism for proactive fault tolerance for arbitrary MPI applications has been studied and implemented using Xen live migration [6]. In their research, the authors give a general overview on total migration time and possible parameters affecting it, specifically the amount of memory allocated to a guest VM. We provide a more detailed study on all parameters affecting migration and their influence on migration performance.

VI. FUTURE WORK

The experiments that we have carried out prove that the migration link speed is the most influential parameter on performance. We have been working on local area networks assuming live migration inside one datacentre. However, moving workloads between different datacentres, especially for cloud providers, is also useful. Providers might want to balance loads over different datacentres, move resources geographically closer to clients for better response time, implement disaster recovery capabilities at another site, or shift computations to areas where energy is available and cheap. For these cases, the link latency has a significant effect on migration performance due to geographical limitations. Consequently the models outlined in this work become key instruments in planning and efficiently scheduling migrations. We plan to further utilise the models to study migration behaviour on wide area networks.

As part of our research into Computing for the Future of the Planet we are working on a new computing framework that will automatically relocate workloads to chase surplus energy, which would otherwise be wasted [24]. Energy is a significant financial and environmental cost for datacentre operators. Our framework will migrate computing jobs to the locations of energy sources whilst maintaining service level agreement. We believe this can be beneficial in absorbing intermittent and distributed generation from renewable energy sources in addition to exploiting geographic and temporal variation in electricity prices [25]. Predicting the cost of migration is an integral part to making an informed decision to where a workload is relocated.

VII. Conclusions

In this paper, we studied live migration behaviour in pre-copy migration architectures, specifically using the Xen virtualisation platform. We show that the link speed and page dirty rate are the major factors impacting migration behaviour. These factors have a non-linear effect on migration performance largely because of the hard stop conditions that force migration to its final stop-and-copy stage.

We show that the Xen migration architecture does not scale up well with high speed (10 Gbps) links. We implemented several optimisations that increased migration throughput by 125.5% (from 3.2 Gbps to 7.12 Gbps). To the best of our knowledge, this is the first study and characterisation of pre-copy migration performance on 10 Gbps links.

Accurate prediction of migration performance is the objective of this work. Datacentre administrators need to provision and control computing capacity in order to guarantee certain performance levels that do not violate service level agreement. Otherwise, customers are unsatisfied and penalties have to be paid. In a virtualised environment, administrators can dynamically change VM placements in order to plan maintenance, balance loads or save energy. Live migration is the tool used. Migration times should be accurately predicted to enable more dynamic and intelligent placements of VMs without degrading performance.

To provide for this requirement we introduced 2 migration simulation models based on the average rate at which memory pages are dirtied in the VM and based on previous observations of the rate at which the pages are modified. We validated and verified both models showing that they are accurate to more than 90% of actual results.

Acknowledgment

We are grateful to Steven Hand for his useful information about the live migration under Xen. We would like also to thank Kieran Mansley and Solarflare Communications for their generous support of our activities and loan of assorted NICs.

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," in *Proc. ACM Symposium on Operating Systems Principles (SOSP'03)*, New York, NY, USA, 2003, pp. 164–177.

[2] XenMotion. Citrix Systems, Inc. [Online]. Available: http://www.xenserver5.com/xenmotion.php

[3] VMotion. VMware, Inc. [Online]. Available: http://www.vmware.com/products/vmotion/

[4] T. Wood, P. J. Shenoy, A. Venkataramani, and M. S. Yousif, "Black-box and gray-box strategies for virtual machine migration," in *Proc. USENIX Symposium on Networked Systems Design and Implementation (NSDI'07)*, Berkeley, CA, USA, 2007, pp. 229–242.

[5] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines," in *Proc. USENIX Symposium on Networked Systems Design and Implementation (NSDI'05)*, Berkeley, CA, USA, 2005, pp. 273–286.

[6] A. B. Nagarajan, F. Mueller, C. Engelmann, and S. L. Scott, "Proactive fault tolerance for HPC with Xen virtualization," in *Proc. ACM Annual International Conference on Supercomputing (ICS'07)*, New York, NY, USA, 2007, pp. 23–32.

[7] M. Kozuch and M. Satyanarayanan, "Internet suspend/resume," in *Proc. IEEE Workshop on Mobile Computing Systems and Applications (WMCSA'02)*, Washington, DC, USA, 2002, pp. 40–46.

[8] C. P. Sapuntzakis, R. Chandra, B. Pfaff, J. Chow, M. S. Lam, and M. Rosenblum, "Optimizing the migration of virtual computers," *ACM SIGOPS Oper. Syst. Rev.*, vol. 36, no. SI, pp. 377–390, 2002.

[9] A. Whitaker, R. S. Cox, M. Shaw, and S. D. Gribble, "Constructing services with interposable virtual hardware," in *Proc. USENIX Symposium on Networked Systems Design and Implementation (NSDI'04)*, Berkeley, CA, USA, 2004, pp. 169–182.

[10] E. Zayas, "Attacking the process migration bottleneck," *ACM SIGOPS Oper. Syst. Rev.*, vol. 21, no. 5, pp. 13–24, 1987.

[11] Amazon Elastic Compute Cloud (Amazon EC2). Amazon Web Services LLC. [Online]. Available: http://aws.amazon.com/ec2/

[12] S. Hacking and B. Hudzia, "Improving the live migration process of large enterprise applications," in *Proc. ACM International Workshop on Virtualization Technologies in Distributed Computing (VTDC'09)*, New York, NY, USA, 2009, pp. 51–58.

[13] Iperf. The National Laboratory for Applied Network Research. [Online]. Available: http://sourceforge.net/projects/iperf/

[14] F. Tomonori and M. Christie. Linux SCSI target framework. [Online]. Available: http://stgt.berlios.de/

[15] SPEC CPU2000. Standard Performance Evaluation Corporation. [Online]. Available: http://www.spec.org/cpu2000/CINT2000/

[16] SPECweb99. Standard Performance Evaluation Corporation. [Online]. Available: http://www.spec.org/web99/

[17] SPEC SFS97. Standard Performance Evaluation Corporation. [Online]. Available: http://www.spec.org/sfs97r1/

[18] A. Pavlo, E. Paulson, A. Rasin, D. J. Abadi, D. J. DeWitt, S. Madden, and M. Stonebraker, "A comparison of approaches to large-scale data analysis," in *Proc. ACM International Conference on Management of Data (SIGMOD'09)*, New York, NY, USA, 2009, pp. 165–178.

[19] D. A. Patterson, "Technical perspective: the data center is the computer," *ACM Commun.*, vol. 51, no. 1, pp. 105–105, 2008.

[20] U. Höelzle and L. A. Barroso, *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Morgan and Claypool Publishers, 2009.

[21] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *ACM Commun.*, vol. 51, no. 1, pp. 107–113, 2008.

[22] Hadoop. The Apache Software Foundation. [Online]. Available: http://hadoop.apache.org/

[23] B. Cully, G. Lefebvre, D. Meyer, M. Feeley, N. Hutchinson, and A. Warfield, "REMUS: high availability via asynchronous virtual machine replication," in *Proc. USENIX Symposium on Networked Systems Design and Implementation (NSDI'08)*, Berkeley, CA, USA, 2008, pp. 161–174.

[24] A. Hopper and A. Rice, "Computing for the future of the planet," *Philosophical Transactions of the Royal Society*, vol. 366, no. 1881, pp. 3685–3697, 2008.

[25] A. Qureshi, R. Weber, H. Balakrishnan, J. Guttag, and B. Maggs, "Cutting the electric bill for internet-scale systems," in *Proc. ACM Conference on Data Communication (SIGCOMM'09)*, New York, NY, USA, 2009, pp. 123–134.