Pacer: Taking the Guesswork Out of Live Migrations in Hybrid Cloud Computing

Jie Zheng† T. S. Eugene Ng† Kunwadee Sripanidkulchai* Zhaolei Liu†
Rice University† NECTEC, Thailand*

Abstract

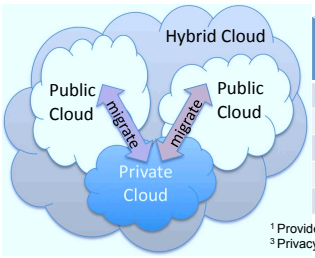
Hybrid cloud computing, where private and public cloud resources are combined and applications can migrate freely, ushers in unprecedented flexibility for businesses. To unleash the benefits, commercial products already enable the live migration of full virtual machines (CPU, memory, disk, network) between distant cloud datacenters.

Unfortunately, no *live migration progress management system* exists, leading to (1) guesswork over how long a migration might take and the inability to schedule dependent tasks accordingly; (2) unacceptable application degradations – application components could become split over distant cloud datacenters for an arbitrary period during migration; (3) inability to balance application performance and migration time – e.g. to finish migration later for less performance interference.

Pacer is the first migration progress management system. Pacer's techniques are based on robust and lightweight run-time measurements of system and workload characteristics, novel efficient and accurate analytic models for progress predictions, and online adaptation to maintain user-defined migration objectives for coordinated and timely migrations.

1 Introduction

Hybrid cloud computing (HCC) – where virtualizable compute and storage resources from private datacenters and public cloud providers are seamlessly integrated into one platform in which applications can migrate freely – is emerging as the most preferred cloud computing paradigm for commercial enterprises according to recent industry reports and surveys [3, 5, 17]. This is not surprising since HCC combines the benefits of public and private cloud computing, resulting in unprecedented flexibility for CAPEX and OPEX savings, application adaptivity, disaster survivability, zero-downtime maintenance, and privacy control. An impressive array of commercial products that facilitates HCC is already available today; Table 1 highlights some examples.



Migration / Reason	Private to Public	Public to Private	Public to Public
Cost saving	✔	✔[1]	✔[1]
Adaptivity	✔	✔	✔
Survivability	✔	✔[2]	✔[2]
Maintenance	✔		
Privacy		✔[3]	✔[4]

[1] Provider price hike. [2] Provider discontinues service. [3] Privacy law change. [4] Provider privacy policy change.

Figure 1: Beneficial usage scenarios of HCC.

Figure 1 illustrates some beneficial usage scenarios of HCC. The ability to migrate applications freely among private and -public clouds, whether it is private-to-public, public-to-private, or -public-to-public, is a key to maximizing the benefits of HCC. Tried and true live virtual machine (VM) migration technologies are therefore central to the HCC paradigm. The migration in HCC is inter-datacenters, and consequently it typically requires the *full* migration of the VM, including virtual CPU, memory, disk storage, and network connection. Full migration is substantially more complex than the CPU/memory-only migration, which is commonly conducted in server cluster load-balancing scenarios. Fortunately, commercial products (Table 1) already enabled the full and live migration of VMs between distant cloud datacenters. In particular, F5 BigIP [11] leverages hypervisor support for live CPU/memory and storage migration, and automates the full migration of VMs between datacenters.

That is, however, not to say live migration in HCC is a solved problem. On the contrary, the use of live migration in HCC has exposed a fundamental weakness in existing solutions, namely the lack of *live migration progress management* – the ability to predict migration time,to coordinate the progress of multiple migrations, and to control the migration time.

Product category	Examples
Seamless private and public cloud networking	Amazon VPC, Rackspace RackConnect, Microsoft Windows Azure Virtual Network
Unified hybrid cloud management console	RedHat CloudForms, RightScale myCloud, VMware vCloud Connector
Datacenter-to-datacenter full VM live migration	F5 BigIP, VMware VM/Storage vMotion, KVM VM/block migration, Microsoft Hyper-V VM/storage migration

Table 1: Examples of commercial products that help make hybrid cloud computing a reality.

1.1 Lack of migration progress management

In traditional datacenters, CPU/memory-only migration is the most common migration use case. Since the migration is often conducted over a fast LAN and there is relatively little data to migrate, the migration typically completes in a few seconds, and there is very little need for migration progress management. In comparison, migration in HCC is over a much slower WAN and the volume of data to migrate is several orders of magnitude larger because of the virtual disk migration, and thus the migration in HCC thus could take thousands of seconds. In general, such a large migration time could happen whenever a very large amount of data needs to be migrated, even if the migration is within a datacenter LAN. At such a timescale, the migration progress management (MPM) is sorely missed, and the users resort to guessing on a variety of issues:

How long does migration take? – is a popular question in live VM migration FAQs [13, 16, 15]. Unfortunately, there is no simple formula for calculating the answer, because the finish time depends on many dynamic run-time variables that are not known before migration *a priori*. Those variables includes application I/O workload intensity, network speed, and disk speed. As numerous online forum discussions indicate (e.g. [21, 22, 23, 20, 27, 29, 30, 28]), users routinely try to guess why migration is slow and whether it could be sped up, and how long they might have to wait.

How to avoid application components getting split between distant datacenters during migration? – This issue is of paramount importance to enterprises, because their applications often consist of multiple interacting components that performs different functions (e.g. content generation, custom logic, data management, etc. [8]). Without the ability to manage migration progress, individual application components could finish the migration at very different time and become split over distant cloud datacenters for an arbitrarily long period. The resulting large inter-component communication delay guarantees detrimental performance impact. Intuitively, a stopgap method is to set the data migration speeds for different components to be proportional to their virtual memory/storage sizes. Unfortunately, the migration time depends on a large number of dynamic run-time variables, and this stopgap method is bound to fail.

How to control the trade-off between application performance and migration time? – is another popular question raised by users [25, 24]. Studies [26, 4, 2, 18] have shown that migration can degrade application performance, and slowing migration down helps [4]. Although the administrator might be willing to slow down the migration to some extent to improve application performance, a migration task must still be finished before a deadline or other dependent tasks cannot proceed. Unfortunately, no solution exists for managing migration progress to finish at a desired time. A stopgap method is to configure the data migration speed to $\frac{data\ size}{desired\ time}$. Again, this method is bound to fail due to the dynamic run-time variables (see Section 3).

1.2 Pacer

The contribution of this paper is Pacer – the first migration progress management system (MPMS). Pacer effectively addresses all three of the aforementioned issues. While much details of Pacer's constituent techniques will be discussed in Section 2. Those techniques share the following key strengths:

Robust and lightweight run-time measurements – Pacer is highly effective because it continuously measures the application I/O workload intensity (both memory and disk accesses) and the bottleneck migration speed (network or disk) at run-time. Furthermore, the implementation of these techniques minimizes the run-time overhead as shown in Section 3.

Novel efficient & accurate analytic models for predictions – Such analytic models are used for (1) predicting the amount of remaining data to be migrated as a function of the application's I/O workload characteristics and the migration progress, and (2) predicting the finish time of the migration (i.e. addressing the first MPM issue) as a function of the characteristics of each migration stage (i.e. disk, dirty blocks, CPU/memory, etc.).

Online adaptation – Addressing the second and third MPM issues requires certain migration objectives: the former requires multiple application components to complete migrations simultaneously; the latter requires a migration to complete at a specific time spot. No static migration settings can successfully meet such objectives due to run-time dynamics. Pacer continuously adapts to ensure that the objectives are met. In the former case, Pacer adapts the targeted migration finish time for all components given what is measured to be feasible. In the latter case, Pacer adapts the migration speed to main-

tain a targeted migration finish time in face of application dynamics.

1.3 Road map

The rest of this paper is organized as follows. Section 2 presents the constituent techniques in Pacer for migration progress management. Section 3 and 4 present experimental results demonstrating the capability and benefit of Pacer. Section 5 introduces how to apply Pacer in the concurrent migration scenario. We discuss related work in Section 6 and conclude in Section 7.

2 Design of Pacer

2.1 Overview

Pacer is designed for the pre-copy migration model, which is widely used in virtualization platforms KVM [9] and XEN [6]. A slightly different variant found in VMware [12, 1] will be discussed later. In the pre-copy model, which is implemented by KVM, the virtual disk migration is prior to the memory migration. During the virtual disk copy, all write operations to the disk are logged. The dirty blocks are retransmitted, and the new dirty blocks generated during that time period are again logged and retransmitted. The dirty block retransmission process repeats until the number of dirty blocks falls below a threshold, and then the memory migration begins.

As far as we know, there is no existing solution to quantify the migration time for each stage, especially for the dirty iteration. The problem is challenging for two reasons. Firstly, the total number of dirty pages/blocks is not known before migration completes, since that number depends on how the application accesses the memory and the storage (i.e., the application workload). Secondly, the migration speed is unknown due to the interference brought by the migrated VM's workload and other competing workloads, which share the network or disk with migration.

Pacer has two main functions: predicting migration time and best-effort time control.

2.2 Predicting migration time

Pacer performs prediction periodically (default configuration is every 5 seconds). To predict the remaining time during the migration, three things must be known(1) what operation is performed in each stage of migration, (2) how much data there is to migrate, (3) how fast is the migration progressing. The paper will address these three topics in the following sections.

2.2.1 Migration Time Model

The total migration time T can be modeled as four distinct parts: $t_{Precopy}$ for the pre-copy stage, $t_{DirtyIteration}$ for the period after pre-copy but before memory migration, t_{Memory} for the period from the beginning of the memory migration until the time the VM is suspended, and $T_{Downtime}$ for a small downtime needed to copy the remaining dirty blocks and dirty pages once they drop below a configured threshold:

$$T = t_{Precopy} + t_{DirtyIteration} + t_{Memory} + T_{Downtime} \quad (1)$$

In all formulas, variables in upper case either have given values or their values are provided by algorithms, which the paper will discuss later in Section 2.2.2. The variables in lower case are to be computed.

For the pre-copy phase, we have:

$$t_{Precopy} = \frac{DISK\_SIZE}{speed_{Precopy}} \quad (2)$$

where $DISK\_SIZE$ is the VM virtual disk size obtained directly from the VM configuration and $speed_{Precopy}$ is the migration speed for the pre-copy stage.

At the end of pre-copy, a set of dirty blocks need to be migrated. The amount is defined as $DIRTY\_SET\_SIZE$. This variable is crucial to the prediction accuracy during the dirty iteration. However, the exact value is unknown until the end of pre-copy phase. It is very challenging to know the dirty set size *ahead-of-time* while the migration is still in the pre-copy stage. The algorithm in Section 2.2.2 is the first to solve this problem.

In the dirty iteration, while dirty blocks are migrated and marked clean, the clean blocks may be overwritten concurrently and get dirty again. The number of blocks getting dirty per second is called the dirty rate. The dirty rate depends on the number of clean blocks (fewer clean blocks means fewer blocks can become dirty later) and the workload of the VM. Similar to the need for dirty set size prediction, we need an algorithm to predict the dirty rate (AVE_DIRTY_RATE) while migration is still in pre-copy. The time for dirty iteration is

$$t_{DirtyIteration} = \frac{DIRTY\_SET\_SIZE}{speed_{DirtyIteration} - AVE\_DIRTY\_RATE} \quad (3)$$

where $speed_{DirtyIteration}$ is the migration speed for the dirty iteration stage.

Memory migration typically behaves similarly to the storage migration dirty iteration. All memory pages are first marked dirty, then dirty pages are iteratively migrated and marked clean. Pages can become dirty again after being written. We propose an algorithm in Section 2.2.2 that is effective for predicting the average memory dirty rate ($AVE\_MEM\_DIRTY\_RATE$).

During the memory migration, different systems have different behaviors. For KVM, the VM still accesses storage in the source and disk blocks could get dirty during memory migration. Thus, in KVM, memory migration and storage dirty iteration may alternatively happen. Then, denoting the size of memory as MEM_SIZE and memory migration speed as $speed_{Memory}$, we have

$$t_{Memory} = MEM\_SIZE / (speed_{Memory} - AVE\_MEM\_DIRTY\_RATE - AVE\_DIRTY\_RATE) \quad (4)$$

Other variants: The previous derivation is based on a fair assumption that memory migration follows storage migration (KVM and XEN). The model can easily be adapted to other systems, e.g. VMware. Storage migration and memory migration are two separate tasks. At the end of the storage dirty iteration, the VM is suspended and the remaining dirty blocks are copied to destination. Subsequently, storage I/O requests go to the destination, and thus no more dirty blocks will be generated, while the memory and the CPU of the VM are still at the source, so the storage I/O accesses remains remote until the memory migration completes. The speed for memory migration in VMware would be lower than that in KVM, because the network bandwidth is shared between the migration and remote I/O requests. Therefore, for VMware, Equation 4 will be adjusted as follows:

$$t_{Memory} = \frac{MEM\_SIZE}{speed_{Memory} - AVE\_MEM\_DIRTY\_RATE} \quad (5)$$

The above migration time model describes how the time is spent in each phase of live migration. The next question to address is the amount of data to migrate.

2.2.2 Dirty set and dirty rate prediction

Migrated data consists of two parts. The first part is the original disk and memory (*DISK_SIZE* in Equation 2 and *MEM_SIZE* in Equation 4), the size of which is known. The second part is the generated dirty blocks and dirty pages during migration (*DIRTY_SET_SIZE* in Equation 3), the size of which is unknown. We now present algorithms for predicting that unknown variable.

Disk dirty set prediction: We divide the VM disk into a sequence of small blocks (e.g. 1MB per block). For each block, we record the average write interval, the variance of write interval (used in dirty rate prediction), and the last written time. When a write operation is issued, Pacer updates the record for blocks accessed by the operation.

Dirty set consists of three types of blocks. $SET1$ is the migrated blocks, which are already dirty. This set can be computed by taking the intersection of already-migrated blocks and dirty blocks. $SET2$ is the migrated blocks which are clean right now, but they are predicted to be dirty before the end of pre-copy. The pre-copy finish time is predicted by the current progress, and the block write access pattern is predicted by the current access sequence. If a block will be written before pre-copy finishes, it should be included in this set. $SET3$ is the not-yet-migrated blocks, which are predicted to get dirty after its migration finish time and before the end of pre-copy. The fact that whether a block should be in this set is predicted by the possible migrated time of that block and its possible written time.

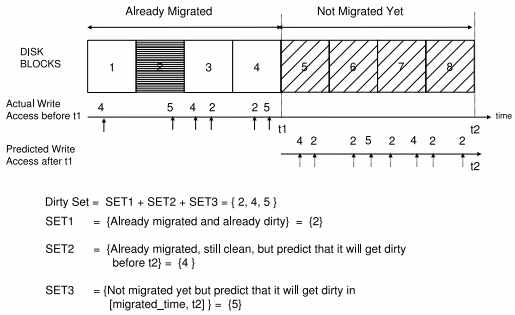


Figure 2: An example of disk dirty set prediction.

FUNCTION getPredictedDirtyBlockSet(*remain_precopy_size*, *speed_expected*)

$SET_{Dirty} = \{\}$
$SET1 = \{block_i |$ already migrated and marked as dirty $\}$
$T_{end} = current\_time + \frac{remain\_precopy\_size}{speed\_expected}$
$SET2 = \{block_i |$ already migrated and marked as clean $\}$
$\cap \{block_i | \exists k : t_{last\_written}(block_i) + k \cdot ave\_write\_interval(block_i) \in [current\_time, T_{end}]\}$ // k is an integer
$SET3 = \{block_i | not\ migrated\ yet\}$
Predict the expected migration time t_i for each $block_i \in SET3$
$SET3 = SET3 \cap \{block_i | \exists k : t_{last\_written}(block_i) + k \cdot ave\_write\_interval(block\_i) \in [t_i, T_{end}]\}$
$SET_{Dirty} = SET1 \cup SET2 \cup SET3$
return SET_{Dirty}

An example is shown in Figure 2. The first 4 blocks are already migrated to destination. $t1$ is the current time when the dirty set prediction algorithm is invoked, and $t2$ is the predicted pre-copy finish time. Among the migrated blocks, block 2 is known to be dirty and is in $SET1$. Block 4 is migrated and is clean so far, but we predict that it will get dirty before $t2$, so block 4 is in $SET2$. Among the non-migrated blocks, block 5 was accessed before, and we predict that it will be written after its migration time and before $t2$. Block 5 is in $SET3$. Thus the dirty set is $\{2,4,5\}$.

Disk dirty rate prediction: We develop an analytical model of the dirty iteration to predict disk dirty rate. Let t be the time budgeted for the dirty iteration. Consider the state of the disk *at the beginning of the dirty iteration*. Let N be the number of dirty blocks in SET_{Dirty} and M be the number of clean blocks in SET_{Clean}, and let $dblock_i$ be the i-th block in the dirty set and $cblock_i$ be the i-th block in the clean set. Abstractly, during each time interval $t' = \frac{t}{N}$, Pacer needs to perform the work to migrate one of the N dirty blocks and any newly generated dirty blocks in the same interval. In the first interval t', $dblock_1$ is migrated. The expected number of new generated

dirty blocks that are assumed to be cleaned immediately during this first interval is $D_1 = \sum \frac{t'}{ave\_write\_interval(block_i)}$ $\forall block_i \in SET_{Clean} \cup \{dblock_1\}$. Note that $dblock_1$ is included because it becomes clean. In general, the expected number of new generated dirty blocks during the k-th interval is $D_k = \sum \frac{t'}{ave\_write\_interval(block_i)}$ $\forall block_i \in SET_{Clean} \cup \{dblock_1, dblock_2, ..., dblock_k\}$. Thus, the average dirty rate can be computed as follows:

$$\begin{aligned} AVE\_DIRTY\_RATE &= \frac{\sum_{i=1}^{N} D_i}{t} \cdot blocksize \\ &= \sum_{i=1}^{M} \frac{blocksize}{ave\_write\_interval(cblock_i)} \\ &+ \sum_{k=1}^{N} \frac{(N+1-k) \cdot blocksize}{N \cdot ave\_write\_interval(dblock_k)} \end{aligned}$$

Furthermore, to remove inactive blocks from dirty rate calculation, we add the following mechanism. For simplicity, assume the write intervals of a block follow a normal distribution [14] $\sim N(\mu, \sigma)$. The possibility that the next arrival time is in $[\mu - 2\sigma, \mu + 2\sigma]$ is 95%. Therefore, if the time since the last write is already longer than 2σ for a block, that block can be safely considered inactive. The average write interval for such a block is set to infinity. This mechanism significantly improves the accuracy of dirty rate prediction.

Memory dirty rate prediction: The disk dirty rate prediction algorithm would incur high tracking overhead if it is applied to the memory dirty rate prediction. Therefore, we propose a sampling-based algorithm to trade precision for reduced overhead. The idea is that Pacer periodically takes a snapshot of the dirty bitmap of memory pages, resets the dirty bitmap, and updates two types of information. The first is a cumulative write access counter for each page. If a page is written to during this period, this counter is incremented. The second is the number of unique written pages u during this period obtained by counting the number of set bits. With these information, we can predict the average dirty rate as follows. We define the access ratio for each page i as $access\_ratio(i) = \frac{write\_access\_counter(i)}{\sum write\_access\_counter(i), i \in \{all\ pages\}}$

Denote the sampling interval to be t_s, and then the rate that unique write pages are generated is $\frac{u}{t_s}$. The rate is an upper bound for the true dirty page rate, and it corresponds to the worst case scenario, where all pages were clean at the beginning of the interval. With access ratio representing the contribution of a page to the overall dirty rate, the dirty rate for page i can be predicted as $d(i) = \frac{u}{t_s} \cdot access\_ratio(i)$. Similar to the analysis for the disk dirty iteration, when migrating the n-th page, the dirty rate is $\sum_{i=1}^{n} d(i)$. The average dirty rate is therefore $\frac{\sum_{k=1}^{N} \sum_{i=1}^{k} d(i)}{N}$ where N is the total number of memory pages.

The selected sampling interval would affect the accuracy of the prediction. For example, if we sample at 2s and there is a page written every one second, its predicted dirty rate will be lower than the real dirty rate. A way to increase the accuracy is to reduce the sampling interval in consecutive rounds and see whether the predicted dirty rate increases. If the dirty rate increases, the sampling interval will be reduced further until the rate stabilizes or the interval meets a configured minimal interval. In Pacer, the sampling interval starts at 2s and is reduced by half if needed. To bound the overhead, we set a minimum sampling interval to 0.25s.

2.2.3 Speed measurement

In KVM, a static *configured migration speed* is used to decide how fast the migration system will copy and transmit migrated data. However, the *actual migration speed* may be smaller than the configured speed due to interference. It is therefore important to measure and use the actual speed for migration time prediction in Equations (2) (3) (4)

Smoothing measurements: In each interval, we measure the migrated data and compute the average actual speed during the interval. In order to smooth out short time scale variations on the measured actual speed, we apply the commonly used exponential smoothing average method to update the measured actual speed. The smoothing weight α is set to 0.8 in the implementation.

$$speed_{smooth} = \alpha \cdot speed_{smooth} + (1-\alpha) \cdot speed_{measured} \quad (6)$$

2.3 Controlling migration time

Pacer divides the migration time into rounds of small intervals. In each round, Pacer adapts the configured migration speed to maintain a target migration finish time. It updates the prediction of dirty block set, dirty disk rate, and dirty memory rate based on the algorithms in Section 2.2.2, and then Pacer computes proper configured migration speed in a way as the following section describes. The speed is adjusted later based on the algorithms that handle I/O interference in Section 2.3.2.

2.3.1 Solving for speeds in each phase of migration

For a specific desired migration time T, many combinations of migration speeds in each phase are feasible. Pacer aims to control the migration progress in a systematic and stable way, which leads to the following speed solutions.

Migrating memory pages generally will not generate disk I/O, because for performance consideration the memory of the VM is usually mapped to the memory of the physical machine, and thus the speed of memory migration is limited by the available network bandwidth ($NETWORK\_SPEED$ which can be directly measured) and so

$$speed_{Memory} = NETWORK\_SPEED \quad (7)$$

With above simplification, only two variables need to be solved: $speed_{Precopy}$ and $speed_{DirtyIteration}$. There are still many combinations of such speeds that can finish migration in time T. Migration generates I/O reads which interfere with the I/O workload of the VM during the storage migration. To minimize the severity of disk I/O interference caused by migration, we seek to minimize the maximum migration speed used. This policy implies that

$$speed_{Precopy} = speed_{DirtyIteration} \quad (8)$$

where $speed_{DirtyIteration}$ is the average speed for the dirty iteration during storage migration. Thus, the appropriate $speed_{Precopy}$ can finally be solved by substituting and rearranging terms in Eq. (1).

More precisely, during the pre-copy stage, at the beginning of each interval, we solve the following equations to obtain the migration speed ($speed_{Precopy}$ or s_1 for short) to use for the interval.

Solve following equations. We use t_1, t_2, t_3
to represent $t_{Precopy}, t_{DirtyIteration}, t_{Memory}$
and s_1, s_2 to represent $speed_{Precopy}$,$speed_{DirtyIteration}$
remain_time is the remaining migration time before deadline
remain_precopy_size is the remaining disk data in the precopy

$$\begin{cases} t_1 + t_2 + t_3 = remain\_time - T_{downtime} \\ t3 = \frac{remain\_msize}{NETWORK\_SPEED - dirtyrate\_mem - dirtyrate\_disk} \\ s_1 = \frac{remain\_precopy\_size}{t_1} \\ dirty\_set\_size + dirtyrate\_disk \cdot t_2 = s_2 \cdot t_2 \\ s_1 = s_2 \\ s_1, s_2 \geq 0 \\ 0 \leq t_1, t_2 \leq remain\_time - T_{Downtime} - t_3 \end{cases}$$

During the dirty iteration, we have the total bytes of current dirty blocks $dirty\_dsize$. The migration speed consists of two parts. One part is to migrate the current dirty blocks in the remaining time before memory migration. The other part is to migrate newly generated dirty blocks at the rate of $dirtyrate\_disks$.

$$speed_{DirtyIteration} = \frac{dirty\_dsize}{remain\_time - t_{Memory}} + dirtyrate\_disk \quad (9)$$

During memory migration, the migration speed is set to the available network bandwidth.

We apply an algorithm which will be described in Section 2.3.2 for computing the maximal actual migration speed that can be realized under interference. When Pacer detects that the computed speed is higher than the maximal actual speed, it knows finishing by the desired time is not feasible. Then it employs disk I/O throttling to upper bound the disk dirty rate to a configurable fraction of the achievable migration speed.

Figure 3 illustrates how the migration speed might be controlled by Pacer during different migration stages.

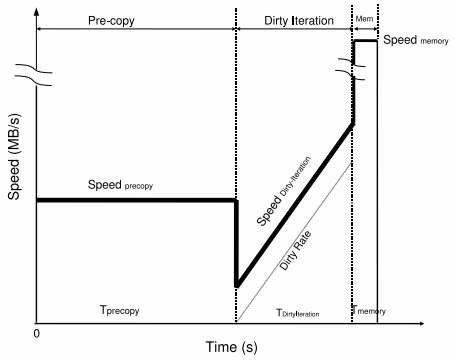


Figure 3: An example of migration speeds in different stages.

During pre-copy, Pacer aims to maintain a stable speed but adapts to workload changes if necessary. During dirty iteration, the migration speed depends on the dirty set size and the dirty rate. At the beginning of dirty iteration, the dirty set already includes the most frequently written blocks, so few new blocks will get dirty, corresponding to a low dirty rate. As more dirty blocks become clean, the dirty rate increases. The shape of the curve in practice depends on the workload. Pacer aims to migrate the dirty set at a stable pace, and thus the dirty iteration migration speed curve is parallel to the dirty rate curve. Finally, during memory migration, migration can typically proceed at a higher speed than the speed in the previous two stages because the bottleneck is most likely in the network.

Other variants: Similar to the discussion in Section 2.2.1 for migration time model, speed control can readily be adapted to other hypervisor. As an example, for VMware, Equations 7 will be adjusted as follows:

$$speed_{Memory} = NETWORK\_SPEED - IO\_RATE \quad (10)$$

where IO_RATE denotes the bandwidth consumed by remote storage I/O and can be predicted by monitoring the application workload.

2.3.2 Maximal speed prediction and tuning

Due to interference from disk or network, the achieved actual migration speed may vary. Therefore, it is important to predict the true maximal actual migration speed and ensure that the configured migration speed is realized.

We predict the maximal actual speed by comparing the configured speeds as specified by Pacer and the measured actual speeds in reality. When the migration starts, if we detect that the measured actual speed cannot reach the configured speed, we will record this speed values pair. In subsequent rounds, if the new measured actual speed is lower than or equal to the previous recorded actual speed, and the new configured speed is higher than

previous recorded configured speed, we predict that the maximal actual speed has been reached. The maximal actual speed is updated by the current actual speed. In the future, when any measured actual speed is higher than the maximal actual speed, the maximal actual speed is updated. In order to smooth out short time scale variations on the maximal actual speed, we use an exponential smoothing average for updating the maximal actual speed. The smoothing weight β in Pacer is set to 0.8.

When the measured actual speed cannot reach the configured speed in a round, Pacer will scale up the configured speed for the next round and set a scale-up flag to indicate that the speed has been scaled up. In the next round, if the new measured configured speed is not higher than the previous measured actual speed, that means the scaling up did not help. Pacer then does not perform scale up for the next round.

3 Evaluation

3.1 Implementation

Pacer is implemented on the kernel-based virtual machine (KVM) platform. KVM consists of a loadable kernel module, a processor specific module and a user-space program – a modified QEMU emulator. QEMU performs management tasks for the VM. Pacer is implemented on QEMU version 0.12.50. We add two options in the migration command to enable prediction and to specify desired migration time with about 1500 lines of code.

3.2 Experiment Setup

The experiments are set up on two physical machines. Each machine has a 3GHz Quad-core AMD Phenome II X4 945 processor, 8GB DRAM, 640GB WD Caviar Black SATA hard drive, and Ubuntu 9.10 with Linux kernel (with the KVM module) version 2.6.31. In all experiments unless otherwise specified, the migration speed is restricted to no more than 32MBps to mimic the level of available bandwidth in inter-datacenters scenarios.

In our test platform, the I/O write speed on the destination disk for migration is at most 15MBps, while RAID is widely used in commercial clouds to increase the I/O speed to be over a hundred MBps. To fully measure the migration prediction with a wide range of configured speeds and to meet the time control requirement of various desired migration times, we modify QEMU at the destination machine not to write the received data to the disk. To ensure that the result is not biased by the disabled writing, we run a set of experiments with enabling and disabling writing at the destination. We vary the client number and compare the average prediction error in both cases. The difference is less than 1*s*. We vary the desired migration time and compare the difference between the actual migration time and desired time in both cases. The difference is less than 1*s* again. The results show that disabling writing does not bias the experiment results.

Workload Name	VM Configuration	Server Application	Default # Clients
File Server (fs)	SLES 10 32-bit 1 CPU,256MB RAM,8GB disk	dbench	45
Mail Server (ms)	Windows 2003 32-bit 2 CPU,1GB RAM,24GB disk	Exchange 2003	1000
Java Server (js)	Windows 2003 64-bit 2 CPU,1GB RAM,8GB disk	SPECjbb @2005-based	8
Web Server (ws)	SLES 10 64-bit 2 CPU,512MB RAM,8GB disk	SPECweb @2005-based	100
Database Server (ds)	SLES 10 64-bit 2 CPU,2GB RAM,10GB disk	MySQL	16

Table 2: VMmark workload summary.

The experiment VMs run VMmark Virtualization Benchmark [19]. VMmark consists of five types of workloads: file server, mail server, database server, web server, java server, each representing different types of applications. Table 2 shows the configurations of those servers. We vary the number of client threads to generate different levels of workload intensity. A simple program is used to generate competing disk I/O traffic on the source machine to create more challenging test scenarios that are more representative of multi-tenancy clouds. It randomly accesses the disk by generating read/write I/O requests. Three models are applied to control the I/O rate by varying the interval between two I/O requests. The static model generate I/O with a constant interval. Two dynamic models generate I/O following the exponential distribution or the Pareto distribution. Each experiment is run for 3 times with different random number seeds.

3.3 Prediction of migration time

3.3.1 Naive predictors do not work

In the following experiment, we will show that predicting migration time based on the naive static prediction or dynamic prediction does not work.

Static prediction is based on the formula $\frac{storage\_size+memory\_size}{configured\_migration\_speed}$. It is commonly used when users want to predict their migration time.

We also implement a dynamic predictor, called progress meter based on the migration progress. Whenever migration progress increases by 1%, the predictor records the current migration time t and progress x%, computes the progress rate $\frac{x\%}{t}$, and uses that rate to predict the finish time $\frac{100\%*t}{x\%}$ dynamically.

The experiment runs on two types of image sizes to represent the typical image sizes in the industrial environment. 160GB is the size of the Amazon EC2 small instance and 8GB is the image size of the VMmark file server workload. We use a micro benchmark to repeatedly write to a data region of the VM's virtual disk at the specified write rate. The size of the written region and the write rate are varied to create different dirty set sizes

(a) VM-160GB

Predictor	Prediction Error					
	Vary Write Rate (Written Region Size 10GB)			Vary Written Region Size (Write Rate 20MBps)		
	5MBps	15MBps	25MBps	5GB	15GB	25GB
Static	326s	395s	519s	185s	698s	1157s
Prog. Meter	316s	382s	510s	169s	687s	1149s
Pacer	6s	5s	8s	8s	10s	9s

(b) VM-8GB

Predictor	Prediction Error					
	Vary Write Rate (Written Region size 1GB)			Vary Written Region Size (Write Rate 20MBps)		
	5MBps	15MBps	25MBps	512MB	1GB	2GB
Static	43s	74s	99s	46s	60s	122s
Prog. Meter	41s	70s	94s	45s	51s	114s
Pacer	4s	6s	5s	5s	6s	4s

Table 3: Prediction errors for naive solutions are several orders of magnitude higher than Pacer.

and dirty rates during migration.

Table 3 shows the results. The prediction errors for static prediction and progress meter are several orders of magnitude higher than Pacer, because they ignore the dirty iteration and memory migration periods. They scales up with higher write rates and larger written region sizes. Pacer always achieves small prediction errors.

3.3.2 Pacer in face of uncertain dynamics

We vary multiple dimensions in the migration environment to illustrate that Pacer has good performance when it predicts migration time in those different scenarios. We use fileserver with 8GB storage as the representative workload in many experiments because it is the most I/O intensive workload in VMmark and it challenges Pacer the most. In the first three experiments there is no competing traffic. Pacer computes a predicted time every five seconds and reports it. To avoid prediction spikes due to the sudden, temporary workload shifts, a cumulative average predicted time over all individual predicted times is generated and reported to the user. After migration, we evaluate the accuracy of the prediction by computing the absolute difference of the actual migration time and each cumulative average time, and then report the average of those values of absolute difference in Table 4.

Vary configured speed: This experiment is based on the file server with the workload of 15 clients. We vary the configured migration speed from 30MBps to 50MBps. As Table 4 shows, the average prediction error varies from 2s to 7s.

Vary the number of clients: This experiment is based on the file server with the default configured speed of 32MBps. We vary the number of clients from 0 to 30 to represent light workload, medium workload, and heavy workload. The average prediction error ranges from 2s to 6s. The result shows that Pacer achieves good prediction even with heavy workload.

Vary workload type: We vary the workload types with the default configured speed of 32MBps. The average prediction error varies from 1s to 8s across four types of workload.

	Actual Migration Time	Average Prediction Error
Vary configured speed (fs-15 clients)		
30 MBps	309s	5s
40 MBps	234s	2s
50 MBps	201s	7s
Vary the number of client (Configured speed 32MBps)		
0 client	263s	2s
15 client	288s	2s
30 client	331s	6s
Vary workload types		
ms-200 client	794s	8s
js-16 client	264s	1s
ws-100 client	269s	2s
ds-16 client	402s	8s
Vary additional competing traffic (fs-15 clients)		
Pareto 50ms	319s	4s
Pareto 90ms	299s	6s

Table 4: Prediction with Pacer

Vary additional competing traffic: This experiment is based on file server with 15 clients. We vary the intensity of additional competing traffic based on the Pareto model of average 50ms and 90ms sleeping time. The average prediction errors are 4s and 6s.

We observe one more advantage of Pacer, which is that it achieves accurate prediction from the very beginning of the migration. We take the predictions in the first minute and compute the average prediction error for each experiment above. The result errors are within the range of $[2s, 12s]$, which is slightly larger than the one for the entire duration of migration. The reason why Pacer achieves accurate prediction from the very beginning is that the dirty set and dirty rate prediction algorithms are very effective. We will quantify the benefits of these algorithms in Section 3.4.3 in detail.

In summary, Pacer provides accurate average prediction in various scenarios. The prediction error ranges from 1s to 8s across all the above scenarios.

3.4 Best-effort migration time control

3.4.1 Naive time controller does not work

We implement a naive adaptive time controller. It is similar to Pacer except that it assumes no future dirty blocks. The migration speed is computed from $\frac{remain\_pre\_copy + existing\_dirty\_blocks}{remain\_time}$. Similar to the setup in Section 3.3.1, the experiment is on two types of image size, 160GB and 8GB. The micro benchmark is leveraged to generate dynamic write workload on VM. The desired migration time is 6500s for the migration of VM (160GB) and is 400s for the migration of VM (8GB).

Table 5 shows the migration time deviation. The actual migration time with Pacer is very close to the desired time, with maximal deviation of [-1s,+6s]. The migration time with naive solution exceeds the desired time up to 1528s and the deviation gets larger when the workload

(a) VM-160GB

Migration Time Controller	Migration Time Deviation					
	Vary Write Rate (Written Region Size 10GB)			Vary Written Region Size (Write Rate 20MBps)		
	5MBps	15MBps	25MBps	5GB	15GB	25GB
Naive Controller	282s	309s	327s	264s	1004s	1528s
Pacer	2s	4s	4s	5s	6s	4s

(b) VM-8GB

Migration Time Controller	Migration Time Deviation					
	Vary Write Rate (Written Region Size 1GB)			Vary Written Region Size (Write Rate 20MBps)		
	5MBps	15MBps	25MBps	1GB	2GB	3GB
Naive Controller	31s	47s	59s	54s	88s	110s
Pacer	1s	2s	-1s	1s	1s	2s

Table 5: Migration time deviation for Pacer is much smaller than naive controller.

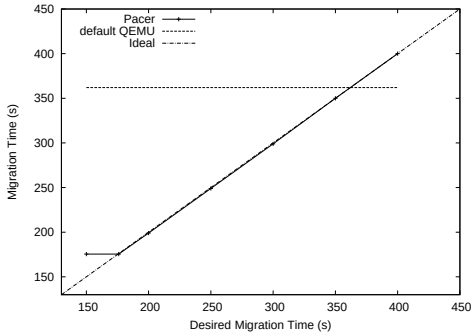


Figure 4: Migration with different desired finish times. Pacer almost matches the ideal case when the desired time is larger than 176s. The deviation is very small in [-2s,2s].

is more write intensive, because it lacks the capability to predict the amount of remaining blocks for migration and selects a wrong speed. We will show how key components in Pacer help to reduce the deviation later in Section 3.4.3.

3.4.2 Pacer in face of uncertain dynamics

Similar to the experiments for prediction, we vary multiple dimensions in the migration environment to show that Pacer can perform adaptive pacing to realize the desired migration time.

Vary desired migration time: This experiment is based on the file server with the workload of 30 clients. We vary the desired migration time from 150s to 400s. The Figure 4 shows that when the desired time is within the range of $[200s, 400s]$, the migration time in the three runs is very close to the desired time, with maximal deviation of $[-2s, 2s]$. When we lower the desired migration time way beyond anything feasible, the I/O bottleneck limits the migration speed, and consequently Pacer will hit its minimal migration time of 176s, while the default QEMU with the configured speed of 32MBps can finish the migration in 362s.

Vary the number of clients: We vary the number of clients from 0 to 60 on the file server. As Figure 5

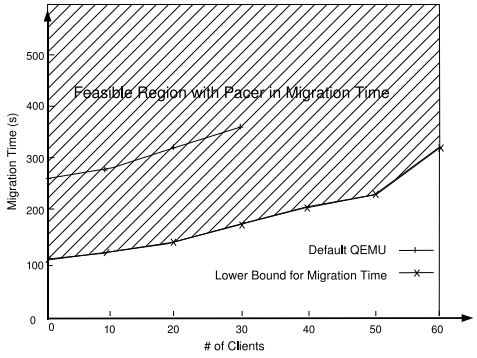


Figure 5: Migration with different degrees of workload intensity. Any point in the feasible region can be realized by Pacer. The lower bound for migration time is limited by I/O bottleneck. Default QEMU can only follow a narrow curve in the region.

Desired Time	10 Clients	20 Clients	30 Clients	40 Clients	50 Clients	60 Clients
144s	$[-1,0]$	$[0,0]$	–	–	–	–
176s	$[0,0]$	$[-1,1]$	$[0,1]$	–	–	–
203s	$[-1,1]$	$[-2,1]$	$[0,0]$	$[0,1]$	–	–
222s	$[0,0]$	$[0,1]$	$[-1,0]$	$[-1,0]$	$[0,1]$	–
305s	$[0,0]$	$[-2,1]$	$[-1,0]$	$[-2,0]$	$[0,0]$	$[0,0]$
400s	$[0,0]$	$[-1,0]$	$[-2,0]$	$[-2,0]$	$[-1,1]$	$[-2,0]$

Table 6: Deviation of migration time on Pacer with different workload intensities. The number in the bracket represents the worst earliest and latest deviation in Pacer. For example, $[-1,1]$ means at most early by 1s and late by 1s. "-" means the time is beyond the feasible region.

shows, there exists a lower bound for migration time that is the minimal migration time due to the I/O bottleneck. Pacer can achieve adaptive pacing in any time in the feasible region above the smallest possible time for migration to complete, while default QEMU can only achieve a non-configurable migration time for a specific number of clients. Moreover, when the number of clients increases to 35 or more, QEMU cannot converge and the migration time becomes infinite. This is because QEMU uses a configured speed which is a constant and will not increase even if there is enough I/O bandwidth.

We choose six different desired migration times from 144s to 400s in the feasible region, and migrate VM with different number of clients with those different desired migration times. The results in Table 6 show that Pacer can achieve the desired time in all cases with maximal deviation of $[-2s, 1s]$.

Vary workload type: We perform live migration with Pacer for five types of VMmark workloads. In order to guarantee that the default QEMU can converge in the migration, we decrease the number of clients. We run default QEMU first and get the migration time, and then we set this time to be Pacer's desired migration time. Table 7 shows that Pacer can achieve desired migration time with a small deviation in $[-2s, +2s]$.

Vary additional competing traffic: To test whether

Workload	Desired Migr Time(s)	Pacer Migr Time(s)
fs-30 clients	362	360
ms-200 clients	897	899
js-16 clients	274	275
ws-100 clients	287	287
ds-16 clients	471	473

Table 7: Migration time on different types of workload. Pacer can achieve the desired migration time.

Sleeping Time	Run1 MigrTime Dev(s)	Run2 MigrTime Dev(s)	Run3 MigrTime Dev(s)
No Add Traffic	-1	0	0
Static 50ms	0	-5	1
Expo (ave 50ms)	-5	0	-4
Pareto (ave 50ms)	0	-2	3
Static 90ms	-3	0	-5
Expo (ave 90ms)	-5	-2	1
Pareto (ave 90ms)	0	2	1

Table 8: Migration time for Pacer when the additional competing traffic varies. Pacer can achieve the desired migration time with a small finish time deviation.

Pacer can achieve desired migration time when different levels of I/O interference exist, we run the following experiment with the program in Section 3.2 to generate additional competing I/O traffic. The migrated VM runs file server with 30 clients. The desired migration time is 264s. Table 8 shows the results for three runs.Pacer can achieve the desired time when the I/O interference varies. The deviation is $[-5s, 3s]$ which is small comparing to the desired time of 264s.

3.4.3 Benefits of key components in Pacer

Dirty set and dirty rate prediction: In order to understand the benefit of key components in Pacer, we design an experiment to compare the systems with and without dynamic dirty set and dirty rate prediction to evaluate the effectiveness of the algorithms. The workload is file server. As Table 9 shows, the actual migration time will exceed the desired migration time significantly without any prediction algorithm. When only the dynamic dirty set prediction algorithm is added into the system, migration time improves but still exceeds the desired time. When both the dirty set and dirty rate prediction algorithms are used in Pacer, Pacer can perform adaptive pacing with very little deviation $[-2s, -1s]$.

Speed measurement and tuning: We design an experiment to run Pacer with and without maximal speed prediction. The VM runs file server with 30 clients. Additional competing traffic is generated by constant interval 10ms. Without maximal speed prediction, migration runs in 697s when the desired time is 600s. With prediction, migration can finish in time. Moreover, we design another experiment to run migration with and without the speed scale-up algorithm on the file server with 30 clients, but without additional competing traffic on the disk. We set the desired migration time to be 200s, 300s and 400s. The results are shown in Table 10. Without the speed scale-up algorithm, migration will considerably exceed the desired time in all three experiments.

Work load	Desired Time(s)	Pacer w/o dirty set/rate prediction(s)	Pace with only Dirty set prediction(s)	Pacer (s)
30 clients	200	216	206	198
60 clients	400	454	431	399

Table 9: Importance of dynamic dirty set and dirty rate prediction. Without any of these algorithms, it is hard to achieve desired migration time.

Desired Time	With	Without
200s	198s	284s
300s	300s	380s
400s	399s	553s

Table 10: Importance of speed scaling up algorithm.

3.5 Overhead of Pacer

In this experiment, we measure the overhead introduced by Pacer in terms of time and space. Take best effort time control for example. We run migration with Pacer for file server with 60 clients and 400s desired migration time. We measure the computation time of Pacer in each round. In our observation, the computation time is 28.24ms at the beginning of migration. As the migration moves on, more blocks in the dirty set are determined, so the computation time drops to below 1ms at the final stage of migration. Overall in this experiment, Pacer on average only incurs 2.4ms of computation for each 5 second interval. The overhead is 0.05% which is negligible. Comparing to default QEMU, the additional memory space consumed is less than 1MB. Prediction consumes less computation resource than best-effort time control. In summary, the overhead of Pacer is small and has no performance impact on application.

3.6 Potential robustness improvements

Pacer could be improved further by including mechanisms to mitigate negative impact of the rare case that the migration environment variables are not steady. First, Pacer is an adaptive system with a fixed adaptation interval (5s) in current design. A flexible interval can be applied when Pacer detects that the state is not steady. Shortening interval incurs fast adaptation but more overhead. Second, we can test migration environment, e.g. newwork bandwidth, against the expected patterns to find out whether any increasing or decreasing trend exists. These mechanisms will be considered in our future work.

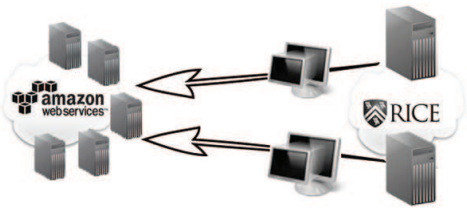


Figure 6: EC2 demonstration setup

Workload intensity	None	Low	Medium	Heavy
Actual Migration Time	227s	240s	255s	250s
Average Prediction Error	6s	5s	4s	7s

Table 11: Prediction accuracy with Pacer in EC2.

4 EC2 Demonstration

To demonstrate the functions of Pacer in a commercial hybrid cloud environment, we conduct a set of experiments using the Amazon EC2 cloud. In these experiments we migrate VMs from the Rice campus network to EC2. On EC2, we use High-CPU Medium instances running Ubuntu 12.04. EC2 instances do not support KVM, thus we use the "no-kvm" mode in QEMU in EC2. The downside is that the performance of VM is not as good as with KVM's hardware virtualization support.

4.1 Network and disk speed measurements

We run experiments to characterize the network and disk speed that can be achieved between Rice and EC2 and make several interesting observations. First, we use "iperf" to measure network throughput for 200s. We find that when transmitting data from Rice to EC2, the throughput increases gradually and linearly for a surprisingly long 30s before it maximizes at roughly 60MBps. More specifically, 50% of the speed samples fall between 58MBps to 62MBps. After the initial 30s, 5% of the speed samples are below 40MBps and 3.5% are below 30MBps. Based on these findings, we cap the migration speed in the experiments to 50MBps. Second, we use "scp" to copy a 8GB file from Rice to EC2 to measure achievable disk speed. We sample the reported speed every 0.5s. The average speed is 30.9MBps and the variation is 5MBps. Thus, disk speed is the most likely bottleneck for migration in the EC2 experiments.

4.2 Prediction of migration time

To measure the accuracy of Pacer's prediction, we migrate one VM that runs the file server from Rice to EC2. We vary the number of clients to emulate different workload intensities of the VM server. The CPU utilization rate is 30-45% for the low workload, 45-55% for the medium workload, and 55-70% for the high workload.

For each intensity of the workload we run three sets of experiments and report the average prediction error in Table 11. The first observation is that the accuracy of the prediction does not decrease as the workload increases.

Desired time	500s	600s	700s	800s
Deviation	[-2s,+2s]	[-2s,+2s]	[-1s,+2s]	[-3s,0s]

Table 12: Migration time control with Pacer in EC2.

Secondly, given the fact that the network and disk speeds are quite unstable, Pacer still can predict with an average absolute error of about 5s. We find that if disk writes at the destination are disabled to eliminate the impact of disk speed variation, the average prediction error is reduced to 2s. Given the disk speed typically fluctuates 16% from the average speed, the obtained average prediction error ranging from 2% to 3% of the actual migration time is quite desirable.

4.3 Best-effort migration time control

In this experiment we migrate the 8GB file server with medium workload and vary the desired migration time from 500s to 800s. For each desired time we run three experiments and report the range of the deviations in Table 12. Although we have reported that the network and disk speeds between Rice and EC2 are not very stable, Pacer still works very well in controlling the migration time to within a small deviation $[-3s, +2s]$ of the desired time.

5 Discussion

Live VM migration progress management functions are sorely missing from today's hybrid cloud computing management systems. Our contribution is Pacer – the first complete system capable of accurately predicting migration time and managing the progress of a migration to finish as close to a desired finish time as possible. Through extensive experimentations, including a real-world commercial hybrid cloud scenario with Amazon EC2, we show that Pacer is highly effective in all its functions.

Because of space limitation, we will briefly introduce how Pacer help in the use case of coordinating multiple VM migrations (refer to the second use case in Section 1.1). Enterprises usually have applications consisting of multiple interacting components. To coordinate the concurrent migration of VMs, we design a centralized controller by leveraging Pacer's prediction and time control capabilities. It continuously gathers and analyzes the predicted migration time for each VM, dynamically update the desired migration time for each VM to a feasible target, and paces the migration of the whole set of VMs in such way. We test it on EC2 platform by migrating a SPECweb 2005 webserver cluster which includes a web server and a database server. Pacer could reduce the performance degradation time from 394s to 3s. We are currently extending Pacer to analyze and model the behavior of complex enterprise applications so as to automatically and optimally manage the migrations of such

complex applications.

6 Related work

While to our knowledge no previous work is directly comparable to Pacer, there exists related work on setting the speed or estimating the time of *CPU/memory-only* VM migration. Breitgand *et al.* [4] propose a cost function for computing the network bandwidth allocated to CPU/memory-only migration in order to minimize the theoretical number of application delay bound violations as given by a queuing theory model. Akoush *et al.* [2] simulate the execution of the iterative data copy algorithm of CPU/memory-only migration so as to estimate the required migration time. The simulation makes certain simplifying assumptions such as fixed network bandwidth and fixed or historically known memory page dirty rate. Relative to these previous works, not only does Pacer address a different set of problems in migration progress management for *full* VM migration, Pacer also takes a system-building approach, using real measurements and run-time adaptation, which are found to be crucial to cope with workload and performance interference dynamics, realized in a complete system.

Beyond VM migration, there is also interesting related work in *disk-data-only* migration. In [10], Lu *et al.* presents Aqueduct, a disk-data-only migration system that minimizes the impact on the application performance. However, Aqueduct simply treats the migration as a low-priority task and does not provide a predictable migration time. Dasgupta *et al.* [7] and Zhang *et al.* [31] present different rate controlling schemes that attempt to meet a disk-data-only migration time goal. However, these schemes are only simulated. Furthermore, these schemes ignore the problem caused by dirty disk-data generated by write operations during migration.

7 Conclusions

We introduce Pacer – the first migration progress management system.Using a fully implemented system on KVM, we show that Pacer is highly effective and provides large performance benefits across a wide range of realistic and popular virtual machine workloads.

References

[1] VMWare Storage vMotion. http://www.vmware.com/products/storage-vmotion/overview.html.

[2] AKOUSH, S., SOHAN, R., RICE, A., W.MOORE, A., AND HOPPER, A. Predicting the performance of virtual machine migration. In *IEEE 18th annual international symposium on modeling, analysis and simulation of computer and telecommunication systems* (2010), IEEE.

[3] ASHFORD, W. Hybrid clouds most popular with UK business, survey reveals. http://tinyurl.com/868pxzd, Feb. 2012.

[4] BREITGAND, D., KUTIEL, G., AND RAZ, D. Cost-aware live migration of services in the cloud. In *USENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services* (2011), USENIX.

[5] BRIDGE, N. Future of Cloud Computing Survey. http://tinyurl.com/7f4s3c9, June 2011.

[6] CLARK, C., FRASER, K., HAND, S., HANSEN, J. G., JUL, E., LIMPACH, C., PRATT, I., AND WARFIELD, A. Live migration of virtual machines. In *NSDI'05: Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation* (Berkeley, CA, USA, 2005), USENIX Association, pp. 273–286.

[7] DASGUPTA, K., GHOSAL, S., JAIN, R., SHARMA, U., AND VERMA, A. Qosmig: Adaptive rate-controlled migration of bulk data in storage systems. In *Proc. ICDE* (2005).

[8] HAJJAT, M., SUN, X., SUNG, Y., MALTZ, D., RAO, S., SRIPANIDKULCHAI, K., AND TAWARMALANI, M. Cloudward bound: planning for beneficial migration of enterprise applications to the cloud. In *ACM SIGCOMM Computer Communication Review* (2010).

[9] KVM. Kernel based virtual machine. `http://www.linux-kvm.org/page/Main_Page`.

[10] LU, C., ALVAREZ, C. A., AND WILKES, J. Aqueduct: Online data migration with performance guarantees. In *Proc. of the USENIX Conference on File and Storage Technologies (FAST)* (2002).

[11] MURPHY, A. Enabling Long Distance Live Migration with F5 and VMware vMotion. http://tinyurl.com/7pyntvq, 2011.

[12] NELSON, M., LIM, B.-H., AND HUTCHINS, G. Fast transparent migration for virtual machines. In *USENIX'05: Proceedings of the 2005 Usenix Annual Technical Conference* (Berkeley, CA, USA, 2005), USENIX Association.

[13] PADALA, P. Understanding Live Migration of Virtual Machines. http://tinyurl.com/24bdaza, June 2010.

[14] STAELIN, C., AND GARCIA-MOLINA, H. Clustering active disk data to improve disk performance. Tech. Rep. CS-TR-283-90, Department of Computer Science, Princeton University, Sep 1990.

[15] STEELE, C. Virtual machine migration FAQ: Live migration, P2V and more. http://tinyurl.com/cxavodk, Aug. 2010.

[16] TECHCENTER, D. Hyper-V R2 Live Migration FAQ. http://tinyurl.com/c8rayf5, Nov. 2011.

[17] TOFEL, K. C. Forget public; private clouds: The future is hybrids! http://tinyurl.com/bsmsj9p, June 2011.

[18] VERMA, A., KUMAR, G., KOLLER, R., AND SEN, A. Cosmig: modeling the impact of reconfiguration in a cloud. In *IEEE 19th annual international symposium on modeling, analysis and simulation of computer and telecommunication systems* (2011), IEEE.

[19] VMWARE. VMmark Virtualization Benchmarks. http://www.vmware.com/products/vmmark/, Jan. 2010.

[20] VMWARE FORUM. http://tinyurl.com/7gttah2, 2009.

[21] VMWARE FORUM. http://tinyurl.com/ccwd6jg, 2011.

[22] VMWARE FORUM. http://tinyurl.com/cr6tqnj, 2011.

[23] VMWARE FORUM. http://tinyurl.com/bmlnjqk, 2011.

[24] VMWARE FORUM. http://tinyurl.com/d4qr2br, 2011.

[25] VMWARE FORUM. http://tinyurl.com/7azb3xt, 2012.

[26] WU, Y., AND ZHAO, M. Performance modeling of virtual machine live migration. In *Proceedings of the 2011 IEEE 4th International Conference on Cloud Computing* (2011), IEEE.

[27] XEN FORUM. http://tinyurl.com/d5v8j9p, 2008.

[28] XEN FORUM. http://tinyurl.com/c37he9g, 2008.

[29] XEN FORUM. http://tinyurl.com/d477jza, 2011.

[30] XEN FORUM. http://tinyurl.com/c7tyg94, 2011.

[31] ZHANG, J., SARKAR, P., AND SIVASUBRAMANIAM, A. Achieving completion time guarantees in an opportunistic data migration scheme. In *Proc. SIGMETRICSREVIEW* (2006).