Pacer: A Progress Management System for Live Virtual Machine Migration in Cloud Computing

Jie Zheng, T. S. Eugene Ng, Kunwadee Sripanidkulchai, Zhaolei Liu

***Abstract*—Live migration of virtual machines is a key management function in cloud computing. Unfortunately, no live migration progress management system exists in the state-ofthe- art, leading to (1) guesswork over how long a migration might take and the inability to schedule dependent tasks accordingly; (2) unacceptable application degradation when application components become split over distant cloud datacenters for an arbitrary period during migration; (3) inability to tradeoff application performance and migration time e.g. to finish migration later for less impact on application performance. Pacer is the first migration progress management system that solves these problems. Pacer's techniques are based on robust and lightweight run-time measurements of system and workload characteristics, efficient and accurate analytic models for progress predictions, and online adaptation to maintain user-defined migration objectives for coordinated and timely migrations. Our experiments on a local testbed and on Amazon EC2 show that Pacer is highly effective under a range of application workloads and network conditions.**

***Index Terms*—Live migration, cloud computing, progress management, datacenter.**

I. Introduction

THIS paper presents techniques to solve an emerging service management problem in cloud computing, the progress management of *live* virtual machine (VM) migration. Our techniques are implemented in a system called Pacer, which accurately predicts the migration time and uses the prediction to realize two advanced functions of migration progress management: (1) best-effort migration time control, and (2) coordinated migration of VMs. By making the VM migration more predictable, Pacer facilitates orderly and timely service management operations.

A. The Role of Virtualization in Cloud Computing

Cloud computing, or "the cloud", is the delivery of on-demand computing resources over the Internet on a pay-for-use basis [1]. In both industry and academia, cloud computing has attracted significant attention. A research report sponsored by Virtustream, an enterprise cloud provider [2], shows that the cloud is being widely used by more than half of the businesses in the US [3]. In the past, cloud computing services have experienced rapid growth, and in the future the public cloud service market is expected to grow to $206.6 billion by 2016 [4]. Internet and business applications are increasingly shifting to the cloud to maximize effectiveness of shared resources and economies of scale. Cloud providers such as Amazon EC2 [5] and IBM [6] operate public cloud to offer storage and virtual hosts to customers at a low rate on demand, while some other clouds are built to deliver development environments as a service, such as Google App Engine [7].

The cloud service usually runs in data centers. The main purpose of a data center is to host the applications that handle the core business logic and process the operational data of the organization. Currently, data centers may contain hundreds of thousands of servers. Applications running on the cloud are typically composed of multiple components running on multiple hosts. The common components could be databases, file servers, application servers, middle-ware, and various others [8]. A multi-tier e-commerce application [9] may include web servers, database servers and application servers. The front end web server displays information such as merchandise lists. The application server communicates to the front end web server and implements the core functionality of the application. The database server stores the transaction data and keeps it neutral and independent from application logic [9].

A key enabling technology for cloud computing is virtualization, which abstracts the physical infrastructure, hides the complexity of the underlying hardware or software, and makes the infrastructure available as a soft component that is easy to isolate and share physical resources [10], [11]. The management of a pool of virtualized resources requires the ability to flexibly map and move the application and the data across and within pools [12]. Usually there are multiple VMs running on a single physical machine. Therefore, virtualization provides an effective way to consolidate hardware to get vastly higher productivity from fewer servers. Virtualization also speeds and facilitates IT management, maintenance and the deployment of new applications [13]. In cloud computing, the hypervisor or the virtual machine monitor (VMM) is a piece of software that creates, runs, and manages VMs. KVM [14], XEN [15], VMware ESX [16] and Hyper-V [17] are four popular hypervisors.

B. Virtual Machine Migration as a Management Primitive

As data centers continue to deploy virtualized services, there are many scenarios that require moving VMs from one

Manuscript received July 21, 2013; revised October 17, 2013; handled by associate editor K. Van der Merwe.

This research was sponsored by the NSF under CNS-1305379, CNS-1018807 and CNS-1162270, by an Alfred P. Sloan Research Fellowship, an IBM Scholarship, an IBM Faculty Award, and by Microsoft Corp.

J. Zheng, T. S. E. Ng, and Z. Liu are with Rice University (e-mail: Jie.Zheng@rice.edu).

K. Sripanidkulchai is with NECTEC, Thailand.

Digital Object Identifier 10.1109/TNSM.2013.111013.130522

physical machine to another within the same data center or across different data centers. Some examples are:

- **Planned maintenance**: To maintain high performance and availability, VMs needs to be migrated from one cloud to another to leverage better resource availability, avoid down time caused by hardware maintenance, and overcome computing power limitation in the source cloud. If a physical machine requires software or hardware maintenance, the administrator could migrate all the VMs running on that machine to other physical machines to release the original machine [18].
- **Load balancing**: VMs may be rearranged across different physical machines in a cluster to relieve workload on congested hosts [18]. The workload increase of a virtual server can be handled by increasing the resources allocated to the virtual server under the condition that some idle resources are available on the physical server, or by simply moving the virtual server to a less loaded physical server [19].
- **Avoiding single-provider lock-in**: While many cloud users' early successes have been realized by using a single cloud provider [20], [21], the ability to use multiple clouds to deliver services and the flexibility of moving freely among different cloud providers are emerging requirements [22]. Users who used to implement their applications using one cloud provider ought to have the capability and flexibility to migrate applications back in-house or to other cloud providers in order to have control over business continuity and avoid fate-sharing with specific providers [23].
- **Enterprise IT Consolidation:** Many enterprises working with multiple data centers have attempted to deal with data center "sprawl" and cut costs by consolidating multiple smaller sites into a few larger data centers. The ability to move a service with minimal or no down-time is attractive due to the corresponding reduction in the disruption seen by a business [12].
- **Hybrid cloud computing:** 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 [24], [25], [26]. This is not surprising since HCC combines the benefits of public and private cloud computing, resulting in extra flexibility for cost saving, application adaptivity, disaster survivability, etc. In order to achieve the above benefits, VM migration technologies are critical to the HCC paradigm.

In summary, from the cloud providers' perspective, VM migration has maintenance, resource management, disaster planning and economic benefits. From the cloud users' perspective, VM migration could provide lower cost, better reliability and better performance.

C. The Benefits of Live Virtual Machine Migration

There are different approaches to migrate VMs, depending on whether the VM runs a stateless server or a stateful server. Some applications contain stateless servers that do not retain any session information, such as a web server with static content [27]. The service provider could easily migrate a stateless server by terminating the old server and provisioning a new server at the new site. However, the majority of enterprise applications run on stateful servers. The stateful server keeps records of the state of the clients from one request to the next. For instance, ftp server, database server, mail server and web server with dynamic content are all stateful. To migrate a stateful server, the traditional approach is to stop the VM, copy the VM state and restart the VM at the destination. This approach incurs a long application downtime. A more attractive mechanism for moving applications is *live migration*, because the live migration is completely application independent (stateless or stateful) and it avoids interrupting the running application.

Live migration refers to the process of moving a running VM or application between different physical machines without disconnecting the client or application. Full migration of a VM includes the following aspects:

- the running state of the VM (i.e., CPU state, memory state)
- the storage or virtual disks used by the VM
- the networking and existing client connections

State-of-the-art hypervisors, such as VMware ESX, Hyper-V, KVM, and Xen, support live migration. The hypervisors on the source and destination control the migration. The source hypervisor typically copies all the disk state of the VM from the source to the destination while the VM is still running on the source. If some disk blocks are written during this process, they will be re-copied. When the number of disk blocks to be copied becomes less than a threshold, the hypervisor starts to copy the memory state. Once the disk and memory state have been transferred, the hypervisor briefly pauses the VM for the final transition of disk, memory, processor and network states to the destination host.

D. The Problem: Lack of Progress Management During Live Migration

Unfortunately, the state-of-the-art hypervisors enable live migration without providing any ability to manage the migration progress. This deficiency leads to the following questions:

- **How long does the migration take?** – This is a popular question among users of live VM migration [28], [29], [30]. It is also an important question. Nicolae et al. [31] point out that the migration time is important, because it indicates the total amount of time, during which the source machine is busy and cannot be reallocated to a different task or shut down. Unfortunately, there is no simple formula for calculating the answer, because the finish time depends on many dynamic run-time variables. Those variables includes the application I/O workload intensity, network throughput, disk throughput, etc.

  There are numerous indications [32], [33], [34], [35], [36], [37], [38], [39] that users routinely try to guess why the migration is slow and whether it could be sped up, and how long they might have to wait.

- **How to control the trade-off between application performance and migration time?** – This is another popular question raised by users [40], [41]. Studies [42], [43], [44], [45] have shown that the migration can degrade application performance, and decreasing the migration speed helps reduce performance degradation [43]. 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.
- **How to avoid application components getting split between distant datacenters during the migration?** – This issue is important to enterprises, because their applications often consist of multiple interacting components that perform different functions (e.g. content generation, custom logic, data management, etc. [46]). 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 leads to decreased performance.

E. The Solution: Pacer

Our contribution is Pacer – the first migration progress management system. Pacer effectively addresses all above open questions by accurately predicting the migration time, and uses this prediction to further realize the best-effort migration time control and coordinated migration of VMs. Our experiments on a local testbed and on Amazon EC2 show that (1) Pacer can predict the time of a long migration, which takes about 1.7 hours, with less than 10 seconds of error, and this prediction accuracy is achieved quickly, merely within 5 minutes after the migration starts; (2) Pacer can control the migration to finish within 2 seconds of the desired finish time under a variety of application workloads; (3) Pacer can dramatically shorten the time period during which the application components are split between remote datacenters from nearly 400 seconds to 3 seconds.

Pacer's effectiveness is due to the following key elements in the design:

- **Using real run-time measurements to drive decisions** – We have identified important measurements that are essential for accurate predictions, such as application I/O workload intensity (both memory and disk accesses) and the bottleneck migration speed (network or disk). Furthermore, in our implementation of Pacer, we continuously collect these measurements at run-time using techniques with low overhead as shown in Section III.
- **Using detailed analytic progress models** – We have developed novel and detailed analytic models 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 as a function of the characteristics of each migration stage (i.e. disk, dirty blocks, CPU/memory, etc.).
- **Adapting to run-time conditions** – Due to run-time dynamics, static migration settings cannot successfully achieve the migration time control objective or the coordinated migration objective. Pacer continuously adapts to ensure that the objectives are met. In the case of migration time control, Pacer adapts the migration speed to maintain a targeted migration finish time in face of application dynamics. In the case of coordinated migration, Pacer adapts the targeted migration finish time for all components, given what is predicted to be feasible from run-time measurements.

F. Road map

The rest of this paper is organized as follows. Section II presents the techniques in Pacer for migration progress management. Section III and IV present experimental results demonstrating the capability and benefits of Pacer. Section V discusses the application of Pacer in the concurrent migration scenario. We discuss related work in Section VI and conclude in Section VII.

II. Design of Pacer

A. Overview

Pacer is designed for the pre-copy migration model, which is widely used in virtualization platforms KVM [14] and XEN [18]. A slightly different variant of this model used by VMware [47], [48] will be discussed later. In the pre-copy model, the virtual disk migration is performed prior to the memory migration. During the virtual disk migration, 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 process of dirty block retransmission 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 in full VM migration, especially for the dirty iteration. The problem is challenging for two reasons. Firstly, the application on the VM is running and constantly creating dirty pages/blocks throughout the migration depending on its workload and how it accesses the memory/storage, so the total number of dirty pages/blocks during migration is a variable that is not known before the migration completes. Secondly, the migration experiences interference from the migrated VM's own workload and other competing workloads that share the network or disk bandwidth with the migration, and thus the migration speed is also a variable that is not known.

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

B. Predicting migration time

Pacer performs predictions 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 the migration is progressing. The paper will address these three issues in the following sections. In the formulas, we use bold font for constants and regular font for variables.

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 $\mathbf{T_{Downtime}}$ for a small downtime needed to copy the remaining dirty blocks and dirty pages once they drop below a configured threshold. $\mathbf{T_{downtime}}$ is considered as a constant because the remaining data to be migrated is fixed (e.g. downtime is 30ms in KVM):

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

For the pre-copy phase, we have:

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

where $\textbf{DISK\_SIZE}$ 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 the 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 II-B2 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 become 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 II-B2 that is effective for predicting the average memory dirty rate ($AVE\_MEM\_DIRTY\_RATE$).

During memory migration, different hypervisors have different behaviors. For KVM, the VM still accesses the storage in the source and disk blocks could get dirty during the memory migration. Therefore, in KVM, memory migration and storage dirty iteration may happen alternatively. Denoting the size of the memory as $\textbf{MEM\_SIZE}$ and memory migration speed as $speed_{Memory}$, we have

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

Other variants: The previous derivation is based on the assumption that the memory migration follows the storage migration (KVM and XEN). The model can easily be adapted to other hypervisors, 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 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:

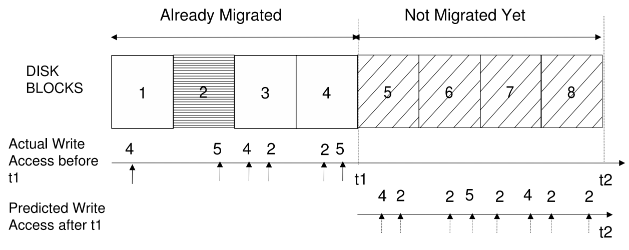
$$t_{Memory} = \frac{\textbf{MEM\_SIZE}}{speed_{Memory} - AVE\_MEM\_DIRTY\_RATE} \quad (5)$$

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) Dirty set and dirty rate prediction: Migrated data consists of two parts. The first part is the original disk and memory ($\textbf{DISK\_SIZE}$ in Equation 2 and $\textbf{MEM\_SIZE}$ in Equation 4), the size of which is known ahead of time. 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 this unknown variable.

Disk dirty set prediction: We divide the VM disk into a sequence of small blocks with $\textbf{BLOCKSIZE}$ (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.



Dirty Set = SET1 + SET2 + SET3 = { 2, 4, 5 }

SET1 = {Already migrated and already dirty} = {2}

SET2 = {Already migrated, still clean, but predict that it will get dirty before t2} = {4 }

SET3 = {Not migrated yet but predict that it will get dirty in [migrated_time, t2] } = {5}

Fig. 1. 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 1. The first 4 blocks are already migrated to the 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 it 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 SET_{Dirty} 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 (D_1)is computed as follows:

$$D_1 = \sum \frac{t'}{ave\_write\_interval(block_i)} \quad \forall block_i \in SET_{Clean} \cup \{dblock_1\} \tag{6}$$

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 computed as follows:

$$D_k = \sum \frac{t'}{ave\_write\_interval(block_i)} \quad \forall block_i \in SET_{Clean} \cup \{dblock_1, dblock_2, ..., dblock_k\} \tag{7}$$

Thus, the average dirty rate can be computed as follows:

$$\begin{aligned} AVE\_DIRTY\_RATE &= \frac{\sum_{i=1}^{N} D_i}{t} \cdot \mathbf{BLOCKSIZE} \\ &= \sum_{i=1}^{M} \frac{\mathbf{BLOCKSIZE}}{ave\_write\_interval(cblock_i)} \\ &+ \sum_{k=1}^{N} \frac{(N+1-k) \cdot \mathbf{BLOCKSIZE}}{N \cdot ave\_write\_interval(dblock_k)} \end{aligned} \tag{8}$$

Our previous research about I/O characteristic in typical virtualization workloads [23] shows that the disk write rate is stable over long time scales. Therefore, the disk dirty rate prediction is able to perform well. To further optimize the algorithm, we add the following mechanism to remove inactive blocks from dirty rate calculation. For simplicity, assume the write intervals of a block follow a normal distribution [49] $\sim N(\mu, \sigma)$. 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 this information, we can predict the average dirty rate as follows.

We define the access ratio for each page i as follows:

$$access\_ratio(i) = \frac{write\_access\_counter(i)}{\sum write\_access\_counter(i), i \in \{all\ pages\}} \tag{9}$$

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.

3) Speed measurement: In KVM, a statically *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 of the measured actual speed, we apply the commonly used exponential smoothing average method to update the measured actual speed. The smoothing weight α represents the degree of weighting decrease, a constant smoothing factor between 0 and 1. A lower α discounts older observations faster and does not smooth-out short term fluctuation well. We ran some experiments to test α in $[0.5, 0.9]$ and found 0.8 to be a reasonable choice.

$$speed_{smooth} = \alpha \cdot speed_{smooth} + (1-\alpha) \cdot speed_{measured} \quad (10)$$

C. Controlling the 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 II-B2, and then Pacer computes a proper configured migration speed described in the following section. The speed is adjusted later based on the algorithms that handle I/O interference in Section II-C2.

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 equations.

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

$$speed_{Memory} = NETWORK\_SPEED \quad (11)$$

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. If we set a high $speed_{Precopy}$ and a low $speed_{DirtyIteration}$, or a low $speed_{DirtyIteration}$ and a high $speed_{Precopy}$, the high migration speed may degrade the application performance significantly. To maintain a stable migration speed, make the pre-copy and dirty iteration balanced, and minimize the severity of disk I/O interference caused by the migration, we apply the following policy:

$$speed_{Precopy} = speed_{DirtyIteration} \quad (12)$$

where $speed_{DirtyIteration}$ is the average speed for the dirty iteration during the 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. NETWORK_SPEED is measured in the previous interval and passed into the equations as a constant.

Solve the 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

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

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 $dirtyrate\_disks$.

$$speed_{DirtyIteration} = \frac{dirty\_dsize}{remain\_time - t_{Memory}} + dirtyrate\_disk \quad (13)$$

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

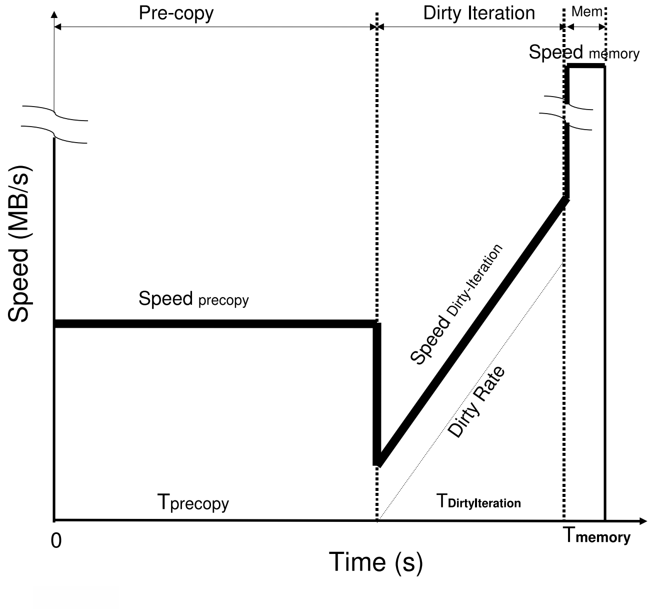


Fig. 2. An example of migration speeds in different stages.

We apply an algorithm which will be described in Section II-C2 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 that finishing the migration by the desired time is not feasible. Then it conducts disk I/O throttling to upper bound the disk dirty rate to a configurable fraction of the achievable migration speed.

Figure 2 illustrates how the migration speed might be controlled by Pacer during different migration stages. During pre-copy, Pacer aims to maintain a stable speed but adapts to workload changes if necessary. During the 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 the memory migration, the migration can typically proceed at a higher speed than the speed in the previous two stages because the bottleneck is most likely to be the network.

Other variants: Similar to the discussion in Section II-B1 for migration time model, speed control can readily be adapted to other hypervisor. As an example, for VMware, Equations 11 will be adjusted as follows:

$$speed_{Memory} = \text{NETWORK\_SPEED} - \text{IO\_RATE} \tag{14}$$

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

2) Maximal speed prediction and speed tuning: Due to the 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 the 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.

III. Evaluation

A. 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 with about 2500 lines of code. Two options are added to the migration command: (1) an option to enable migration prediction and report the predicted migration time periodically (2) an option to specify the desired migration time and let Pacer control the migration progress to achieve the specified desired finish time.

B. 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 RAM, 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 specified), the migration speed is restricted to be 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 prediction accuracy with a wide range of configured speeds, and to meet the time control requirement of various desired migration time, we modify QEMU at the destination machine not to write the received data to the disk. To ensure that the result is not

TABLE I
VMMARK WORKLOAD SUMMARY

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 II
PREDICTION ERRORS FOR THE VM SIZE-BASED PREDICTOR AND THE PROGRESS METER ARE SEVERAL ORDERS OF MAGNITUDE HIGHER THAN PACER

(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
VM size-based Predictor	326s	395s	519s	185s	698s	1157s
Progress 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
VM size-based Predictor	43s	74s	99s	46s	60s	122s
Progress Meter	41s	70s	94s	45s	51s	114s
Pacer	4s	6s	5s	5s	6s	4s

biased by the disabled writing, we run a set of experiments of enabling and disabling writing at the destination, vary the number of clients, and compare the average prediction error in both cases. The difference is less than $1s$. 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 $1s$ again. The results show that disabling writing does not bias the experiment results.

The experiment VMs run VMmark Virtualization Benchmark [50]. VMmark consists of five types of workloads: file server, mail server, database server, web server, and java server, with each representing different types of applications. Table I shows the configuration 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 generates I/O with a constant interval. Two dynamic models generate I/O following an exponential distribution ($\lambda = 10$, 50 or 90) or Pareto distribution ($PAR(\alpha, k)$ where $\alpha = 2$ and $k = 10$, 50, or 90). Each experiment is run for 3 times with different random number seeds. The results show very little variance ($< 0.1\%$). We believe that is because the VMmark workload is quite stable from run to run as our previous research [23] about VMmark workload shows.

The performance of prediction is evaluated by **prediction error**. The predictor computes and reports its prediction t_{pi} every N seconds from the beginning of migration until the migration finishes. After the migration, we evaluate the accuracy of the prediction by computing the absolute difference between the actual migration time t and the reported prediction time, and then report the average of those absolute differences: $\frac{\sum |t_{pi} - t|}{t/N}$. We optimize Pacer to avoid prediction spikes due to some sudden temporary workload shifts by generating a cumulative average predicted time over all past individual predicted times and using it as the reported prediction time.

C. Prediction of migration time

1) VM-size based predictor and progress meter do not work: In the following experiment, we will show that the VM-size based prediction method and a more dynamic method, progress meter, fail to give an accurate prediction of the migration time.

The VM-size based predictor uses the formula $\frac{storage\_size + memory\_size}{configured\_migration\_speed}$. This approach is commonly used when users want to predict the migration time.

Another dynamic predictor is also implemented for comparison. The predictor is called progress meter, which is based on the migration progress reported by QEMU. Whenever the migration progress increases by 1%, the predictor records the current migration time t and the 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 VM image sizes to represent the typical image sizes in industrial environments. 160GB is the size of an Amazon EC2 small instance and 8GB is the size of the VMmark file server image. We use a micro benchmark that repeatedly writes to a data region of the VM's virtual disk at a specified write rate. The size of the written region and the write rate vary to create different dirty set sizes and dirty rates during the migration.

Table II shows the results. The prediction errors for the VM-size based predictor and the progress meter are several orders of magnitude larger than those of Pacer, mainly because those two methods do not predict the time of the dirty iteration and memory migration. The prediction errors of those two methods scales up with higher write rates and larger written region sizes, while Pacer always achieves small prediction errors in all cases.

2) Pacer in face of uncertain dynamics: We vary multiple dimensions in the migration environment to demonstrate that Pacer performs well under different scenarios. We use the file server VM 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. Pacer computes and reports a predicted time every five seconds.

Figure 3 shows an example of the prediction process during migration. The experiment is based on the migration of a file server with 30 clients. There is additional competing traffic on

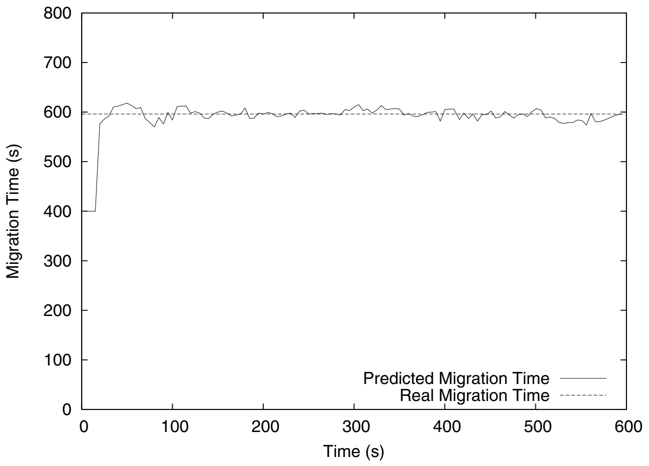


Fig. 3. The prediction of a VM (file server-30clients) migration. Pacer achieves accurate prediction from the very beginning of the migration.

the same hypervisor. The additional competing traffic follows the exponential distribution of average 10ms sleeping time. The actual migration time is 596s. In the first 20 seconds, Pacer predicts the migration time as 400 seconds because it does not have enough data for an accurate prediction. From 20 seconds onwards, its prediction time is very close to the actual migration time. The prediction error is $[0s, 26s]$ excluding the first 20 seconds. The average prediction error is 14s over the entire migration period and 7s for the period excluding the first 20 seconds.

Table III shows more scenarios for evaluating Pacer under different dynamic changes. The first three experiments have no additional competing traffic.

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 III 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 results show 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.

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.

According to the results and observations, an advantage of Pacer is that Pacer achieves accurate prediction from the very beginning of the migration. We take the prediction values in the first minute and compute the average prediction error for each experiment above. The resulting errors are within the range of $[2s, 12s]$, which is slightly larger than the average prediction error of the entire migration. The reason why Pacer achieves accurate prediction from the very beginning is because of the effective dirty set and dirty rate prediction algorithms. We will quantify the benefits of these algorithms in Section III-D3.

TABLE III
PREDICTION WITH PACER

	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

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

D. Best-effort migration time control

1) Dirty block prediction is critical for effective time control: We implement an adaptive time controller without dirty block prediction. The migration speed is computed by the formula $\frac{remain\_pre\_copy+existing\_dirty\_blocks}{remain\_time}$. Similar to the setup in Section III-C1, the experiment uses 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 IV shows the migration time deviation. The actual migration time of Pacer is very close to the desired time, with maximal deviation of [-1s,+6s]. The migration time of the controller without dirty block prediction exceeds the desired time up to 1528s and the deviation gets larger when the workload is more write intensive, because the controller lack the capability to predict the amount of remaining blocks for migration and thus it selects a wrong speed. We will show how the key components in Pacer help to reduce the deviation later in Section III-D3.

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 decrease the desired migration time way beyond anything feasible, the

TABLE IV
MIGRATION TIME DEVIATION FOR PACER IS MUCH SMALLER THAN THE CONTROLLER WITHOUT DIRTY BLOCK PREDICTION

(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
Controller w/o dirty block prediction	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
Controller w/o dirty block prediction	31s	47s	59s	54s	88s	110s
Pacer	1s	2s	-1s	1s	1s	2s

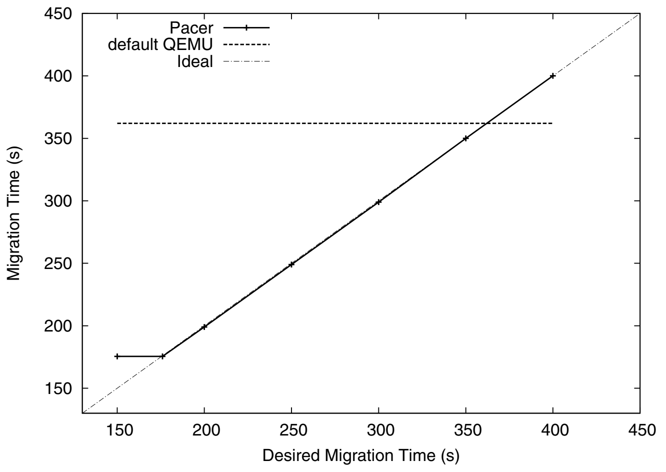


Fig. 4. Migration with different desired finish times. Pacer almost matches the ideal case when the desired time is larger than 176s. The deviation is very small in [-2s,2s].

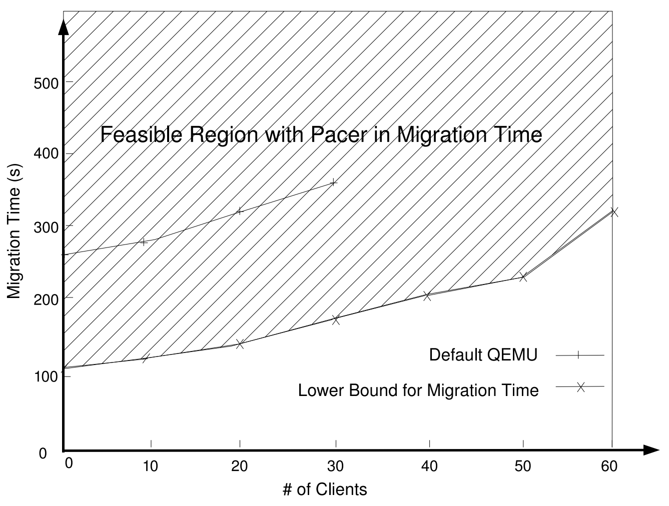


Fig. 5. Migration with different degrees of workload intensity. Any point in the feasible region can be realized by Pacer. The lower bound for migration time is limited by I/O bottleneck. Default QEMU can only follow a narrow curve in the region.

TABLE V
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

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	$[0, 0]$	$[-1, 0]$	$[-2, 0]$	$[-2, 0]$	$[-1, 1]$	$[-2, 0]$

I/O becomes the bottleneck, 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 shows, there exists a lower bound for migration time (minimal migration time) because of the I/O bottleneck. Pacer can adaptively pace the migration to achieve any target migration time in the feasible region above the smallest possible time for migration to complete, while default QEMU can only achieve one migration time for a specific number of clients. Moreover, when the number of clients increases above 35, QEMU cannot converge and the migration time becomes infinite. The reason is that QEMU uses a configured constant speed that will not increase when the I/O bandwidth becomes higher.

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 V 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 set the desired migration time to be 900s and use Pacer to control the migrations with different types of workload. Table VI shows that Pacer can achieve desired migration time with a small deviation in $[-2s, +2s]$.

Vary additional competing traffic: To test whether Pacer can achieve desired migration time when different levels of I/O interference exist, we run the following experiment with the program in Section III-B to generate additional competing I/O traffic. The migrated VM runs file server with 30 clients. The desired migration time is 264s. Table VII 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) 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 those algorithms. The workload is file server. As Table VIII shows, the actual migration time will exceed the desired migration time significantly in the case that there is no prediction algorithm. When only the dynamic dirty set prediction algorithm is added into the system, the accuracy of migration time improves but still exceeds the desired time. When both the dirty set and dirty

TABLE VI
MIGRATION TIME ON DIFFERENT TYPES OF WORKLOAD. PACER CAN ACHIEVE THE DESIRED MIGRATION TIME

Workload	Desired Migr Time(s)	Pacer Migr Time(s)
fs-30 clients	900	898
ms-200 clients	900	902
js-16 clients	900	901
ws-100 clients	900	900
ds-16 clients	900	901

TABLE VII
MIGRATION TIME FOR PACER WHEN THE ADDITIONAL COMPETING TRAFFIC VARIES. PACER CAN ACHIEVE THE DESIRED MIGRATION TIME WITH A SMALL FINISH TIME DEVIATION

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 predictor 90ms	-3	0	-5
Expo (ave 90ms)	-5	-2	1
Pareto (ave 90ms)	0	2	1

TABLE VIII
IMPORTANCE OF DYNAMIC DIRTY SET AND DIRTY RATE PREDICTION. WITHOUT ANY OF THESE ALGORITHMS, IT IS HARD TO ACHIEVE DESIRED MIGRATION TIME

Work load	Desired Time(s)	Pacer without 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 IX
IMPORTANCE OF SPEED SCALING UP ALGORITHM

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

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 the file server with 30 clients. Additional competing traffic is generated by a constant interval of 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 IX. Without the speed scale-up algorithm, migration will considerably exceed the desired time in all three experiments.

E. Overhead of Pacer

In this experiment, we measure the overhead introduced by Pacer in terms of time and space. For example, for best effort time control, we run migration with Pacer for the file server workload with 60 clients and a desired migration time of 400s. We measure the computation time of Pacer in each round. We observe that the computation time is 28.24ms at the beginning of migration. As the migration progresses and more blocks in the dirty set are determined, the computation time drops to below 1ms in the final stage of migration. Overall, Pacer on average only incurs 2.4ms of computation time for each 5 second interval. The overhead is 0.05% ,which is negligible. The space overhead in terms of additional memory required to run Pacer compared to default QEMU is less than 1MB. Prediction consumes less computation resource than best-effort time control.

We also evaluate the overhead introduced by Pacer for each disk I/O write operation during migration. The default QEMU already has a dirty block tracking function to track each disk write operation during migration. Pacer just leverages the existing tracking system and performs a simple update for average write interval. We ran experiments to measure the disk I/O write latency with and without Pacer. The average disk I/O latency at millisecond accuracy and throughput at MB/s accuracy is the same with and without Pacer.

We also measure the application throughput and response time on the file server during migration with and without Pacer. The results show no side effect on application performance with Pacer. In summary, the overhead of Pacer is small and has no impact on the performance of the application.

F. Potential robustness improvements

Pacer could be improved further by including mechanisms to mitigate the negative impact of rare case when migration environment variables are not steady. Firstly, Pacer is an adaptive system with a fixed adaptation interval (5s) in the current design. Instead, a flexible interval can be applied when Pacer detects that the workload intensity or the network available bandwidth varies significantly. Reducing the adaptation interval will improve the adaptivity but it also incurs more overhead. By adjusting the adaptation interval, we can make a trade-off between the speed of adaptation and overhead. Secondly, we can test the migration environment, e.g. network bandwidth, against expected patterns to find out whether any increasing or decreasing trend exists. These mechanisms will be considered in our future work.

IV. 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 Rice University 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 VMs are run without KVM's hardware virtualization support which has lower performance than if there were hardware virtualization support.

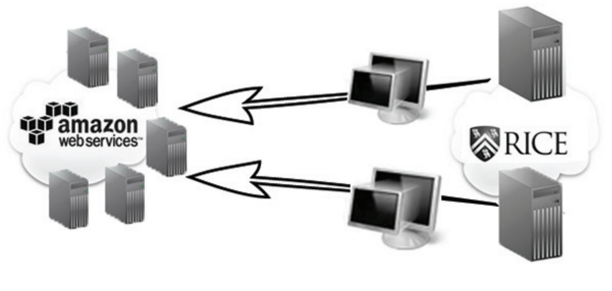


Fig. 6. EC2 demonstration setup.

TABLE X
PREDICTION ACCURACY WITH PACER IN EC2

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

A. 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. Firstly, we use "iperf" to measure the network throughput for 200 seconds. We find that when transmitting data from Rice to EC2, the throughput increases gradually and linearly for a surprisingly long 30s before reaching the peak 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. Secondly, we use "scp" linux command to copy a 8GB file from Rice to EC2 to measure the 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.

B. 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 X. The first observation is that the accuracy of the prediction does not decrease as the workload increases. Secondly, given the fact that the network and disk speeds are quite unstable, Pacer can still predict with an average absolute error of about 5s. We find that, if disk write at the destination is 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.

TABLE XI
MIGRATION TIME CONTROL WITH PACER IN EC2

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

C. 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 XI. 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.

V. CONCURRENT MIGRATION

In this section, we will briefly introduce how Pacer helps in the use case of coordinating migrations of multiple VMs (refer to the second use case in Section I-D). Enterprises usually have applications consisting of multiple interacting components. To coordinate the concurrent migration of VMs and avoid application components getting split between distant datacenters during migration, we design a centralized controller by leveraging Pacer's prediction and time control capabilities. The controller continuously gathers and analyzes the predicted migration time for each VM, dynamically updates the desired migration time for each VM with a feasible target time, and thus paces the migration of the whole set of VMs. In our current design, we apply a simple target finish time selection policy – the maximum expected finish time among the related VMs is chosen as the uniform target finish time. When Pacer on a VM receives a new target finish time from the controller, it adjusts its migration speed as explained in Section II-C to adapt to the new target finish time.

On the EC2 test bed, we migrate a web service (SPECweb 2005 in VMmark) from Rice to Amazon EC2. The web service contains a frontend apache server and a backend database server. The two servers are deployed on two VMs. The image sizes for the frontend and backend VMs are 8GB and 16GB respectively. The experiment migrates the web service in three different methods and illustrates how Pacer can reduce the time of performance degradation, which is defined as the period that one VM runs at the source and the other VM runs at the destination. The degradation happens when two VMs finish migration at different times. During the degradation period, the communication between frontend and the backend goes across the wide area network between Rice and EC2. The first method is sequential migration. It migrates the frontend VM first. When the first VM migration finishes, it starts the second VM migration. The second method is parallel migration which starts both VM migrations at the same time. The sequential migration and parallel migration result in 577s and 394s of degradation time. Pacer's coordinated migration function is able to reduce the performance degradation time from hundreds of seconds to 3s.

VI. Related Work

To our knowledge no previous work is directly comparable to Pacer. VMware ESX[16], KVM [14] and XEN [15] only report the live migration progress percentage in their user interfaces. They do not provide any prediction or time control functions of live migration progress. In section III-C1, we already show that predicting or controlling migration time based on progress (called progress meter) does not work.

There exists related work on setting the speed or estimating the time of VM migration. Breitgand *et al.* [43] 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.* [44] 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. Liu *et al.* [51] investigate methodologies to quantitatively predict the migration performance and energy cost. Similarly it focuses on *CPU/memory-only* migration and the prediction model requires prior knowledge about the workloads at the hypervisor. In contrast, Pacer addresses the issue of *full* VM migration and performs adaption to dynamically predict and control migration time.

Relative to the above related work, not only does Pacer address a different set of problems in migration progress management for *full* VM migration, Pacer also takes a systems 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.

Several optimization techniques have been proposed to reduce migration traffic and therefore reduce migration time. For example, compression techniques are applied to memory migration [52] and storage migration [53] to avoid transmitting duplicate data. A scheduling algorithm is used to reduce the amount of dirty blocks by reordering the migration sequence of storage blocks [23]. A prefetching strategy is designed to tolerate rapid changes of disk state during storage migration when the VM runs I/O intensive workload [31]. Pacer is complementary to these techniques as it aims to provide accurate migration time prediction and control. It is possible to extend Pacer to function alongside these optimization techniques by adjusting Pacer's migration time model accordingly. This is an area for our future research.

Beyond VM migration, there is also interesting related work in *disk-data-only* migration. In [54], 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.* [55] and Zhang *et al.* [56] 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.

VII. Conclusions

We have argued that the live VM migration progress management functions are much needed in the hybrid cloud computing environment. Our contribution is Pacer – the first system capable of accurately predicting the migration time, coordinating the migrations of multiple application components to minimize the performance degradation, and managing the progress so that the actual migration finishing time is as close to the desired finish time as possible. Through extensive experimentation, including a realistic commercial hybrid cloud scenario with Amazon EC2, we show that Pacer is highly effective.

Recently we have extended Pacer by providing a new function for prediction migration time before migration begins. The main addition is to monitor the disk I/O workload and to measure the available network bandwidth for a short period of time, e.g. 3 minutes, and to use these observations for migration time prediction. We have found that the prediction accuracy is as good as the prediction during migration. The new function is helpful for operators for planning and scheduling cloud management tasks.

As future work, we are extending Pacer to analyze and model the behavior of complex enterprise applications so as to automatically and optimally manage the migration of such complex applications.

References

[1] "Cloud computing," http://www.ibm.com/cloud-computing/us/en/what-is-cloud-computing.html.
[2] "Virtustream," http://www.virtustream.com/.
[3] "Mainstream cloud computing," http://www.forbes.com/sites/reuvencohen/2013/04/16/the-cloud-hits-the-mainstream-more-than-half-of-u-s-businesses-now-use-cloud-computing/.
[4] Gartner, "Gartner says worldwide cloud services market," http://www.gartner.com/newsroom/id/2163616/, 2012.
[5] "Amazon Web Service," http://aws.amazon.com.
[6] "IBM Homepage," http://www.ibm.com.
[7] "Googleappengine," https://developers.google.com/appengine/.
[8] "Data Center," https://en.wikipedia.org/wiki/Data_center.
[9] wiki, "Multi-tier architecture," http://en.wikipedia.org/wiki/Multitier_architecture.
[10] "Cloud Computing," http://en.wikipedia.org/wiki/Cloud_computing.
[11] C. Poelker, "Why virtualization is the foundation of cloud computing," http://tinyurl.com/cdtcyqz, 2009.
[12] T. Wood, P. Shenoy, K. K.Ramakrishnan, and J. V. der Merwe, "Cloudnet: dynamic pooling of cloud resources by live WAN migration of virtual machines," in *2011 ACM VEE*.
[13] "Virtualize your IT Infrastructure," http://www.vmware.com/virtualization/.
[14] KVM, "Kernel based virtual machine," http://www.linux-kvm.org/page/Main_Page.
[15] XEN, "XEN Project," http://www.xen.org, Jan. 2009.
[16] "VMware ESX," http://www.vmware.com/products/vsphere/esxi-and-esx/index.html.
[17] Microsoft, "Hyper-V live migration FAQ," http://technet.microsoft.com/en-us/library/ff715313(v=ws.10).aspx, Jan. 2012.
[18] 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. 2005 Conference on Symposium on Networked Systems Design & Implementation*, pp. 273–286.
[19] T. Wood, P. Shenoy, A. Venkataramani, and M. Yousif, "Black-box and gray-box strategies for virtual machine migration," in *2007 NSDI*.
[20] A. W. S. Blog, "Animoto—scaling through viral growth," http://aws.typepad.com/aws/2008/04/animoto—scali.html, Apr. 2008.
[21] D. Gottfrid, "The New York Times Archives + Amazon Web Services = TimesMachine," http://open.blogs.nytimes.com/ 2008/05/21/the-new-york-times-archives-amazon-web- services-timesmachine/, May 2008.

[22] M. Armbrust, A. Fox, R. Griffith, and *et al.*, “Above the clouds: a Berkeley view of cloud computing,” EECS Department, University of California, Berkeley, tech. rep. UCB/EECS-2009-28, Feb 2009. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.html

[23] J. Zheng, T. Ng, and K. Sripanidkulchai, “Workload-aware live storage migration for clouds,” in *2011 ACM VEE*.

[24] W. Ashford, “Hybrid clouds most popular with UK business, survey reveals.” Available: http://tinyurl.com/868pxzd, Feb. 2012.

[25] N. Bridge, “Future of cloud computing survey.” Available: http://tinyurl.com/7f4s3c9, Jun. 2011.

[26] K. C. Tofel, “Forget public; private clouds: the future is hybrids!” Available: http://tinyurl.com/bsmsj9p, Jun. 2011.

[27] “Stateless protocol,” http://en.wikipedia.org/wiki/Stateless_protocol.

[28] P. Padala, “Understanding live migration of virtual machines.” Available: http://tinyurl.com/24bdaza, Jun. 2010.

[29] D. TechCenter, “Hyper-V R2 live migration FAQ.” Available: http://tinyurl.com/c8rayf5, Nov. 2011.

[30] C. Steele, “Virtual machine migration FAQ: Live migration, P2V and more.” Available: http://tinyurl.com/cxavodk, Aug. 2010.

[31] B. Nicolae and F. Cappello, “Towards efficient live migration of I/O intensive workloads: a transparent storage transfer proposal,” in *2012 ACM HPDC*.

[32] “VMware Forum,” http://tinyurl.com/ccwd6jg, 2011.

[33] “VMware Forum,” http://tinyurl.com/cr6tqnj, 2011.

[34] “VMware Forum,” http://tinyurl.com/bmlnjqk, 2011.

[35] “VMware Forum,” http://tinyurl.com/7gttah2, 2009.

[36] “Xen Forum,” http://tinyurl.com/d5v8j9p, 2008.

[37] “Xen Forum,” http://tinyurl.com/d477jza, 2011.

[38] “Xen Forum,” http://tinyurl.com/c7tyg94, 2011.

[39] “Xen Forum,” http://tinyurl.com/c37he9g, 2008.

[40] “VMware Forum,” http://tinyurl.com/7azb3xt, 2012.

[41] “VMware Forum,” http://tinyurl.com/d4qr2br, 2011.

[42] Y. Wu and M. Zhao, “Performance modeling of virtual machine live migration,” in *Proc. 2011 IEEE International Conference on Cloud Computing*.

[43] D. Breitgand, G. Kutiel, and D. Raz, “Cost-aware live migration of services in the cloud,” in *2011 USENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services.*

[44] S. Akoush, R. Sohan, A. Rice, A. W.Moore, and A. Hopper, “Predicting the performance of virtual machine migration,” in *2010 IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems.*

[45] A. Verma, G. Kumar, R. Koller, and A. Sen, “Cosmig: modeling the impact of reconfiguration in a cloud,” in *2011 IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems.*

[46] M. Hajjat, X. Sun, Y. Sung, D. Maltz, S. Rao, K. Sripanidkulchai, and M. Tawarmalani, “Cloudward bound: planning for beneficial migration of enterprise applications to the cloud,” in *2010 ACM SIGCOMM Computer Commun. Rev.*

[47] M. Nelson, B.-H. Lim, and G. Hutchins, “Fast transparent migration for virtual machines,” in *Proceedings 2005 Usenix Technical Conference.*

[48] “VMWare Storage vMotion,” http://www.vmware.com/products/storage-vmotion/overview.html.

[49] C. Staelin and H. Garcia-Molina, “Clustering active disk data to improve disk performance,” Department of Computer Science, Princeton University, tech. rep. CS-TR-283-90, Sep. 1990.

[50] VMWare, “VMmark Virtualization Benchmarks,” http://www.vmware.com/products/vmmark/, Jan. 2010.

[51] H. Liu, C.-Z. Xu, H. Jin, J. Gong, and L. Xiaofei, “Performance and energy modeling for live migration of virtual machines,” in *Proc. 20th International Symposium on High Performance Distributed Computing*, pp. 171–182.

[52] S. Hacking and B. Hudzia, “Improving the live migration process of large enterprise applications,” in *Proceedings 2009 International Workshop on Virtualization Technologies in Distributed Computing.*

[53] C. P.Sapuntzakis, R. Chandra, B. Pfaff, J. Chow, M. S.Lam, and M. Rosenblum, “Optimizing the migration of virtual computers,” in *Proc. 2002 Symposium on Operating Systems Design and Implementation.*

[54] C. Lu, C. A. Alvarez, and J. Wilkes, “Aqueduct: online data migration with performance guarantees,” in *Proc. 2002 USENIX Conference on File and Storage Technologies.*

[55] K. Dasgupta, S. Ghosal, R. Jain, U. Sharma, and A. Verma, “Qosmig: adaptive rate-controlled migration of bulk data in storage systems,” in *Proc. 2005 ICDE.*

[56] J. Zhang, P. Sarkar, and A. Sivasubramaniam, “Achieving completion time guarantees in an opportunistic data migration scheme,” in *Proc. 2006 SIGMETRICSREVIEW.*

Jie Zheng is a Ph.D. student in Computer Science at Rice University. She received a B.S. in Computer Science from Xiamen University in 2004, a M.S. in Computer Science from Chinese Academy of Science in 2007 and a Ph.D. in Computer Science from Rice University in 2013. Her research interests are in cloud computing and virtualization.

T. S. Eugene Ng is an Associate Professor of Computer Science at Rice University. He is a recipient of a NSF CAREER Award (2005) and an Alfred P. Sloan Fellowship (2009). He received a Ph.D. in Computer Science from Carnegie Mellon University in 2003. His research interest lies in developing new network models, network architectures, and holistic networked systems that enable a robust and manageable network infrastructure.

Kunwadee (Kay) Sripanidkulchai is a senior researcher at the National Electronics and Computer Technology Center (NECTEC) in Thailand. Her research interests are in services delivery, cloud computing, and networking. In her previous role as a Research Staff Member at the IBM T.J. Watson Research Center, she led a team to develop processes and tools for a new virtualization service offering for IBM Global Technology Services. She has co-authored a number of papers and patents in networking and cloud computing, and has taught at several universities including Columbia University, Mahidol University, Khonkaen University and Thammasat University. She received a Ph.D. and M.S. from the Department of Electrical and Computer Engineering at Carnegie Mellon University and a B.S. from the Department of Electrical Engineering at Cornell University.

Zhaolei Liu is a graduate student from Rice University, Department of Computer Science. He got his B.S. in Computer Science from Rice University within three years. His current research interest is networking and distributed systems.