Resource Availability Based Performance Benchmarking of Virtual Machine Migrations

Senthil Nathan, Purushottam Kulkarni and Umesh Bellur
Department of Computer Science and Engineering
Indian Institute of Technology, Bombay
Mumbai, India
{cendhu,puru,umesh}@cse.iitb.ac.in

ABSTRACT

Virtual machine migration enables load balancing, hot spot mitigation and server consolidation in virtualized environments. Live VM migration can be of two types - adaptive, in which the rate of page transfer adapts to virtual machine behavior (mainly page dirty rate), and non-adaptive, in which the VM pages are transferred at a maximum possible network rate. In either method, migration requires a significant amount of CPU and network resources, which can seriously impact the performance of both the VM being migrated as well as other VMs. This calls for building a good understanding of the performance of migration itself and the resource needs of migration. Such an understanding can help select the appropriate VMs for migration while at the same time allocating the appropriate amount of resources for migration. While several empirical studies exist, a comprehensive evaluation of migration techniques with resource availability constraints is missing. As a result, it is not clear as to which migration technique to employ under a given set of conditions. In this work, we conduct a comprehensive empirical study to understand the sensitivity of migration performance to resource availability and other system parameters (like page dirty rate and VM size). The empirical study (with the Xen Hypervisor) reveals several shortcomings of the migration process. We propose several fixes and develop the Improved Live Migration technique (ILM) to overcome these shortcomings. Over a set of workloads used to evaluate ILM, the network traffic for migration was reduced by 14-93% and the migration time was reduced by 34-87% compared to the vanilla live migration technique. We also quantified the impact of migration on the performance of applications running on the migrating VM and other co-located VMs.

Categories and Subject Descriptors

C.4 [**PERFORMANCE OF SYSTEMS**]: [Performance attributes]; D.4.8 [**OPERATING SYSTEMS**]: Performance—*Measurements*

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
ICPE'13, April 21–24, 2013, Prague, Czech Republic.
Copyright 2013 ACM 978-1-4503-1636-1/13/04 ...$15.00.

Keywords

Virtualization; Live Migration

1. INTRODUCTION

Advances in virtualization technologies have enabled hosting of applications and services in isolated and resource-guaranteed virtual machines. Typically, a single physical machine (PM) hosts multiple applications, each executing in it's own virtual machine (VM). In addition, today's virtualization technologies [4, 12, 6] allow VMs to be migrated from one physical machine to another during execution (VM live migration). This feature along with dynamic resource provisioning for a virtual machine can be used to either balance load [15, 11] across hosts (physical machines) in a data center or to consolidate virtual machines [5, 14, 7] onto fewer physical machines for power savings and reducing operational costs (i.e., server consolidation).

Live VM migration entails an iterative process which transfers memory contents of a VM from one host to another. Since the VM is executing, memory pages are dirtied while migration happens. Thus, multiple iterations are required to transfer the state. Further, the migration can be of two types– a non-adaptive process in which a VM's memory pages are transferred at the maximum possible rate, and an adaptive process in which the rate of page transfers is tied to the page dirty rate. Heuristics for migration coupled with memory usage patterns of the VM impact the performance of live migration. The important performance/cost metrics of migration are: *migration time*—the total duration for the migration to complete; *down time*—the duration of time for which the virtual machine is not executing (during switchover); CPU and network resources consumed for migration; and performance implications on migrating VM and other co-hosted VMs.

The resource management heuristics [15, 11, 5, 16] employed in today's virtualized data centers to mitigate overload situations are migration-aware and base their migration decisions on factors such as resource utilization levels across machines, the "size" of the virtual machine to be migrated, the availability of resources at target machines, the number of migrations to achieve dynamic provisioning goal, etc. Accurate characterization of the cost-benefit trade-off between performance/cost metrics and configurations of the migration process is vital for migration-based resource management. Consider the following scenario, involving a cluster of PMs, each hosting several VMs (a typical hosting situation). A dynamic resource management problem is to modify resource allocations for the VMs to ensure SLA guarantees of

services/applications executing in the VMs. In such a scenario, suppose the resources allocated to a VM need to be increased but its host PM has no spare capacity (an overload condition). A migration based reconfiguration of VMs migrate the highly loaded VM to a lightly loaded PM or migrate other VMs to create additional local capacity. For example, some existing approaches select VMs for migration based on the size (e.g., memory size) of the VMs. Verifying the efficacy and applicability of this basis is important. Selection of input parameters for migration and characteristics of the VM (e.g., memory size, memory usage dynamics etc.) impacts the cost-benefit trade-off of migration-enabled resource management. This manifests into several interesting questions, (i) given a cost metric (e.g., minimize interference to other VMs due to migration) which set of VMs should be migrated?, (ii) under which conditions would adaptive migration be preferred over non-adaptive migration?, (iii) which destination host should be selected, to minimize migration costs?, (iv) how much quantity of the CPU and network resources should be allocated to the migration process itself, to minimize migration costs?

While many studies related to migration performance exist in current literature, none of them answer all the questions mentioned above. A comprehensive empirical study of migration performance and related cost-benefit analysis will enable improved decision making for managing resources. Our contributions are as follows:

• We performed a comprehensive empirical study of VM live migration and compared the performance of adaptive live migration and non-adaptive live migration with respect to parameters which affect migration performance, such as VM size, page dirty rate, resource availability.

• We identified and implemented optimizations to the migration technique based on the above studies. In particular, we propose an enhanced migration process—the Improved Live Migration (ILM) technique.

• We quantified the performance degradation in the migrating VM and in other VMs during adaptive live migration, non-adaptive live migration and ILM technique.

In the rest of the paper, we present a short background of live migration and motivate the need for a migration performance study in Section 2. In Section 3, we describe the experimental setup and methodology. An empirical comparison of *non-adaptive* live migration and *adaptive* live migration is presented in Section 4. The Improved Live Migration techniques and its evaluation are presented in Section 5. In Section 6, we present the impact of migration on application's performance. Related work is presented in Section 7, while Section 8 concludes the paper.

2. BACKGROUND

Virtual machine live migration [10] is a process of copying memory pages of the virtual machine to the destination machine in rounds—known as as the *iterative pre-copy* phase. Iteration #1 transfers all memory pages allocated to a VM. Subsequent iterations transfer only pages dirtied during the previous iteration. However, pages dirtied before their transmission are **skipped** from transmission in the current iteration, based on the expectation that these pages may get dirtied again. The Xen live migration process [4] transfers pages in batches, i.e., a set of pages at a time instead of one page at a time. After a batch is transferred, the migration process locates pages that were dirtied (with the help of Xen shadow page tables) during the last batch transfer time. The yet-to-be-transferred dirty pages are added to a **skip list**. The rate at which pages are transferred is based on two approaches—**non-adaptive** and **adaptive**. Non-adaptive live migration transfers pages at the maximum available network capacity, whereas the adaptive technique adjusts the rate of page transfers to the rate at which pages are dirtied by the virtual machine. Iteration #1 transfers pages at the rate of r_{min} Mbps, and subsequent iterations (i>1) transfer pages at a rate r_i Mbps, which is equal to

$$r_i = \min\left\{\max(d_{i-1} + \delta, r_{min}), r_{max}\right\} \quad (1)$$

where r_{min}, r_{max} and δ are configurable variables. d_{i-1} is the **page dirty rate** (unit in Mbps) during iteration #(i-1) and is equal to,

$$d_{i-1} = \frac{\#\text{pages dirtied in iter. \#(i-1)} \times \text{page\_size}}{\text{time taken by iter. \#(i-1)}} \quad (2)$$

Based on the page dirty rate dynamics, the *iterative pre-copy* may never complete. To avoid such situations, a *stop-and-copy* condition is used. The switch from *iterative pre-copy* to *stop-and-copy* occurs when either one of the following four conditions occur:

1. a fixed number n of *pre-copy* iterations are completed.
2. the total network traffic generated is greater than a threshold, i.e., m times greater than the VM size.
3. the number of pages dirtied during the current iteration is less than a threshold, i.e., less than p pages.
4. the page dirty rate of the last iteration is greater than a threshold, i.e., greater than r_{max} Mbps **and** the number of pages sent in the current iteration is greater than the previous iteration (imply a decrease in the page dirty rate).

With *non-adaptive* live migration, only the first three stopping conditions are applicable, while all four conditions are applicable for *adaptive* live migration. In the *stop-and-copy* phase, the virtual machine being migrated is suspended and pages dirtied during the last *iterative pre-copy* round are transferred at a rate equal to the available network capacity (with both *adaptive* and *non-adaptive* live migration techniques). The performance of live migration process can be evaluated in terms of:
(i) **migration time**—the time between the initiation of migration and the successful completion, and
(ii) **downtime**—the time for which the service provided by the VM is unavailable, i.e., the time taken by the *stop-and-copy* phase.

The *cost* of the live migration process can be evaluated in terms of: (i) total **CPU utilization** at the source and destination PMs to transfer and receive pages, respectively, (ii) the total **network traffic** generated during migration—i.e., total number of pages transferred (some possibly multiple times) during migration, (iii) **migration rate**—the rate at which pages are transferred. Cost of migration is a function of these parameters as each of them can impact the performance of applications running in virtual machines at the source and destination PMs as well as other machines which share the network link.

Since the migration process consumes memory, network and CPU resources, the following parameters affect the *performance* and *cost* of the migration process:

1. Memory size allocated to the virtual machine.

2. Available CPU resource levels at the source and destination physical machines.
3. Available network capacity between source and destination physical machines.
4. The rate at which the application dirties memory pages.

2.1 Need for Migration Performance Study

Virtual machine migration based dynamic resource management opens up several questions:

• What amount of CPU and network resources should be allocated for the migration process? Migration performance itself is proportional to the resources allocated to it. However, resource allocation for the migration process usually comes at a cost of degraded performance in other VMs, from which resources are taken away to enable migration. Should we then treat the migration activity as secondary? Or should we speed it up in order to free resources on the PM from which the VM is migrating? These questions can only be answered by a thorough understanding of the dependencies between migration performance and resource allocation.

• Given an operating state of the VMs, which migration technique is best suited to minimize migration cost?

• What parameters should be considered while selecting a VM for migration? Existing solutions either select a VM randomly or consider only size of the VM to reduce migration time, downtime and migration cost. Is considering only VM size adequate?.

• What are the factors need to be considered while selecting a new destination PM for a VM? Existing approaches select a PM which is least loaded. Should the network bandwidth availability between source and destination PM, and CPU availability at destination PM for migration process be considered?

• How do the four stop-and-copy conditions impact the performance of live migration?

1. It is not clear how the number of iterations affects migration time and downtime.
2. What should be the threshold for iterative pre-copy condition 2?
3. The iterative pre-copy condition 4 is applicable for adaptive live migration but not for non-adaptive live migration. It is not clear what will be the impact of condition 4 on adaptive live migration.

While many studies of migration performance exist in the literature, we are not aware of any that have explored all of these questions. A comprehensive empirical study of migration performance can answer these and related questions. This will help to make effective resource-availability based decisions for virtual machine based load balancing and server consolidation.

3. METHODOLOGY AND SETUP

In order to compare the performance of adaptive live migration and non-adaptive live migration and answer other questions, migrations of VMs hosting four different workloads are performed by (i) varying the VM size and by keeping the load level of each workload constant, (ii) varying the load level of each workload (to create different page dirty rates) and by keeping the VM size and available resources constant, and (iii) varying the available resources and by keeping the load level of each workload and virtual machine size constant. We measured migration time, downtime, total network traffic generated and CPU resource utilization. We performed experiments on the Xen virtualization platform.

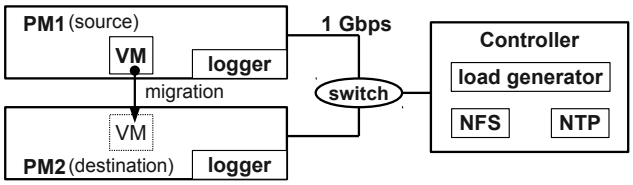


Figure 1: Experimentation setup for performance benchmarking.

We used the following workloads as application running within VMs (to be migrated). We picked these workloads as each one has a different page dirty pattern.

1) **Linux idle**: an idle Linux installation that does not host any application. This workload is mainly used as a baseline for comparison.

2) **RUBiS** [2]: an auction site prototype modeled after eBay.com. Clients perform browse and bid on existing items, register and sell items. We used the *PHP* implementation of RUBiS v1.4.3 with *MySQL*. We hosted the RUBiS web server (Apache/PHP) and database server (MySQL) on two different virtual machines. This workload results in high CPU usage and low I/O usage. The load was varied by varying the number of clients in the RUBiS workload generator.

3) **File Server**: an HTTP file server that hosts several files of different sizes. We used four clients to download four different files of size ranging from 1 GB to 4 GB using *wget*. This is mainly an I/O intensive workload. Unless otherwise specified, the default transfer rate is 240 Mbps. We controlled the transfer rate with the help of *wget --rate-limit* parameter.

4) **Kernel Compile**: a CPU intensive workload that compiles the Linux kernel v2.6.39 with the default configuration. Unless otherwise specified, the default number of threads used for compilation was 2.

Migrations of these workloads were performed using the setup shown in Figure 1. Our setup consisted of five physical machines, each with a 2.8 GHz Intel Core i5 760 CPU (4 cores), and 4GB of main memory. Out of the five physical machines, three acted as a *controller* whereas the other two physical machines (PM1 and PM2) were installed with *Xen hypervisor v4.0.1*. All five machines were connected through a 1 Gbps D-Link DGS-1008D switch. All physical machines and virtual machines were installed with the *Ubuntu Server 10.04 64-bit* Linux distribution.

We used three *controllers* to make sure that it does not become a bottleneck during the experiments. Each *controller* acted as a Network File System (NFS) server (i.e., shared storage) providing a specific set of virtual machine's disk and swap images. It also acted as Network Time Protocol (NTP) server (for time synchronization). Each *controller* executed a *load generator* daemon which issued commands to (i) create or destroy a virtual machine, (ii) specify the size of a VM and allocate CPU for the migration process, (iii) migrate the VM from PM1 to PM2, and (iv) generate load for the virtual machines.

The *logger* daemon executed in the management domain (Dom0) of PM1 and PM2. The Dom0 runs the Xen man-

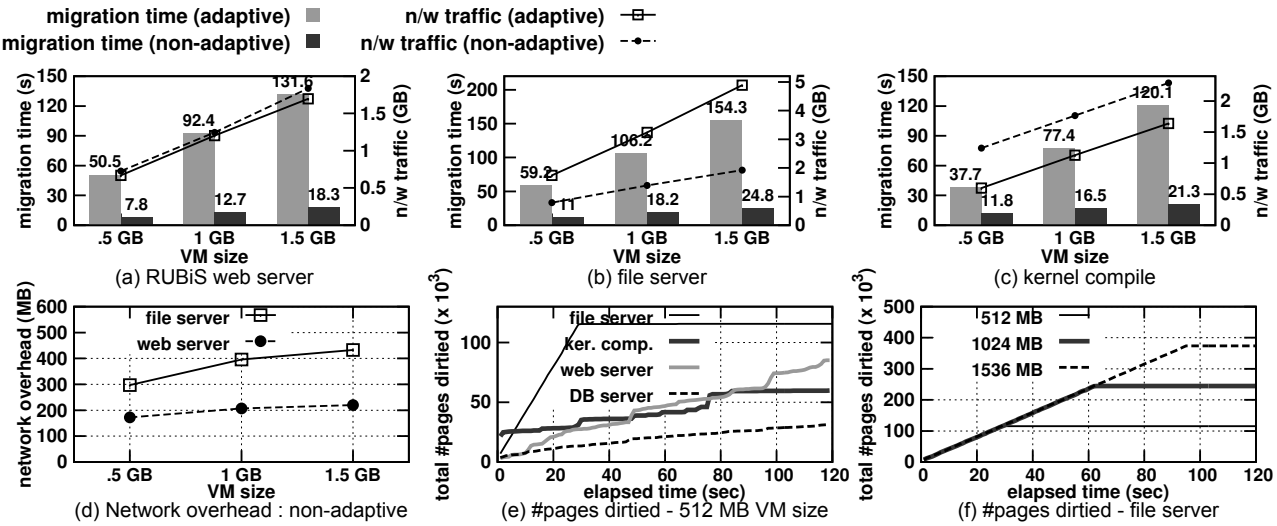


Figure 2: Effect of VM size on the performance of live migration

Table 1: Application's downtime during live migration

	VM size (GB)								
	0.5 GB	**1** GB	**1.5** GB	**0.5** GB	**1** GB	**1.5** GB	**0.5** GB	**1** GB	**1.5** GB
Benchmark	**adaptive migration rate** (Mbps)			**adaptive downtime** (ms)			**non-adaptive downtime** (ms)		
RUBiS web server	122	108	106	33	37	43	33	35	41
RUBiS db server	108	107	106	27	32	34	26	31	33
kernel compile	132	120	112	541	640	696	362	377	475
file server	243	252	263	822	988	1149	23	28	28

agement tool stack that performs the migration of virtual machine. The *logger* daemon recorded the CPU utilization of Dom0 (using *Xentop*[1]) and network utilization (using */proc/net/dev* file) during migration. The duration of each iteration, number of pages dirtied, the number of pages transmitted, the migration rate, etc. during each *pre-copy* iteration were logged in */var/log/xen/* by the Xen hypervisor. The Dom0 of PM1 and PM2 were configured with 512 MB of main memory and 1 CPU core for all the experiments. The *logger* daemon utilized less than 2% of Dom0 CPU per second and was considered as negligible. In all experiments, a virtual machine was migrated from PM1 to PM2. Each migration experiment was repeated 10 times and results reported are averaged over 10 trails.

In Xen Hypervisor, by default, the following values were assigned to variables in iterative pre-copy termination conditions, i.e., n = 29 iterations, m = 3, p = 50 pages and r_{max} = 500 Mbps. In equation (1), the variables δ and r_{min} were assigned with 50 Mbps and 100 Mbps, respectively.

4. COMPARISON BETWEEN ADAPTIVE AND NON-ADAPTIVE LIVE MIGRATION

In this section, we compare the performance of two variants of the live migration process and try to answer the questions presented in Section 2.1.

4.1 Change in VM size

In a virtualized server environment, each virtual machine is allocated some amount of memory along with other resources to handle its peak workload. To evaluate the performance of adaptive live migration and non-adaptive live migration, we migrated VMs of different memory sizes hosting the four workloads described in Section 3.

Figures 2(a), (b) and (c) plot the migration time and the total volume of network traffic generated over different memory sizes for both *adaptive* and *non-adaptive* live migration.

Observation 1: *The time taken by adaptive live migration was three to eight times* ***higher*** *than that of non-adaptive live migration.* This is because migration rates with adaptive live migration were three to eight times lower compared to non-adaptive live migration. Table 1 shows the migration rate of adaptive live migration. The network bandwidth utilized by non-adaptive live migration for the file server workload was 640 Mbps (which was the available capacity), whereas for the other workloads it was observed to be 830 Mbps to 900 Mbps (network bandwidth requirement of other workloads was lower as compared to the file server workload).

Observation 2(a): *The total volume of network traffic generated (i.e., cost of migration) was on average 2.45 times* ***higher*** *with adaptive live migration for the file server workload as compared to non-adaptive live migration.* The reasons for this result are: (i) *Adaptive* live migration spent more time in each iteration as compared to *non-adaptive* live migration as the migration rate was low. Further, the number of unique pages dirtied increased significantly over time for the file server workload – refer Figure 2(e). Hence, the number of unique pages dirtied and the number of pages transferred during an iterative pre-copy iteration were always higher with *adaptive* live migration compared to the corresponding iteration in non-adaptive live migration. (ii) Both *adaptive* live migration and *non-adaptive* live migration with the file server workload executed more or less same number of iterations (≈10 iterations).

Observation 2(b): *For the kernel compile workload, the total network traffic generated was on average 1.75 times* ***lower*** *with adaptive live migration compared to non-adaptive live migration. Similarly, for the RUBiS workload it was*

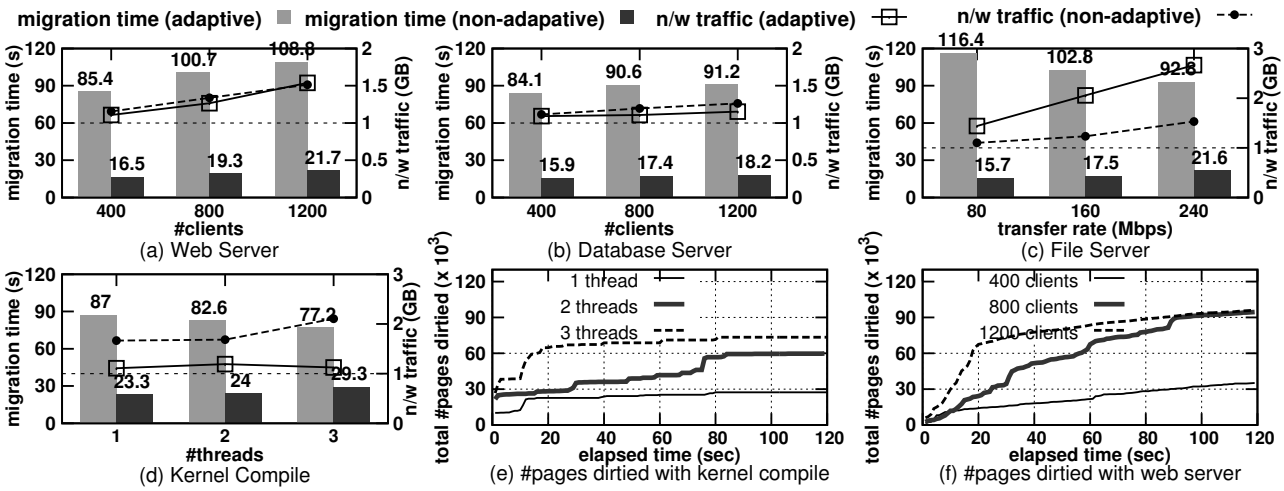


Figure 3: Effect of page dirtying rate on the performance of *non-adaptive* live migration

1.08 times lower. The reasons for this behavior are: (i) Though the time for each iteration was higher with adaptive live migration as compared to the corresponding iteration with non-adaptive live migration, the number of unique pages dirtied and the number of pages transferred during an iteration increased only marginally. Figure 2(e) shows the slowly increasing page dirty rate of the RUBiS and the kernel compile workloads. (ii) The *non-adaptive* live migration and *adaptive* live migration of these two workloads executed 29 iterations and 12 iterations, respectively. The adaptive live migration aggressively terminated iterative pre-copy due to the "page dirty rate is greater than a threshold" condition which is not used as a stopping condition for non-adaptive live migration. With increase in the number of iterations executed, the network traffic generated increased.

Takeaway: *Irrespective of the migration technique, the network traffic generated during migration depends on the page dirtying characteristics of the VM and the migration rate.*

Observation 3: *With non-adaptive live migration, migration time for RUBiS web server workload with 1 GB of memory and kernel compile workload with .5 GB of memory were* ***almost the same****. Similarly, the total network traffic generated during the migration of these two workloads were observed to be same.* The reason is that 27 out of 29 iterations spent less than 20 seconds for both of the workloads. The number of unique pages dirtied (refer Figure 2(e)) during those 27 iterations was higher with kernel compile workload as compared to RUBiS web-server workload.

Takeaway: *Considering only VM size to reduce the migration cost (i.e., migration time and resource utilization) is not adequate. It is necessary to account for page dirty rate of VM as well.* Though number of pages dirtied per second with kernel compile workload (25,000 pages/sec) was higher than file server workload (8,000 pages/sec), number of unique pages dirtied over time was higher with file server. Hence, only the number of unique pages dirtied over time should be considered.

Observation 4: *The downtime with adaptive live migration was observed to be* ***higher*** *as compared to non-adaptive live migration.* The reasons are as follows: (i) For an iteration #i, the number of unique pages dirtied with adaptive live migration was always higher than non-adaptive live migration (as mentioned in point (i) of observation 2(a) & 2(b)). (ii) Number of pages dirtied in iteration #i was always lesser than iteration #(i-1) for both adaptive live migration and non-adaptive live migration. This is because the iteration time decreased with increase in iteration number as the number of pages to be transferred per iteration decreased. As number of iterations with adaptive live migration was always lower as compared to non-adaptive live migration and also due to the reason (i) mentioned above, the number of pages to be transferred during stop-and-copy phase was higher.

Observation 5: *With increased VM size, the network overhead, i.e., the difference between total network traffic generated and size of the VM, increased.* This is plotted in Figure 2(d). As the size of the VM increased, the time taken by iteration #1 increased. As a result, the total number of unique pages dirtied during iteration #1 increased. Thus, the time taken by later iterations and the unique number of pages dirtied per iteration also increased. Further, with the file server workload, increase in VM size resulted in increase in size of the page caches. This resulted in higher number of pages dirtied per second. Figure 2(f) shows the increasing application page dirty rate of the file server workload for different VM sizes.

Increase in time per iteration and pages dirtied per iteration resulted in larger number of pages to be copied in the stop-and-copy phase. As a result, the downtime of applications increased with increase in VM size, as reported in Table 1.

Observation 6: *The migration rate of adaptive live migration decreased as the VM size increased except for the file server workload.* Table 1 presents the migration rate of adaptive live migration. This is because increase in VM size increased the duration of each iteration significantly but increment in the number of unique pages dirtied per iteration was less significant. As a result, the migration rate calculated using equation (1) and (2) for each iteration decreased with increase in the VM size. With the file server workload, the number of unique pages dirtied per iteration also increased significantly. As a result, the migration rate increased with increase in the virtual machine size of the file server workload.

Observation 7: *CPU utilization at the destination PM was observed to be* ***higher*** *than at the source PM.* For exam-

Table 2: Migration rate with adaptive live migration.

Benchmark	**migration rate** (Mbps)		
	load L_1	load L_2	load L_3
RUBiS web server	104	106	117
RUBiS db server	105	107	109
kernel compile	105	119	121
file server	102	165	237

ple, with non-adaptive live migration, with a migration rate of 640 Mbps, Dom0 CPU utilization at the source and destination PMs was observed to be 24% and 48%, respectively. The reasons for this behavior are: (i) At the source PM, the TCP segmentation offload feature was enabled to reduce the CPU overhead of TCP/IP. (ii) As decoding of packets at the destination PM tend to be more expensive than encoding at the source, CPU utilization was higher. (iii) In addition, memory write operation (i.e., storing memory pages of VM) at the destination PM is higher in cost compared to page read at the source.

Through these experiments, we observe that page dirty rate and migration rate play an important role in deciding the performance and cost of the live migration process. In the following sections, we study the impact of page dirty rate and resource availability (indirectly the migration rate) on the performance and cost of virtual machine migration.

4.2 Impact of Page Dirty Rate

In this section, we analyze and quantify the impact of the page dirty rate on the performance of the migration process. In order to generate different page dirty rates, we executed three workloads each with three different configurations (load levels). As expected, the number of unique pages dirtied increased as the load levels increased. All VMs were allocated memory of 1 GB each and the available network capacity for the migration process was 600 Mbps.

Figures 3(a), (b), (c), and (d) plot migration time and total network traffic over different load levels for both adaptive live migration and non-adaptive live migration. The number of unique pages dirtied over time at different load levels for kernel compile and RUBiS web server workloads are plotted in Figures 3(e), and 3(f), respectively.

Observation 8: *With adaptive live migration, only the migration time of the RUBiS workload increased as the load level increased, whereas the migration time of the file server and the kernel compile workloads decreased.* The reason is that, with the file server and kernel compile workloads, the migration rate increased significantly as the page dirty rate increased. Remember that page dirty rate is calculated using equation (2). With the RUBiS workload, the increase in migration rate was not significant. Table 2 presents the migration rate for adaptive live migration. This is because most of the iterations with adaptive live migration of all three workloads spent less than 10 seconds. The unique number of pages dirtied during that period was larger for the file server and kernel compile workloads (Figures 3(e) and 3(f))) compared to the RUBiS workload.

Observation 9: *The total volume of network traffic increased as the load level increased, except with adaptive live migration of VM hosting kernel compile workload.* This is because the total number of pages dirtied during any particular iteration increased as the page dirty rate increased. As a result, the number of pages transmitted during any particular iteration also increased. However, when three threads were used in the kernel compile workload, the total network traffic generated was lower (due to fewer iterations) than the two-threaded kernel compile workload. As the page dirty rate of three threads kernel compile workload was higher, the iterative pre-copy phase terminated aggressively with fewer iterations due to the "page dirty rate of previous iteration is greater than a threshold and #pages sent in current iteration is greater than previous iteration" condition as compared to the two threads kernel compile workload.

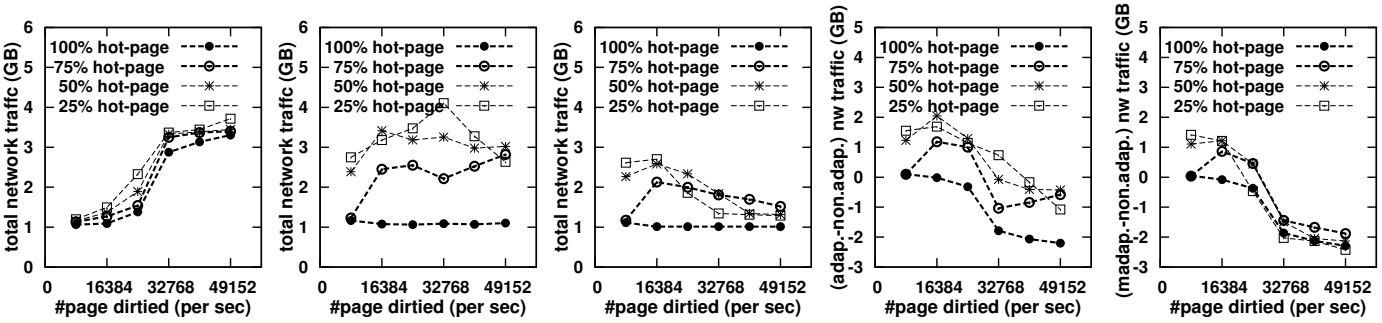
In order to understand the impact of page dirty rate further on network traffic, we performed migration of a VM hosting a micro-benchmark, where we can configure the number of pages dirtied per second (n), and a *hot-page* factor (h), which denotes the percentage of "n" pages that are dirtied every second (i.e., same pages). The remaining $100 - h$ percentage of "n" pages are selected randomly. The size of the VM was 1 GB and the available network capacity for the migration process was 1 Gbps.

Takeaway: *Allocation of network resources to non-adaptive live migration process should be much higher than the page dirty rate of the VM that is being migrated to reduce total network traffic. When the number of pages dirtied per second is greater than the network capacity, it is better to reduce the threshold value in iterative pre-copy condition 2 to reduce migration cost.*

The reasons are as follows: Figure 4(a) plots total network traffic generated with non-adaptive live migration over different number of pages dirtied per second. When the number of pages dirtied per second was less than the network capacity (i.e., < 32768 pages), the iterative pre-copy phase terminated due to the "number of pages dirtied in current iteration is less than a threshold" condition. This is because the rate at which pages dirtied during every iteration was less than the migration rate. As a result, the number of unique pages dirtied decreased in each iteration, as the number of iteration increased. When the number of pages dirtied per second was greater than the network capacity, the iterative pre-copy phase terminated due to the "total network traffic generated is greater than a threshold" condition. This is because the rate at which pages dirtied during every iteration was greater than the page transfer rate. With a decrease in the hot-page factor (h), the total network traffic increased as more pages were dirtied.

Takeaway: *To reduce the total network traffic generated with adaptive live migration, it is better to remove part (ii) from iterative pre-copy condition 4. This reduces the total resource utilization significantly but with little increase in the downtime.*

The reasons are as follows: Figure 4(b) plots the total network traffic generated with adaptive live migration over different number of pages dirtied per second. With a hot-page factor of 100%, the total network traffic generated is almost same for different number of pages dirtied per second. This is because the iterative pre-copy phase terminated at iteration #3 due to the "(i) page dirty rate of the last iteration is greater than a threshold and (ii) #pages sent in current iteration is greater than the last iteration" condition (the dirty rate is calculated using equation (2)). When the hot-page factor was less than 100% (implying increase in the page dirty rate), we expected the iterative pre-copy to terminate at iteration #3 or earlier due to the above mentioned iterative pre-copy condition. However, a larger number of



(a) Non-adaptive live migration (b) Adaptive live migration (c) madap live migration (d)(adap-non.adap) n/w traffic (e)(madap-non.adap) nw traffic

Figure 4: Impact of page dirtying rate on the migration generated network traffic

Table 3: Downtime with adaptive and *modified-adaptive* live migration (madap) techniques.

#pages dirtied n	**adaptive**(secs)		**madap**(secs)	
	h=25	h=75	h=25	h=75
16384	0.22	0.44	0.5	0.67
32768	1.2	3.4	1.52	5.09
49152	2.2	5.3	3.2	5.47

iterations were required to meet "# pages sent in current iteration is greater than the last iteration" condition. As a result, the network traffic was higher.

To speed up termination of the migration process and hence reduce the network overhead, we eliminate the stopping condition #(ii) and study its impact on the performance and cost of migration. We call this updated technique the *modified-adaptive* (madap) migration technique. Figure 4(c) plots the total network traffic generated with *modified-adaptive* live migration over different number of pages dirtied every second. The total network traffic observed was quite low compared to adaptive live migration. However, the downtime of the VM increased and is presented in Table 3. The reason is that, the number of iterative pre-copy iterations executed with *modified-adaptive* live migration was less than adaptive live migration. However, with the four workloads listed in Section 2, we did not observe a significant difference (< 100 MB) in network traffic with adaptive live migration and *modified-adaptive* live migration and we observed that the downtime increased on average by 500 milliseconds.

Observation 10: *When the number of pages dirtied per second is greater than the network capacity, the total network traffic generated with adaptive live migration and* modified-adaptive *live migration was less than non-adaptive live migration.* This behavior occurs because adaptive live migration and *modified-adaptive* live migration executed fewer iterations when the number of pages dirtied per second was greater than the network capacity. The iteration terminated due to iterative pre-copy termination condition #4 (refer Section 2). Figures 4(d) and 4(e) plot the difference in network traffic generated by adaptive and modified-adaptive live migration (madap) with respect to non-adaptive live migration.

Takeaway: *If we enable iterative pre-copy termination condition #4 for non-adaptive live migration, the total network traffic generated with non-adaptive will become less than adaptive.* We present the impact of this modification on non-adaptive live migration in Section 5.1.

In these experiments, we do not restrict the resources to be utilized by the migration process. However, dynamic resource management techniques may constrain the availability of resources for migration—finite resources are dynamically distributed to balance performance and resources utilized over all VMs on a host. In the following section, we study the performance of the migration processes with resource constraints.

4.3 Impact of Resource Availability

In this section, we quantify the impact of resource availability, (i) network capacity, and (ii) Dom0 CPU, on the performance of VM migration process. First, we performed adaptive and non-adaptive live migration of VMs by restricting the network bandwidth utilization. To restrict the bandwidth utilization of a migration process, we introduce a new variable named *max_bw_limit* in the source code. As a result, the migration process's transfer rate does not exceed *max_bw_limit* Mbps. To achieve this, we re-used some lines of code from the adaptive live migration. The RUBiS workload was configured with 1,200 clients and all VMs were allocated 1 GB of memory.

Figures 5(a) and 5(b) plot the migration time of the four workloads over different available network capacities for non-adaptive live migration and adaptive live migration, respectively. Increase in the available network capacity results in a non-linear decrease in the migration time. The time to migrate the file server workload was higher than the other workloads, as the unique number of pages dirtied over a given period was higher. Also (referring to Figure 2(e)), migration time is directly proportional to the unique number of pages dirtied over a period of time. The downtime of the workloads are presented in Table 4 for different available network capacities.The downtime decreased as the available network capacity increased.

Takeaway: *Allocating low resource level for the VM migration process results in disproportionately higher migration time and downtime. Hence, we should not allocate too low resources for migration process.*

Observation 11: *Adaptive live migration under-utilizes available resources, which results in higher migration times.* The reasons are as follows: Migration time decreased till the available network capacity reached the page dirty rate, and then it became constant. This is because adaptive live migration fixed the migration rate to the rate at which pages were dirtied. The page dirty rate of the workloads was lower than the available network capacity.

Observation 12: *With increase in the available network capacity, the network traffic for migration either decreased non-linearly or remained more or less constant.* Figures 5(d) and 5(e) plot the total network traffic generated during migration over different available network capacities for the file

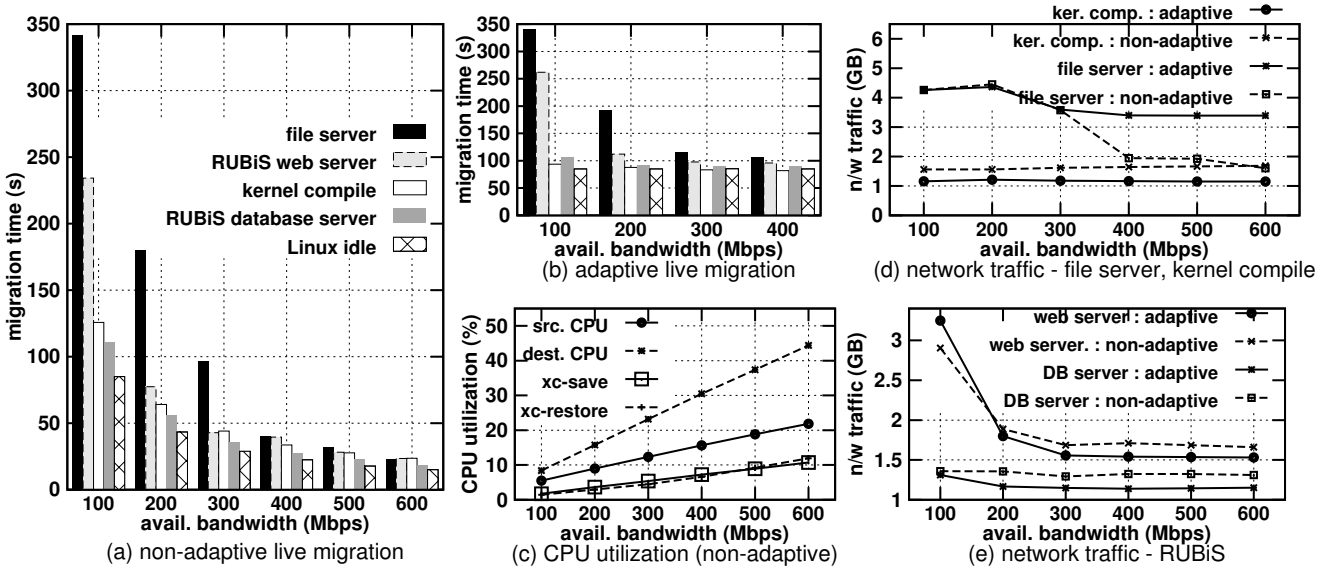


Figure 5: Impact of available network capacity on migration time, network traffic and Dom0 CPU utilization

Table 4: Impact of available network capacity on downtime of the application

	migration rate (Mbps)											
	100	**200**	**300**	**400**	**500**	**600**	**100**	**200**	**300**	**400**	**500**	**600**
Benchmark	**downtime** : adaptive (sec)						**downtime** : non-adaptive (sec)					
RUBiS web server	15.1	2.54	2.2	0.65	0.46	0.36	8.78	0.63	0.41	0.61	0.32	0.3
RUBiS database server	2	0.57	0.4	0.26	0.22	0.17	1.38	0.63	0.43	0.33	0.29	0.25
kernel compile	9.6	4.2	2.96	1.86	1.45	1.21	7.19	3.41	1.73	1.32	1	.778
file server	74.6	33.5	6.9	3	2.41	2.01	74.6	36.7	6.62	0.12	0.09	0.05

server, kernel compile, and RUBiS workloads, respectively. This is because with a decrease in the migration rate, the time spent on each iteration increased. As a result, the total number of unique pages dirtied per iteration increased. Hence, the network traffic generated also increased. This is because higher network traffic may result in higher overall performance degradation. However, with the kernel compile workload, the total network traffic increased. The reasons are as follows: As the migration rate increased, efficiency of the skip list decreased. Figure 6 plots the percentage of dirtied pages skipped during the migration of the kernel compile and RUBiS database server workloads over different available network capacities for non-adaptive live migration. As the available network capacity increased, total number of pages dirtied decreased for both the kernel compile and RUBiS database server workloads (for other workloads as well) because of decrease in time taken by each iteration. Also, the percentage of the dirty pages skipped decreased as the migration rate increased. The reason is that, with increase in the migration rate, the batch-transfer time (refer Section 2.1) decreased. As a result, only a few pages were added to the **skip list** and more pages were dirtied after their transmission. When the migration rate was low, more pages were added to the **skip list** and fewer pages were dirtied after their transmission due to the high batch-transfer time. However, the total number of pages dirtied increased with decrease in migration rate. With the kernel compile workload, the percentage of the dirty pages skipped decreased significantly, whereas with RUBiS database server workload, the decrease in percentage was not significant. Also, the total number of pages dirtied was higher with the kernel compile workload (implies decrease in the skip-list effectiveness with increase in migration rate).. As a result, the

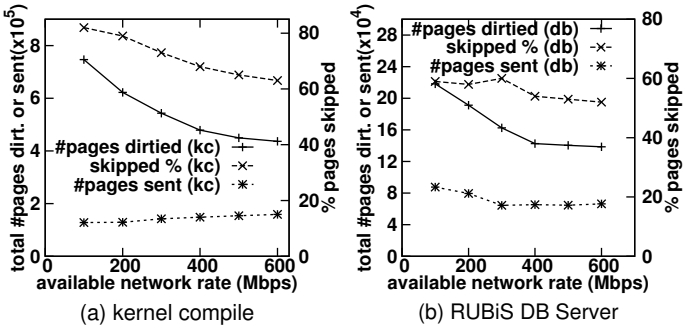


Figure 6: Efficiency of skip list over different available network capacity for non-adaptive live migration

amount of network traffic generated increased with increase in the available network capacity for non-adaptive live migration of kernel compile workload.

Takeaway: *Non-adaptive live migration with finite resources allocated to the migration process provides a tuning knob to adjust the performance and cost of migration. With higher resources allocated for migration, migration time, down time and network traffic decrease monotonically. Quantum of resources allocated can be determined based on minimizing the impact of migration on application performance of other co-located VMs.*

Skip-list effectiveness: To understand the effectiveness of the skip list, we migrated the workloads by disabling the skip list functionality (in source code) for the adaptive and non-adaptive live migration. Table 5 presents the performance of non-adaptive live migration with and without the skip list functionality for the RUBiS web server and the kernel compile workloads. Without the skip list functionality,

migration time, network traffic and downtime were higher (almost double) as compared to the migration with the skip list functionality.

Observation 13: *As the available network capacity increased, the migration rate increased linearly and so did the CPU utilization at the source and destination physical machines.* The process *xc_save* that implements the live migration algorithm transfers memory pages from the source PM to process *xc_restore* which receives memory pages and starts the virtual machine at the destination PM. The CPU utilization (excluding network activity) of these two processes is reported in Figure 5(c). The CPU utilization of *xc_save* and *xc_restore* were similar and increased linearly.

Further, we migrated virtual machines by restricting the Dom0 (the management domain in Xen) CPU utilization levels of *xc_save* and *xc_restore* processes with the help of the *cpulimit* tool. We performed three sets of experiments by restricting Dom0 CPU utilization level of (i) the *xc_save* process, (ii) the *xc_restore* process, and (iii) both *xc_save* and *xc_restore* processes. The CPU levels allocated to *xc_save* and *xc_restore* were the same as the values plotted in Figure 5(c). As the Dom0 CPU availability decreased, the migration rate decreased linearly and the migration time and network traffic were similar to the experiment which restricted available network capacity. We observed that, the migration rate was proportional to the minimum of (i) available Dom0 CPU for *xc_save* process and (ii) available Dom0 CPU for *xc_restore* process.

As a result, each allocated CPU level has a corresponding network rate which can be supported, and either of which can be used to allocate network resources to the migration process.

5. IMPROVED LIVE MIGRATION

The two main qualitative inferences from the empirical evaluation (presented in the previous section) are: (i) Cost of migration—migration time, downtime, network traffic for migration, is lower for non-adaptive live migration as compared to adaptive live migration. The exception to this being workloads with high page dirty rate. (ii) The adaptive migration technique seldom uses all the available resources for migration, whereas the non-adaptive technique does not account for "available" resources.

We propose to combine the benefits of both the adaptive and non-adaptive techniques by simultaneously being sensitive to both page dirty rate of applications and available resources for migration. Towards this we propose the **Improved Live Migration** (ILM) technique. The spirit of the Improved Live Migration technique is to use non-adaptive live migration but with finite resources for migration (to reduce interference towards other applications) and to be sensitive to an application's page dirty behavior (to reduce the cost of migration). We propose three main improvements as part of ILM:

(i) Stopping criteria tuned to page dirty rate: The "page-dirty-rate-greater-than-threshold" condition in the adaptive migration technique aggressively terminates the migration process if the dirty rates are very high (see results in Section 4.2). The exact condition being, *page dirty rate of previous iteration is greater than a threshold and number of pages sent in current iteration is greater than previous iteration.* Since no such condition exists in the non-adaptive technique we often observed that the procedure terminated only after higher number of iterations (see results in Section 4). The first component of ILM is to add this condition as a stopping criteria to the non-adaptive live migration procedure.

(ii) Finite resources for migration: The non-adaptive migration technique attempts to use as many resources as possible to reduce the migration time and downtime metrics. This is oblivious to the fact that other applications may need to pay the "cost" during migration. To balance the benefits of improved migration performance and minimize the impact on other applications, we allot deterministic resources to the non-adaptive migration procedure.

The two main resources consumed by the migration procedure are network bandwidth and Dom0 CPU (for the Xen architecture). Restricting usage of these resources will minimize interference with other applications. For the purpose of this work we assume that the "available" levels of resources for migration are known. This level of resource availability depends on the left-over (unallocated) resources after accounting for resources needed to meet the SLA requirements of other VMs. We assume that this estimation happens through a higher-level migration decision process and is orthogonal to the migration procedure optimization.

Further, as seen in Section 4.3, the network utilization of the migration process and the Dom0 CPU utilization are directly (and linearly) related. Hence, as the second component of ILM, we propose to restrict the maximum network bandwidth available to then migration process.

(iii) Only transfer required pages: Conventional migration procedures transfer free-pages and page-cache pages as part of the migration process. Transfer of free-pages and page-cache pages is not required for the correctness of the migration process—free-pages have no relevant content and page-cache misses result in additional disk reads. The usefulness of the page-cache depends on the I/O access patterns and the associated hit rates. As part of ILM, we trade-off the potential I/O access benefit to reduce in the migration cost. Reduction in migration cost is more deterministic, as it is based on reducing the number of pages transferred as compared to the utility of the page cache pages. The third component of ILM is to ignore free-pages and page-cache pages from the set of pages to be migrated to the destination machine.

In the next section, we evaluate the performance of the Improved Live Migration technique (ILM) and compare it with the conventional adaptive and non-adaptive live migration techniques.

5.1 ILM Evaluation

To quantify the improvement in *performance* and *cost* of live migration techniques resulting from the proposed modifications, we performed migrations of a VM (of size 1 GB) hosting four different workloads with adaptive live migration, non-adaptive live, Improved Live Migration technique without dropping free-pages and page-cache pages (we call this ILM-fc) and Improved Live Migration technique (ILM)– with all three modifications.

Table 6 presents the network traffic (NT), migration time (MT) and Downtime (DT) with conventional live migration techniques and Improved Live Migration techniques. The network traffic generated with ILM-fc was 1 to 2.24 times lower as compared to *adaptive* live migration and 1.21 to 1.41 times lower as compared to *non-adaptive* live migration.

Table 5: Effectiveness of skip list functionality.

Benchmark	with skip-list			without skip-list		
	migration time (s)	network traffic (GB)	downtime (ms)	migration time (s)	network traffic (GB)	downtime (ms)
RUBiS web server	28.1	1.68	325	40.3	2.43	1257
kernel compile	27.7	1.67	1000	53.8	3.26	1707

Table 6: Available Network Capacity = 400 Mbps

Benchmark	adaptive			non-adaptive			ILM-fc			ILM		
	NT (GB)	MT (sec)	DT (sec)	NT (GB)	MT (sec)	DT (sec)	NT (GB)	MT (sec)	DT (sec)	NT (GB)	MT (sec)	DT (sec)
RUBiS web server	1.46	100.1	0.69	1.71	39.5	0.61	1.34	27.8	0.65	1.25	26	0.98
RUBiS db server	1.13	89.2	0.39	1.32	27.5	0.33	1.09	23	0.35	0.14	6.5	0.2
kernel compile	1.16	81.9	1.86	1.64	33.62	1.32	1.16	24.48	2.2	0.21	4.7	1.38
file server	3.39	105.8	3	1.94	40	0.12	1.51	35.8	0.2	0.23	5.2	0.24

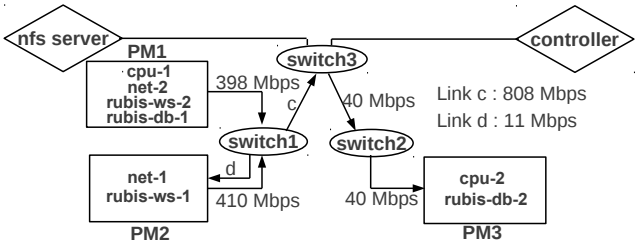


Figure 7: Experimentation setup used for measuring performance degradation.

The reason is that, ILM-fc and adaptive live migration executed more or less the same number of iterations and lower than non-adaptive live migration. We already observed that the higher the number of iterations, the higher the network traffic and shorted the downtime. As a result, downtime with ILM-fc was 1.06 to 1.66 times higher as compared to *non-adaptive* live migration and 1.06 to 15 times lower as compared to *adaptive* live migration.

The network traffic generated with ILM was 1.16 to 14 times lower as compared to adaptive live migration and 1.3 to 8.4 times lower as compared to non-adaptive live migration. The RUBiS web server workload did not have more free-pages and page-cache pages. As a result, the improvement in performance and cost of migration was low. Whereas, in case of other workloads, the memory was mainly occupied with page-cache pages.

In the following section, we study the impact of migration with and without resource restrictions on an application's performance

6. IMPACT OF MIGRATION ON APPLICATION PERFORMANCE

To quantify the impact of migration techniques on the performance of applications running in the VMs, we performed experiments with the setup shown in Figure 7. We used three machines (PM1 to PM3, whose configurations were same as the one reported in Section 3). The set of workloads collectively hosted on the three PMs are: two web servers (rubis-ws-1 and rubis-ws-2), two RUBis database servers (rubis-db-1 and rubis-db-2), two file server workloads (net-1 and net-2) and two CPU intensice PHP scripts (cpu-1 and cpu-2). RUBiS-1 (rubis-ws-1+rubis-db-1) and RUBiS-2 (rubis-ws-2+rubis-db-2) represent the two RUBiS applications. An initial mapping of workloads (running in VMs) to PMs is show in Figure 7. The average network utilization of each link is also marked in Figure 7.

We allocated resources for each workload to handle a specific load level. We used Apache JMeter to generate requests for each workload. The throughput (τ) of each workload is presented in Table 7 (refer column $\tau_{without-migration}$). The throughput of the VM hosting CPU intensive PHP script (cpu-1) was 8 requests/sec. We introduce an overload situation in physical machine PM1 by increasing the request arrival rate of cpu-1 from 8 requests/sec to 23 requests/sec, which requires an additional 50% of CPU. However, there is not enough CPU resource available at physical machine PM1 to allocate for the cpu-1. Hence, this overload situation can be solved only through VM migration. In order to mitigate the hot-spot, we migrated rubis-ws-2 (allocated with 100% of CPU) of size 1 GB from PM1 to PM3 (we assume that we cannot migrate cpu-1) to free CPU resources. The average network utilization of the link connecting PM1 to PM3 was 808 Mbps (it varied from 700 Mbps to 940 Mbps). Hence, network link was the bottleneck during VM migration and observations made below are considering this bottleneck scenario. The available Dom0 CPU at PM1 and PM3 were adequate to migrate a VM at a rate of 1 Gbps. We measured the performance degradation in terms of throughput (reqs/sec) of the workloads with adaptive live migration, non-adaptive live migration, and ILM-fc with four different migration rates. Table 7 presents migration time (MT), downtime (DT), migration rate (MR), network traffic generated (NT), throughput of each workload without migration ($\tau_{without-migration}$), and throughput drop (τ_{drop}) during migration.

Observation 14: *The throughput drop per second (τ_{drop}) for RUBiS-1, RUBiS-2, net-1 and net-2 workloads with non-adaptive live migration was observed to be 1.6 to 2.4 times higher than with adaptive live migration. However, the total #requests dropped for each workload was lower with non-adaptive live migration* and are presented in Table 8. This is because the total amount of resources utilized was higher with adaptive live migration. The network traffic generated with adaptive live migration was 1.23 times higher than non-adaptive live migration. Also, the downtime of adaptive live migration was three orders of magnitude higher than non-adaptive live migration.

Observation 15: *As the migration rate of the ILM-fc technique increased, the total number of requests dropped during migration decreased.* This is because with increase in migration rate, the total network traffic generated decreased (refer Section 4.3). As a result, the impact of migration on other applications also decreased.

As the migration rate of ILM-fc decreased from 400 Mbps to 100 Mbps, the percentage of the throughput drop was de-

Table 7: Impact of migration on the performance of application

Technique	MT sec	DT msec	MR Mbps	NT MB	$\tau_{without-migration}$ req/s				τ_{drop} req/s			
					rubis-1	rubis-2	net-1	net-2	rubis-1	rubis-2	net-1	net-2
adaptive	79	1,251	135	1,293	412	591	3.25	2.7	117	223	0	0.35
non-adaptive	26	3	350	1,051	448	545			287	363	0.07	0.81
ILM-fc (100 Mbps)	96	2,972	108	1,405	410	596			53	262	0	0.12
ILM-fc (200 Mbps)	52	2,632	205	1,283	419	592			157	273	0	0.46
ILM-fc (300 Mbps)	34	2,700	298	1,205	433	563			253	293	0.03	0.76
ILM-fc (400 Mbps)	26	4	351	1041	448	545			288	360	0.07	0.82

Table 8: Total number of requests drop per workload during migration

Technique	rubis-1 (#reqs)	rubis-2 (#reqs)	net-1 (#reqs)	net-2 (#reqs)	cpu-1 (#reqs)	total #reqs
adaptive	9,243	17,617	0	27	1,185	28,072
non-adaptive	7,462	9,438	2	21	390	17,313
ILM-fc (100 Mbps)	5,088	25,152	0	12	1,440	31,692
ILM-fc (200 Mbps)	8,164	14,196	0	22	780	23,162
ILM-fc (300 Mbps)	8,602	9,962	1	26	510	19,101
ILM-fc (400 Mbps)	7,488	9,360	2	22	390	17,262

creased disproportionately from 66% to 43% for RUBiS-2 workload. Whereas with RUBiS-1 workload, it decreased from 64% to 12%. This is because of the impact of enabling the shadow page table and retrieving the dirty bitmap after every batch transfer to construct skip list during migration. These two operations increased the CPU utilization of rubis-ws-2 during migration, making the CPU a bottleneck even at lower migration rate (i.e., 100 Mbps). The CPU utilization of rubis-ws-2 was always between 50% and 70% before the migration. However, during the migration, the CPU utilization was often at 100%. As a result, even with lower migration rate, the throughput drop was higher.

Takeaway: *In a network bottleneck scenario, if we migrate VMs at low network rate, the throughput drop (requests per seconds) of applications will be low but the total #requests dropped (during migration) will be high.*

In this section, we studied the impact of migration on the performance of web applications by making network link as bottleneck during migration. As a part of future work, we would like to quantify the performance degradation by making Dom0 CPU as the bottleneck. Also, we need to study the impact of dropping page cache pages on the application performance.

7. SUMMARY

• We conducted a comprehensive empirical study of migration techniques along the axes of migration time, downtime, and network traffic generated during migration. We showed that non-adaptive live migration performs better (implies lower migration time, downtime and network traffic) than adaptive live migration in most of the cases.

• We showed that only considering VM size as a parameter to reduce migration cost is inadequate. The page dirty rate of the VM along with the migration rate (indirectly governed by the available network capacity) are vital parameters deciding the cost of migration.

• Adaptive live migration under utilizes available CPU and network resources and cannot minimize the migration cost. We proposed ILM, an improved migration technique, which allocated finite resources to the non-adaptive live migration technique, and delivers the least migration cost as compared to the other techniques.

• Resource allocation levels for the migration process affects the throughput drop (requests per second) and the total #requests dropped (during migration) in opposite proportions. Allocation levels should follow the user's requirements in such cases.

8. RELATED WORK

The parameters of migration and related performance metrics considered in studies so far are summarized in Table 9.

Huang et al. [8] studied the performance of *non-adaptive* live migration with both the Xen [4] and KVM [12] virtualization technologies. Their study was restricted to two workloads—a Java virtual machine benchmark and a CPU-intensive benchmark. Xen and KVM implementations of *non-adaptive* live migration were compared based on downtime, migration time, total volume of network generated, and the performance degradation of the benchmark in the VM which was migrated. This work did not consider the different system parameters that can affect migration performance.

Akoush et al. [3] presented a simulation model to predict migration time and downtime of *non-adaptive* live migration. The migration technique was simulated using the page dirty trace collected from Xen shadow page tables. The model assumed that the recorded set of pages dirtied will be approximately representative of the application and will repeat under similar environments. Experiments were performed with 100 Mbps, 1 Gbps and 10 Gbps link capacity, and different VM sizes. However, the impact of available resources on the performance of non-adaptive live migration was not studied.

Hai et al. [9] evaluated the performance of *non-adaptive* live migration against page dirty rate using a micro-benchmark. Only downtime of the VM was measured and discussed. They also performed migration of idle VMs by reserving Dom0 CPU for migration and reported migration time. However, the impact of resource availability on downtime, total network traffic and CPU utilization were not discussed. Also, the impact of page dirty rate of the workloads was not explicitly considered or discussed.

Clark et al. [10] studied the performance of *adaptive* live migration for various workloads. However, only the page dirty rate was considered as a parameter which affected the performance. How each iteration's migration rate and network traffic in each iteration varied were shown for different

Table 9: Parameters and Performance metrics accounted in current literature

	Huang et. al. [8]	Akoush et. al. [3]	Liu et. al. [13]	Clark et. al. [10]	Hai et. al. [9]	Kejiang et. al. [17]	Our work
Live Migration Algorithms Evaluated							
Non-adaptive	√	√			√		√
Adaptive			√	√		√	√
Parameters Considered							
VM Size						√	√
Page dirtying rate		√	√	√	√		√
Network bandwidth				√			√
Dom0 CPU						√	√
Performance Metrics Accounted							
Migration time, downtime	√	√	√	√	√	√	√
Network traffic	√		√		√		√
CPU utilization							√

workloads. The effect of migration on web server transmission rates was also studied. However, this work did not study the impact of page dirty rate on the CPU resources required for migration and performance of non-adaptive migration.

Liu et al. [13] presented mathematical models and techniques to predict the energy consumption, migration time, downtime, and total volume of network traffic for an *adaptive* live migration. The model used resource-usage and migration-related parameters from past VM migrations to predict the migration performance parameters. However, this work covered only the page dirty rate parameter while evaluating the migration technique.

Ye et al. [17] evaluated the performance of adaptive live migration (only migration time and downtime) against available Dom0 CPU level at the source and destination machines. As adaptive live migration utilizes lower resources, they could not find any significant impact of available Dom0 CPU level. Migration rate, CPU utilization of migration process and page dirty rate of the workloads were not reported.

As can be seen, no prior work benchmarks the performance of VM migration comprehensively— against all parameters that affect migration and measure all performance metrics. Our work fills this gap and comprehensively evaluates the impact of all systems parameters on migration performance. Also, this is the first study to compare both *non-adaptive* live migration and *adaptive* live migration techniques.

9. CONCLUSION AND FUTURE WORK

In this paper, we presented a comprehensive empirical study of the Xen live migration techniques—adaptive live migration and non-adaptive live migration. We showed how the four parameters—VM size, Dom0 CPU, network capacity and application page dirty rate affect the performance of live migration in terms of migration time and downtime, and migration cost in terms of total CPU utilization and total network traffic. We also compared adaptive live migration and non-adaptive live migration, identified shortcomings and proposed three improvements. For our experimental setup and workloads, the Improved Live Migration technique (ILM) reduced network traffic for migration reduced by 14-93% and migration time reduced by 34-87% as compared to the vanilla live migration techniques.

As a part of future work, we intend to create a mathematical model to predict migration performance. The aim of model would be to predict migration time, downtime, total network traffic and CPU utilization by taking VM size, resource availability and application page dirty rate as inputs. Such prior predictions can be utilized for effective migration decisions for purposes of load balancing, consolidation and hot-spot mitigation within a virtualized environment.

10. REFERENCES

[1] http://linux.die.net/man/1/xentop.

[2] RUBiS: Rice University Bidding System. http://rubis.ow2.org/.

[3] S. Akoush, R. Sohan, A. Rice, A. W. Moore, and A. Hopper. Predicting the Performance of Virtual Machine Migration. In MASCOTS, Aug. 2010.

[4] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the Art of Virtualization. In SOSP, Oct. 2003.

[5] T. Das, P. Padala, V. N. Padmanabhan, R. Ramjee, and K. G. Shin. LiteGreen: Saving Energy in Networked Desktops using Virtualization. In USENIX ATC, June 2010.

[6] E. L. Haletky. *VMware ESX Server in the Enterprise: Planning and Securing Virtualization Servers*. 1 edition.

[7] F. Hermenier, X. Lorca, J.-M. Menaud, G. Muller, and J. Lawall. Entropy: a Consolidation Manager for Clusters. In VEE, March 2009.

[8] D. Huang, D. Ye, Q. He, J. Chen, and K. Ye. Virt-LM: a Benchmark for Live Migration of Virtual Machine. In ICPE, March 2011.

[9] H. Jin, L. Deng, S. Wu, X. Shi, and X. Pan. Live Virtual Machine Migration with Adaptive, Memory Compression. In IEEE CLUSTER, Aug. 2009.

[10] C. C. Keir, C. Clark, K. Fraser, S. H, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live Migration of Virtual Machines. In NSDI, May 2005.

[11] G. Khanna, K. Beaty, G. Kar, and A. Kochut. Application Performance Management in Virtualized Server Environments. IEEE/IFIP NOMS, April 2006.

[12] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori. KVM: The Linux Virtual Machine Monitor. In OLS, June 2007.

[13] H. Liu, C.-Z. Xu, H. Jin, J. Gong, and X. Liao. Performance and Energy Modeling for Live Migration of Virtual Machines. In HPDC, June 2011.

[14] A. Verma, P. Ahuja, and A. Neogi. pMapper: Power and Migration Cost Aware Application Placement in Virtualized Systems. In Middleware, Dec. 2008.

[15] T. Wood, P. Shenoy, A. Venkataramani, and M. Yousif. Black-box and Gray-box Strategies for Virtual Machine Migration. In NSDI, April 2007.

[16] J. Xu and J. Fortes. A Multi-objective Approach to Virtual Machine Management in Datacenters. In ICAC, June 2011.

[17] K. Ye, X. Jiang, D. Huang, J. Chen, and B. Wang. Live Migration of Multiple Virtual Machines with Resource Reservation in Cloud Computing Environments. In IEEE CLOUD, July 2011.