Doctoral Dissertation
博士論文

Department of Creative Informatics
Graduate School of Information Science and Technology
The University of Tokyo

Reducing Energy Consumption of Data Centers by Improving Virtual Machine Live Migration

(仮想マシンライブマイグレーションの高効率化による
データセンタ消費電力の削減)

Soramichi Akiyama
穐山 空道

Supervisor: Professor Shinichi Honiden

December 2014

ABSTRACT

Cloud computing has become a common computing paradigm both for individual and enterprise uses. As the use of cloud computing emerges, data centers for hosting clouds are becoming huge. In particular, large amount of energy consumption by data centers is one of the biggest issues in this cloud age. Energy consumption of data centers have been evolving by more than 10% per year, and it counts for non-negligible part of the whole energy consumption of countries such as the US.

In order to reduce energy consumption of data centers, the goal of this thesis is to achieve green data centers with aggressive VM relocation at runtime using efficient live migration. There are two problems that must be tackled to achieve this goal:

1. Live migration efficiency is not fully optimized yet. More concretely, a part of memory transfer mechanism of live migration costs much despite of the efforts by many live migration optimization researches.
2. Integrated evaluation of aggressive VM relocation at runtime and live migration is missing in the research field. This lack causes a problem that actual energy saving given by aggressive VM relocation cannot be easily estimated without using real implementation and real servers.

To tackle these problems, this thesis has three contributions for reducing energy consumption of data centers by improving efficiency of live migration.

1. First, this thesis discusses the requirements and the technical challenge toward goal that are missing in existing studies. They include efficient live migration and integrated evaluation of live migration and aggressive VM relocation and this thesis tackles both of them.
2. Second, this thesis proposes efficient live migration methods with two novel ideas: memory reusing and parallel transfer of page cache.

   The first idea is based on an observation that in an aggressive VM relocation system VMs migrate many times among a certain set of PMs such as ones in the same rack. In this situation, VMs can migrate "back" to a host on which it was executed before. MiyakoDori, the proposed system, leverages this fact for efficient live migration by caching the memory image of the target VM on a migration on the source PM, and reuse the memory image when the VM migrates back in the

future.

The latter idea is based on an observation that IO-intensive VMs have large amount of page cache (also known as file cache or file buffer) in the memory. In addition, modern data centers are equipped with a dedicated storage area network (SAN) along with a general purpose network (GPN). Page cache teleportation, the proposed system, utilizes the SAN and the GPN in parallel for efficient live migration of IO-intensive VMs by transferring page cache from storage PMs to the destination PM of a migration (instead of transferring it from the source PM) via the SAN.

3. Third, this thesis evaluates MiyakoDori and Page cache teleportation in integrated simulations with aggressive VM relocation algorithms. Although there are many existing researches on live migration and VM relocation, they are done separately and how much actual energy an efficient migration method can save is still not revealed. As a result of the simulation, this thesis figures out that our methods can reduce energy consumption of data centers up to a few percent, which is equivalent with a large amount of electricity usage and energy cost because the whole data center energy consumption is extremely huge.

概要

クラウドコンピューティングは現在では企業，個人のいずれにおいても重要な計算パラダイムの一つに成長した．クラウドコンピューティングの隆盛に伴い，クラウドをホストするデータセンタも巨大なものとなり，特にその消費電力が重大な課題となっている．データセンタの消費電力量は毎年 10 パーセント以上増加し，アメリカ等では一国の消費電力量全体のうち無視できない割合を占める．

増大する消費電力に対し，本博士論文では，効率的な仮想マシンライブマイグレーションと積極的な仮想マシン再配置による低消費電力なデータセンタを目指す．この目標を達成するために解決すべき技術的課題は以下である．

1. ライブマイグレーション技術を様々な既存研究からさらに効率化する必要がある．具体的には，ライブマイグレーションにかかるメモリ転送のコストの一部が未だ高いまま残されている．
2. 積極的な仮想マシン再配置とライブマイグレーションの統合評価が必要である．既存研究で統合評価がなされていないことにより，ある仮想マシン再配置アルゴリズムとライブマイグレーション手法の組み合わせによるデータセンタ全体の消費電力削減量が実環境でのテストなしには見積りできない．

これらの技術的課題に対処しデータセンタの消費電力を削減するため，本博士論文は以下の三点の貢献をする．

1. 第一に，本博士論文の目標を達成するために必要な要素を議論する．必要な要素のうち既存研究で達成されていないものは効率的なライブマイグレーションとその積極的仮想マシン再配置との統合評価であり，以下ではこれらに対処する．
2. 第二に，本博士論文は効率的なライブマイグレーションを実現する手法として，メモリ再利用とページキャッシュの並列転送を提案する．

   メモリ再利用手法は，積極的仮想マシン再配置を利用するデータセンタでは仮想マシンは少数のグループの物理マシン（例えば一つのラックに属する物理マシン）の間を何度も移動するという着想に基づく．この状況下では，ある仮想マシンはその仮想マシンが一度実行されたホストに「戻る」ことがある．そこで本論文では，仮想マシンが一度実行されたホストに戻る際に，以前の実行時のメモリイメージを再利用して更新されたメモリページのみ転送するシステム「都鳥」を提案・実装する．

   次にページキャッシュの並列転送手法は，ファイル入出力を多く行う仮想マシンではメモリの多くをページキャッシュ（ファイルキャッシュ，ファイルバッファとも呼ばれ

る）が占めるという着想に基づく．また近年のデータセンタには通常のネットワークの他にストレージ専用のネットワーク（SAN）を持つことが多い．そこで本論文では，ページキャッシュの一部を通常のネットワークからではなく SAN から転送しメモリ転送を並列に行うことでライブマイグレーションを高効率化するシステム「Page cache teleportation」を提案・実装する．

3. 第三に，本博士論文では提案したライブマイグレーション高効率化によりデータセンタの消費電力量がどの程度削減されるかを統合評価する．動的 VM 集約の研究およびライブマイグレーション高効率化の研究は数多く行われているが，それらを統合し実際のデータセンタでどの程度消費電力が削減できるかの評価は行われていない．本論文ではシミュレーション評価の結果として，提案したライブマイグレーション高速化手法がデータセンタの消費電力量を数パーセント削減されることを示す．データセンタ全体の消費電力量は極めて大きいため，数パーセントの削減は重要な結果である．

ACKNOWLEDGEMENTS

First, I especially would like to thank Prof. Shinichi Honiden, Deputy Director General at National Institute of Informatics and Professor at Graduate School of Information Science and Technology in the University of Tokyo. It was because of his kind advice and supports that I could enjoy my Ph.D. career and finish this thesis. He has patiently believed me for the whole my career and he allowed me to research whatever I believe important.

My special thanks also go to the thesis committee members, Prof. Shigeru Chiba, Prof. Kei Hiraki, Prof. Hiroshi Esaki, Prof. Mary Inaba, and Prof. Takahiro Shinagawa, for having accepted to be my thesis committee and for giving detailed and fruitful comments on this Ph.D. thesis.

My research activities were guided by Dr. Takahiro Hirofuchi and Dr. Ryousei Takano, both of whom are Senior Researcher at National Institute of Advanced Industrial Science and Technology (AIST). They guided me on many things from finding core problems to tackle in my research, the technical details of the ideas, and to writing papers. Dr. Hirofuchi kindly replied to my sudden email about his research project, and he has guided me closely and continuously since then.

I was given many comments on my research ideas and directions from Prof. Fuyuki Ishikawa, Associate Professor at National Institute of Informatics, Prof. Kenji Tei, Assistant Professor at National Institute of Informatics, and Prof. Shigetoshi Yokoyama, Project Professor at National Institute of Informatics.

During my internship at Microsoft Research, Redmond, USA, I received kind help from my mentor Dr. Gabriel Kliot along with other members of the Orleans team and XCG group. It was my very first time to live and research in a foreign country, but I felt really comfortable thanks to them. I believe this experience really broadened my research directions, as well as my way of thinking things internationally.

In my B.Sc. career in Kyoto University, I was taught the way of academic thinking and presentation by Prof. Hiroshi G. Okuno, currently Professor Emeritus at Kyoto University and Professor at Waseda University, and Prof. Kazunori Komatani, currently Professor at Osaka University. It was a great fortune that I received their advice in the beginning of my research life.

I would love to state my thanks to all members of Honiden laboratory and also of our

friend labs, Fukazawa laboratory and Ohsuga-Tahara laboratory. My five years here were really precious, and with people in Honiden lab, Fukazawa lab and Ohsuga-Tahara lab I did experience many things. Some were enjoyable, some were tough, but all of them are living in me as good memories. My appreciation goes not only to the current members but also to former members and staffs: Prof. Yoshinori Tanabe, Prof. Fumihiro Kumeno, Prof. Kazunori Sakamoto, Masaru Nagaku, Kyoko Oda, Nao Kaneko, Saki Narimatsu, Katsushige Hino, Taku Inoue, Rey Abe, Daisuke Fukuchi, Johan Nystrom-Persson, Ryuichi Takahashi, Yoshiyuki Nakamura, Yukino Baba, Adrian Klein, Florian Wagner, Hirotaka Moriguchi, Jiang Fan, Susumu Toriumi, Valentina Baljak, Yusaku Kimura, Atsushi Suyama, Satoshi Katafuchi, Nobuaki Hiratsuka, Atsushi Watanabe, Mohammad Reza Motallebi, Ryo Shimizu, Yuta Maezawa, Chavilai Sombat, Hisayuki Horikoshi, Hiroki Yoshiike, Katsunori Ishino, Masahiro Ito, Makoto Fujiwara, Shizu Miyamae, Taiken Zen, Yuji Kaneko, Fernando Tarin, Naoki Tsurumi, Shingo Horiuchi, Shoichi Kamiya, Tsutomu Kobayashi, Kazuki Nishiura, Shengbo Xu, Yuki Inoue, Junto Nakaoka, Kiichi Ueta, Kohsuke Yatoh, Shun Lee, Takayuki Suzuki, Kazuya Aizawa, Natsumi Asahara, Tomoya Katagi, Yuta Tokitake, Katsuhiko Ikeshita, Masaki Katae, Miki Yagita, Moeka Tanabe, Yasuhiro Sezaki, Yasuo Tsurugai.

I am also grateful to all my friends who are not only in Japan but also in all over the world, not only from academic field but also from anywhere I have been involved. I was gifted many kind friends in Yamaguchi Senior High School, Kyoto University, The University of Tokyo, internship at Microsoft Research, academic conferences I attended, and many meet-ups and events related to my hobbies. Having friends outside of the laboratory allowed me to broaden my sights in the sense both of research and private life. I wish I were able to put the names of all of them, but my special thanks especially go to: Angelica Lim, Bharadwaj Krishnamurthy, Chen Chen, Daniela Setyobudi, David Devecsery, Eric Chen, Gang Yu, Hiroaki Omi, Hiromitsu Awano, Hiroshi Miyahara, Irina Todoran, Izaaz Yunus, Jianfu Chen, John Vilk, Keita Higuchi, Kousuke Ono, Lanyue Lu, Manjula Peiris, Masami Tazuke, Minoru Nagame, Muntasir Rahman, Naoki Nishikawa, Norihiro Kamae, Shinpei Aso, Shintaro Kasai, Shogo Makihara, Takashi Tamura, Takashi Toyoshima, Takayuki Mizobe, Yasuharu Hirasawa, Yoshiko Matsuda, Yoshitaka Ushiku, Yu Tomita, Yuan Fang, Yuki Matsuda, Yutaka Tokuda.

Lastly but mostly, I want to express my deepest thanks to my family, for believing and supporting me for 27 years of my life. I believe that the education and many chances they gave to me are the most valuable things that I have ever received.

Soramichi Akiyama

RESEARCH ACTIVITIES

PUBLICATIONS IN DOMESTIC JOURNAL

(1) 穐山 空道, 広渕 崇宏, 高野 了成, 本位田 真一. 都鳥: メモリ再利用による連続するライブマイグレーションの最適化, 情報処理学会論文誌コンピューティングシステム, Vol. 5, No. 2 (ACS37), pp. 74 - 85, March 2012.

PUBLICATIONS IN INTERNATIONAL CONFERENCES

(2) Soramichi Akiyama, Takahiro Hirofuchi, Ryousei Takano and Shinichi Honiden. MiyakoDori: A Memory Reusing Mechanism for Dynamic VM Consolidation. In *Proceedings of The 2012 IEEE 5th International Conference on Cloud Computing (IEEE CLOUD)*, pp. 606 - 613, June 2012.

(3) Soramichi Akiyama, Takahiro Hirofuchi, Ryousei Takano and Shinichi Honiden. Fast Wide Area Live Migration with a Low Overhead Through Page Cache Teleportation. In *Proceedings of The 13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pp. 78 - 82, May 2013.

(4) Soramichi Akiyama, Takahiro Hirofuchi, Ryousei Takano and Shinichi Honiden. Fast Live Migration with Small IO Performance Penalty by Exploiting SAN in Parallel. In *Proceedings of The 2014 IEEE 7th International Conference on Cloud Computing (IEEE CLOUD)*, pp. 40 - 47, June 2014.

(5) Soramichi Akiyama, Takahiro Hirofuchi, and Shinichi Honiden. Evaluating Impact of Live Migration on Data Center Energy Saving, In *Proceedings of 6th IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pp. 759 - 762, December 2014.

AWARDS AND HONOURS

(6) Research Fellowship for Young Scientists, Japan Society for the Promotion of Science, April 2013 – March 2015.

(7) IPSJ Computer Science Research Award for Young Scientists, July 2012 .

(8) Best Paper Award in ComSys2011 (1st place out of 11 accepted papers), November 2011.

(9) Student Poster/Demonstration Award in ComSys2011 (1st place out of all student posters and demos), November 2011.

MISCELLANEOUS

(10) *(Non-refereed Talk)* Soramichi Akiyama. Acceleration of Virtual Machine Live Migration on QEMU/KVM by Reusing VM Memory LinuxCon Japan 2012, Student Presentations Track, June 2012.

(11) *(Non-refereed Talk)* Soramichi Akiyama. Fast Live Migration for Data-intensive VMs by Exploiting Storage Area Network in Datacenter. LinuxCon Japan 2014, LinuxCon Staging Session, May 2014.

(12) *(OSS Contribution)* A bug-fix patch to the QEMU project. `http://git.qemu.org/?p=qemu.git;a=commitdiff;h=dde3a2184074f5c4279fd7fbfc597b5dc5859fb8`

TABLE OF CONTENTS

LIST OF FIGURES

List of Tables

CHAPTER 1

INTRODUCTION

1.1 CLOUD COMPUTING

Cloud computing is a computing paradigm where users can borrow/rent a virtually infinite amount of computing resources behind the "cloud" of the internet. More precisely, cloud provider companies equip thousands or tens of thousands of computers on their data centers and allow users to use a part of their computing resources on demand through the internet.

Types of "computing resources" provided in cloud computing have a wide variety. In the lowest layer, virtual computers are provided with all the privileges on them enabled by hypervisors (e.g. KVM [1], Xen [2]) or OS level virtualization (e.g. OS containers [3] such as Open VZ [4] or Jails in FreeBSD [5]). This type of cloud computing is called *Infrastructure as a Service (IaaS)* and the most famous example is Amazon EC2 [6]. In the middleware layer, clouds provide software platforms that make it possible to easily develop, deploy, operate, and scale customized applications on them. This type of cloud computing is called *Platform as a Service (PaaS)* and the examples include Google App Engine [7] and Salesforce1 Platform [8]. In the top layer, software systems that used to be hosted in each organization are replaced by using cloud computing. This type of cloud computing is called *Software as a Service (SaaS)* and the examples include Office 365 [9] (Microsoft Office in the cloud) and SAP Cloud Applications [10] (business management software in the cloud).

The number of users and use-cases of cloud computing has become huge. According to a report in which 141 chief information officers were involved, 9.7% of all computing workloads are executed using cloud computing as of October 2013 and the amount is predicted to rise to 30.2 % in five years [11]. Amazon introduces on the web site that its

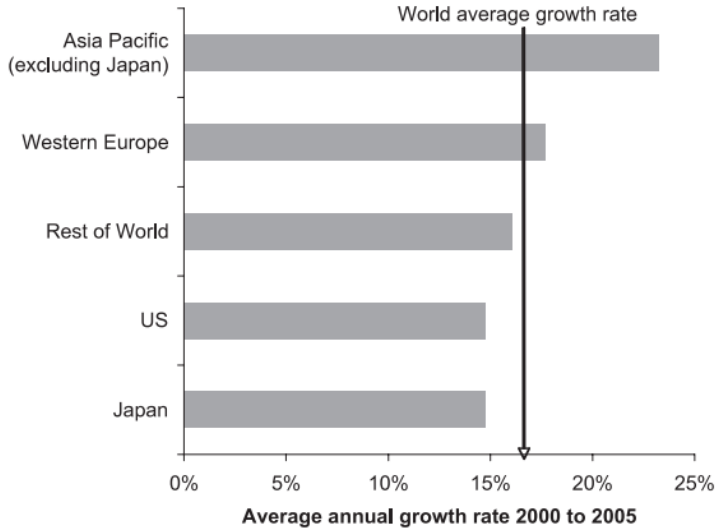


FIG. 1.1. Annual Growth of Data Center Energy Consumption between 2000 and 2005, cited from [13]

cloud service is used by many big players of the IT industry, including Adobe, Citrix, Nokia, NASA and FourSquare [12].

Building and maintaining clouds is involved in bodies of knowledge on system level techniques such as operating systems, virtualization techniques, networking, filesystems, and programming paradigms. Therefore, the trend of cloud computing has attracted many system level researches. The contents of this thesis are categorized in virtualization techniques, in the context of system level researches to build efficient clouds.

1.2 ENERGY CONSUMPTION OF CLOUD DATA CENTERS

In response to the emerging trend of cloud computing described in Section 1.1, energy consumption of data centers that host the clouds is becoming enormous. In the United States of America, it is reported by its Environmental Protection Agency (EPA) that 1.5% of the overall electricity consumption in the country was due to data centers in 2006 [14]. The report claims that this consumption "is similar to the amount of electricity used by the entire U.S transportation manufacturing industry (which includes the manufacture of automobiles, aircraft, trucks, and ships)". A research on data center energy efficiency estimates that the sum of the energy cost across five Google data centers in the US reaches up to $1.4 million per month [15]. Another study reports that average annual growth of data center energy consumption between 2000 and 2005 was around 15% in Japan and

the US, and 16.7% in the worldwide [13] (Figure 1.1).

Reducing energy consumption of data centers is thus an important topic both from the economical and environmental aspects. Many researches have been done from various directions. Following are examples of researches that focus on reducing energy consumption of data centers.

WORKLOAD ANALYTICS This type of researches analyze characteristics of workloads running on the target data center to adopt suitable energy strategy for each workload. Verma *et al.* predict suitable number of VMs a-priori by analyzing static daily load patterns [16]. Tsai *et al.* exploit static load pattern of real data center to propose semi-static VM placement, which prevents live migration because it is a high-cost operation as focused on this thesis [17]

SOFTWARE IMPROVEMENT This type of researches change/improve software or algorithms used in the workloads. Xu *et al.* reduce the peak energy consumption of a data center by partial execution of workloads; a web search task is terminated before the full completion when it has consumed too much energy [15]. DeVuyst *et al.* propose process migration between heterogeneous instruction set architectures to utilize the difference of energy characteristics between them [18].

FACILITY IMPROVEMENT This type of researches change/improve the data center facility itself, but not servers/software, to improve energy efficiency. Endo *et al.* make servers fan-less by fully utilizing facility fans of a data center to reduce energy consumption of cooling [19]. Durr *et al.* offload idle applications from servers to raspberry-pies, which consume just few Watts [20]. The offloading is achieved by designing the applications as stateless ones, because state migration between an x86 server and an ARM-based low power machines is not yet achieved.

CLEAN ENERGY PROMOTION This type of researches encourage the use of clean energy (or carbon-free energy in other words) instead of actually reducing energy consumption. Moghaddam *et al.* migrate VMs across data centers to use clean energy as much as possible [21]. Singh *et al.* introduce a new type of virtualized server, named *transient server*, that is powered by clean energy [22]. Stability of clean energy supply is sensitive to climate changes, but they mitigate this problem by letting the VM users allow shutdown or partial data lost of their transient servers, in return of cheaper pricing.

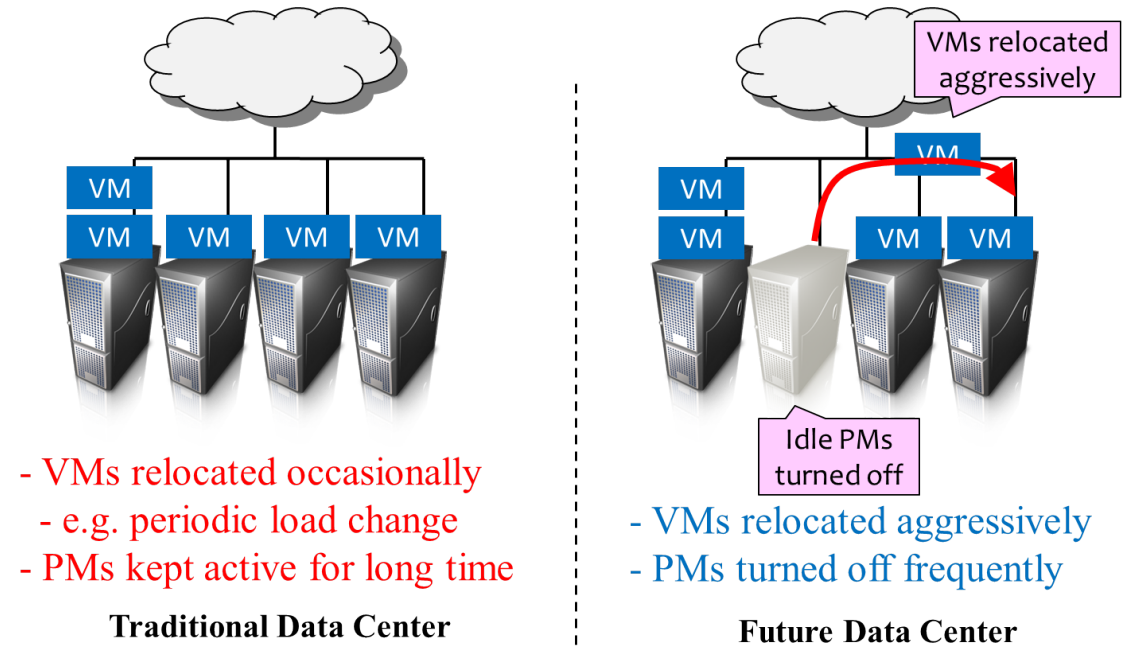


FIG. 1.2. Traditional Data Center v.s. Future Data Center.

1.3 THE GOAL AND THE REQUIREMENTS

1.3.1 FUTURE DATA CENTER MODEL

Given the situation that data center energy consumption is of great concern today, the goal of this thesis is to achieve green data centers with aggressive VM relocation using efficient live migration. The word "relocation" means to change the mapping of VMs and PMs (e.g. VM_n is hosted on PM_m before relocation, but it is moved to $PM_{m'}$ after relocation), in order to achieve lower energy consumption. Live migration is a virtualization technique to move VMs from one PM to another, which is explained later in Section 2.1. The differences between the data center model this thesis aims and a traditional data center model are illustrated in Figure 1.2 and described in the following.

TRADITIONAL DATA CENTER MODEL In traditional data centers, mapping of VMs over PMs is decided offline at the time of involving VMs, and VM relocation occurs only when a relatively large situation change happens. Examples of large situation changes are physical maintenance of a whole rack (all VMs of the rack are relocated to another rack) or large load change based on the users' access patterns such as dramatic load increase during lunch time. This model yields

unnecessary uptime of PMs because even if the load of a PM turns to be light the PM cannot be turned off.

FUTURE DATA CENTER MODEL On contrast to the traditional data center model, VM relocation occurs aggressively at runtime. This model can reduce uptime of PMs by consolidating VMs into small number of PMs, which in turn reduces the total energy consumption of the whole data center. Requirements to achieve this model are discussed in Section 1.3.2.

1.3.2 WHAT THIS THESIS TACKLES TOWARD THE GOAL

To achieve the greener data center model, this thesis tackles two requirements.

EFFICIENT LIVE MIGRATION TECHNIQUE Live migration is used to relocate VMs dynamically at runtime of the data center operation. Although live migration is an established technique and there are many researches on accelerating it, this thesis claims that live migration is still a high-cost operation in some cases. Therefore this thesis explores how to lower the cost of live migration with regard to energy consumption.

OVERALL EVALUATION OF ENERGY CONSUMPTION "Overall" evaluation means that time and energy cost of live migration must be taken into account when evaluating energy aspect of dynamic VM consolidation. Existing studies of dynamic VM consolidation simply ignore all cost of live migration, or take only the time cost into account. However, some studies have pointed out that live migration incurs a certain amount of extra energy overhead due to the load increase. Therefore, this thesis evaluates energy consumption of a data center with both time and energy cost of live migration calculated.

1.3.3 WHAT THIS THESIS DOES NOT TACKLE

The future data center model this thesis aims requires another piece: VM relocation algorithms to calculate the best mapping of VMs over PMs. The algorithms are also known as VM "consolidation" algorithms, if one wants to emphasize to consolidate VMs. However, this thesis does not explore the algorithms because VM relocation algorithms have been widely and deeply researched as described below.

Srikantaiah *et al.* figure out that the least energy consumption per transaction is

achieved when a server is moderately loaded [23], Lee *et al.* evaluate various consolidation heuristics in terms of performance impact on various resources [24], Cao *et al.* focus on *energy-performance tradeoff* and reduces energy consumption while guaranteeing SLAs by establishing a mathematical algorithm to decide if a host can violate SLAs with high probability [25]. Some studies take the time cost of live migration into account when designing dynamic VM consolidation algorithms. Goiri *et al.* propose an energy-aware scheduling of VMs with time overhead of live migration taken into account [26], and Lim *et al.* introduce margins to server and data center capacities to absorb latencies of VM consolidation changes due to the time cost of live migration [27].

1.4 LIVE MIGRATION EFFICIENCY

1.4.1 HOW IT IMPACTS DATA CENTER ENERGY

TABLE. 1.1. Impact of Migration Efficiency on Data Center Energy Saving. The reduction ratio is calculated against the case that all PMs are always on without dynamic VM consolidation.

Method	Migration Efficiency	Energy Reduction
Naïve	Slow, Large memory transfer	25.7 %
MiyakoDori (Chapter 3)	Fast, Small memory transfer	29.6 %

Table 1.1 briefly introduces impact of total migration time on data center energy efficiency. The "energy reduction" columns show how many percentage of energy consumption is reduced against the case where all PMs are always turned on, by applying a simple dynamic VM consolidation algorithm (detailed settings and results are showed in Chapter 5). The reduction ratio is 25.7 % when a naïve live migration mechanisms implemented in the original QEMU/KVM is used and it improves to 29.6 % when an accelerated live migration mechanism developed by us is used.

The difference by 4% in Table 1.1 is a large one because the overall energy consumption of a data center is huge as shown in Section 1.2. Suppose the Power Usage Effectiveness (PUE) of google data centers described in [15] is less than 2. This means that more than the half of total energy consumption of the data centers is used for servers (Note that The values in the table are calculated on server energies). Therefore, the 4% difference is regarded more than 2% of the total energy consumption, which results in more than $28,000 / month energy cost difference.

TABLE. 1.2. Focuses of Existing Live Migration Studies

Focus	Example of Existing Studies
Accelerating 1st phase	[28], [29], [30]
Accelerating 2nd phase	[31], [32], [33], [34], [35], [36], [37], [38], [39]
Other aspects of live migration	[40], [41], [42], [43], [44]

1.4.2 TECHNICAL CHALLENGE

Given that live migration is used within a data center as focused in this thesis, the main concern is how to efficiently transfer memory pages of the target VM (for the technical details of live migration please refer Section 2.1 in Related Work). The memory transfer can be factored into two phases that are equally important: the 1st phase where all memory pages of the target VM are transferred sequentially, and the 2nd phase where memory pages updated during the 1st phase are transferred again repeatedly (details are also in Section 2.1 in Related Work). The amount of memory transferred in the 1st phase is equal to the size of the target VM's memory usage. Thus, to migrate a VM that occupies 4 GB of memory, a 4 GB of data transfer is required in the 1st phase. The amount of memory transferred in the 2nd phase, on the other hand, depends not on the memory usage but on the size and the update speed of the working set of the VM.

Among the two phases of live migration, the cost of the 1st phase is not yet fully reduced in existing studies. The 1st and the 2nd phases are equally important as a target of optimization, because they both can transfer many memory pages. For workloads that do not update memory pages frequently, the 1st phase is more time consuming than the 2nd phase. Table 1.2 shows the focuses of existing studies on efficient live migration. The table shows that many existing studies focus on the 2nd phase of live migration.

This thesis defines the technical challenge for efficient live migration to achieve greener data center is how to reduce the cost of the 1st phase of live migration. A biggest technical difference between reducing the cost of the 1st phase and the 2nd phase of a migration is that in the 2nd phase the destination PM of the migration has an old copy of the memory pages, but in the 1st phase it has almost no information about the target VM. Therefore, this thesis claims reducing the cost of the 1st phase is more technically challenging than reducing the cost of the 1st phase.

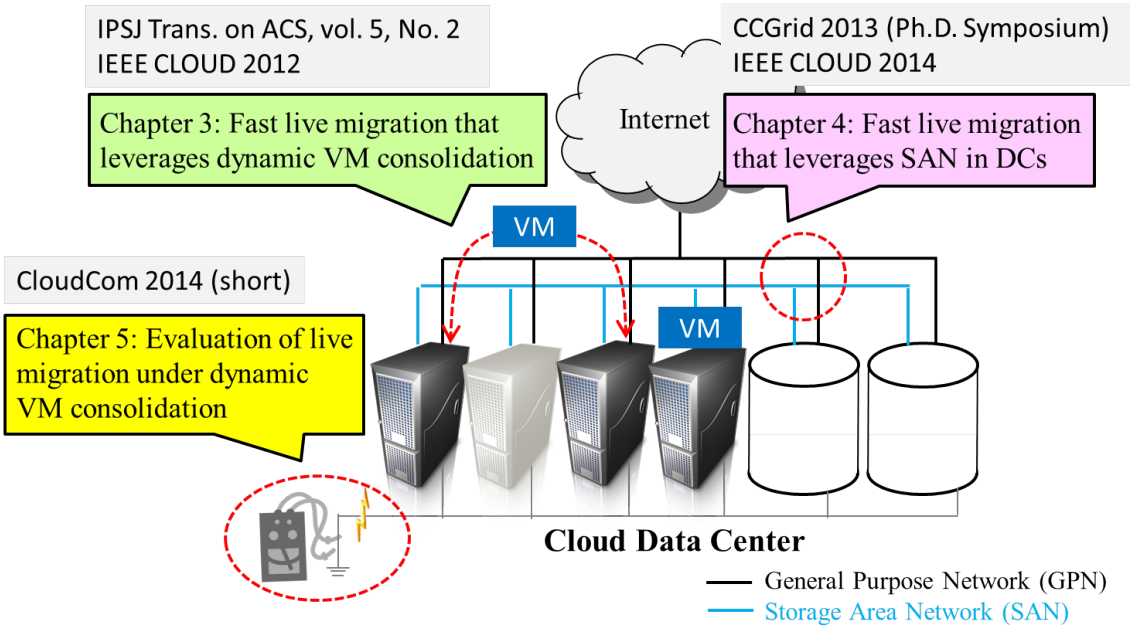


FIG. 1.3. Overview of This Thesis.

1.5 OUTLINE OF THIS THESIS

The outline and the structure of this thesis is explained here. Figure 1.3 shows the relationship of the three main parts (chapters 3, 4, and 5).

CHAPTER 1. INTRODUCTION

This chapter explains the background, the goal that this thesis aims, and the technical challenge to achieve the goal.

CHAPTER 2. RELATED WORK

This chapter refers related work of this thesis in the origin of live migration, its well-known implementations, and recent researches that achieved accelerated live migration using various techniques.

CHAPTER 3. MIYAKODORI

This chapter describes a new live migration acceleration mechanism called MiyakoDori, which reduces total migration time and amount of transferred memory in a migration. The idea is that, in a dynamic VM consolidation system, VMs can migrate among a small set of PMs and a VM can migrate back to a PM on which it was executed before.

MiyakoDori leverages this situation to accelerate live migration by caching VMs' mem-

ory images in PMs and transfer only update memory pages in live migration. The evaluation shows that MiyakoDori reduces total migration time and the amount of transferred memory more than 90% in CPU intensive and web server based workloads.

MiyakoDori reduces total migration time and amount of transferred memory and results in improving both extra energy consumption and short sleep time incurred by live migration, and the impact on the overall energy saving of a data center is evaluation in Chapter 5.

CHAPTER 4. PAGE CACHE TELEPORTATION

This chapter describes another new live migration acceleration mechanism called Page cache teleportation. First, this chapter figures out that large amount of page cache in VM memory is the biggest cause of long total migration time under IO-intensive workloads. After that the system to mitigate large page cache problem by utilizing SAN in data centers in parallel with normal migration network is proposed. The evaluation shows that Page Cache Teleportation shortens total migration time under various IO-intensive workloads.

Page cache teleportation reduces total migration time and results in improving short sleep time incurred by live migration, whose concrete impact on the overall energy saving of a data center is evaluation in Chapter 5.

CHAPTER 5. ENERGY OVERHEAD OF LIVE MIGRATION

This chapter conducts an integrated evaluation of energy overhead of live migration and energy reduction achieved by dynamic VM consolidation. The thoughts behind this chapter is that there are few studies that conduct this integrated evaluation even though plenty of live migration studies exist. This chapter shows how to model MiyakoDori in terms of energy overhead, total migration time and amount of transferred memory and executes large scale simulations to show how MiyakoDori reduces overall energy consumption of a data center.

CHAPTER 6. CONCLUSION AND FUTURE WORK

This chapter summarizes the thesis and gives future directions on the whole live migration research area, focusing on requirements to make live migration applicable in real data centers.

CHAPTER 2

RELATED WORK

2.1 LIVE MIGRATION OF VIRTUAL MACHINES

2.1.1 OVERVIEW

Live migration is a technique that allows VMs to move around from one physical machine (PM) to another. Unlike a naïve method that first makes the target VM sleep on the source PM and then resume it on the destination PM (also kown as *cold migration*), the greatest advantage of live migration is that target VM undergoes short period of service disruption.

Live migration is an essential technique to aggressively relocate VMs of a data center at runtime because users of the VMs never want their services to be interrupted by changes of VM allocations. Suppose cost migration takes one minute to transfer all VM states and a cold migration occurs once per week, then the ratio of the VM downtime against the whole data center operation is:

$$1 \div (60 \times 24 \times 7) \approx 0.01\%. \tag{2.1}$$

This is the longest service downtime in order no to violate the SLA of 99.99% uptime, which is common in today's cloud era. Therefore, cold migration cannot be used for aggressive VM relocation and live migration is essential.

2.1.2 TECHNICAL DETAILS

The main technical focus of live migration when it is used within a data center is how to efficiently transfer the memory of a target VM from the source PM to the destination PM.

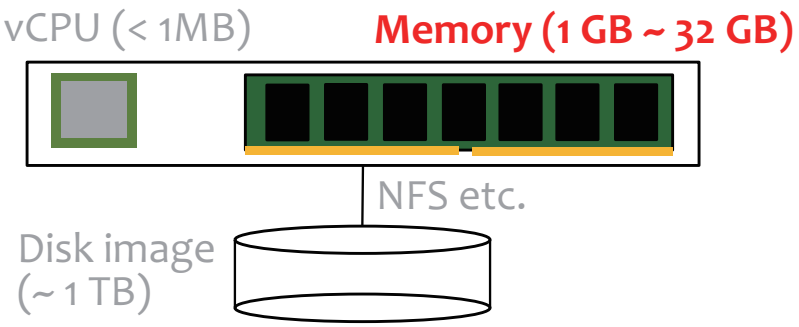


FIG. 2.1. Data to be Transferred in Live Migration. The main focus is memory pages of the target VM. The disk image is shared using NFS in practice, and CPU registers/device states are negligibly small.

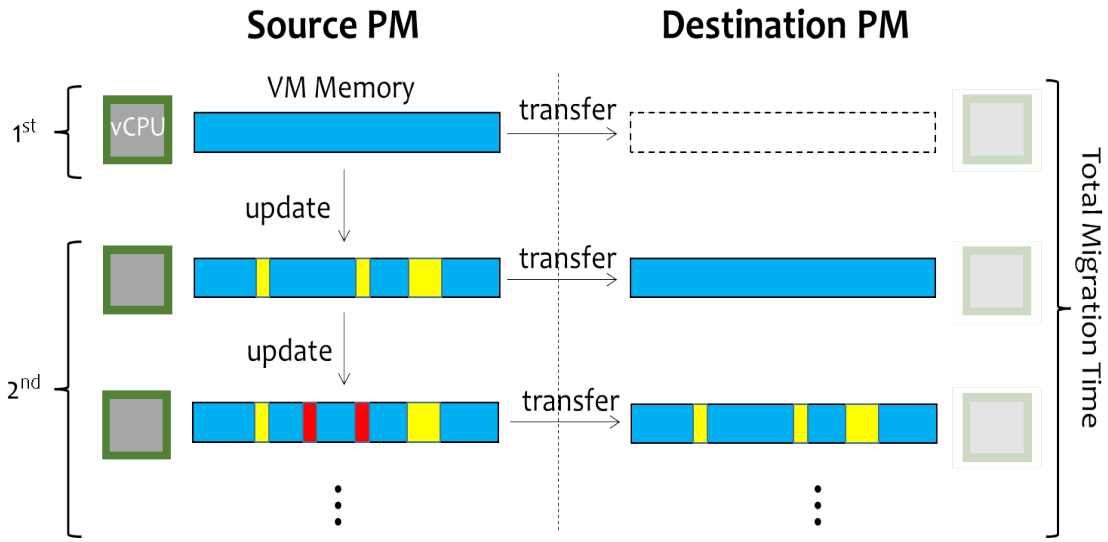


FIG. 2.2. Memory Transfer in Pre-copy Live Migration

Live migration of a VM requires to transfer three types of data of the target VM which are described in Figure 2.1.

1. Memory pages (1 GB to tens of GB)
2. Disk image (tens of gigabytes to a terabytes)
3. Device states such as CPU registers (less than a megabyte)

The disk image is too large to transfer at a migration time thus the practice is to locate it on a shared storage such as NFS or iSCSI. The device states are in contrast very small and never incur problems on transferring them. Therefore, the biggest issue of live migration is how to transfer the memory of the target VM efficiently.

The very original method of live migration, Pre-copy live migration, was proposed by Nelson *et al.* [45] and Clark *et al.* [46] in 2005. Since then it has been widely adopted by hypervisors such as KVM [1] and Xen [2], and also by production software such as VMWare vSphere [47] and Oracle VirtualBox [48][*1]. Figure 2.2 shows the procedure of pre-copy live migration that is explained as follows:

1. All the memory pages of the migrated VM are transferred at first. For example, if the VM's memory footprint is 1 GB, the amount of transferred memory in this phase is also 1 GB (1^{st} phase).
2. Some memory pages are updated while other pages are transferred because the VM keeps running during a migration. The updated memory pages are repeatedly transferred until the number of remaining memory pages become sufficiently small (2^{nd} phase).
3. The VM is instantly suspended to transfer the remaining memory pages and device states (3^{rd} phase).

There are two terminologies related to the time taken by live migration:

TOTAL MIGRATION TIME refers the time length that a migration takes from its beginning to the end. This is almost dominated for the time consumed to transfer the memory pages, because other data to be transferred (vCPU registers and device states) are negligibly small compared to the main memory as described above. Typical total migration time ranges from several seconds to few minutes.

DOWNTIME refers the time length during which the migrated VM is suspended due to a migration. Downtime depends on the speed of memory update of the target VM and the actual bandwidth available to transfer memory. Typical downtime ranges from tens of milliseconds to few seconds.

Among the two, this thesis focuses on total migration time, which largely impacts the energy efficiency of data centers when VMs are aggressively relocated with live migration.

[*1] Live migration is called *vMotion* in the VMWare terminology, and *teleporting* in the Oracle terminology.

2.2 THE ORIGIN OF LIVE MIGRATION

Sapuntzakis *et al.* [49] was the first to propose migrating a virtual machine (called *capsule* in their terminology) in 2002. 20 years of gap is there between this and inventions of two concepts from which it can directly stem, namely virtual machine and process migration. Virtual machine technology had been proposed by IBM in 1960s [50], and process migration had already been implemented in some operating systems in early 1980s [51]. An important reason of the birth of the virtual machine migration is changing demands of users. A year before [49], Chen *et al.*claimed in [52] that process migration is not enough to serve users' demand of mobility. Sapuntzakis *et al.*also claim that usages of their virtual machine migration include user mobility as well as easy management of underlying physical machines, both of which are regarded to be unique characteristics of today's cloud computing.

The main focus of [49] was how to transfer a large disk state with a narrow DSL link. The migrated virtual machine is suspended during a migration thus the migration was not "live". As the cloud paradigm becomes popular, it evolved into "live" migration and researches have also focused on how to efficiently transfer memory pages while they are being updated by the running VM. This focus shift is because of two reasons. From the user demand perspective, cloud users require availability more than 99%, therefore live migration is mandatory. From the technical perspective, gigabits or even 10 gigabits ethernet have become common in data centers, thus large disk images can be simply located on a shared storage if storage performance is not the primary concern.

2.3 ACCELERATING PRE-COPY LIVE MIGRATION

Many researchers have tried to accelerate pre-copy live migration using various techniques.

Similarities between memory pages are important to accelerate pre-copy live migration. Svärd *et al.* [31] exploit "temporal" similarity within VM memory by a fixed-size cache to store memory pages that are once transferred during a migration. When the memory pages are updated and transferred again, only the difference between previous versions are transferred. This technique can reduce both the total migration time and the downtime because the downtime depends largely on the ratio between the speeds

of memory transfer and memory updating. They claim that in the normal pre-copy live migration downtime-sensitive applications such as SAP system cannot be migration because the database transactions are lost, but with their acceleration VMs hosting SAP systems can be safely migrated.

Zhang *et al.* [32] exploit "spatial" similarity within VM memory by detecting data duplications within it. When a memory page is transferred, they transfer only the difference between it and a reference page that is similar to it and that has already been transferred. They report that a great portion of memory pages are similar in VMs running various workloads, for example, 43.02% of non-zero memory pages have similarity above 80% each other in an Ecommmerce workload. Jin *et al.* [33] also use "spatial" similarity, with thorough analysis of characteristics of the similarity in various workloads. They found that similarness largely differs depending on the workload running on the VM. From this observation, they adopt a memory compression mechanism that has two phases: (1) detecting in-page similarness of each memory page (2) compression the target page with appropriate compression algorithm.

Another direction of accelerating pre-copy live migration is to use memory access patterns as hints to decide which memory pages should be transferred before others. Du *et al.* [34] clusters memory pages of the target VM into fixed-size regions and transfer them in ascending order of the updated speed of each region. Once accumulated updated speed exceeds the available network bandwidth for the migration, remaining regions are not transferred in the iteration, because it is highly possible that transferring them turns into vain by new updates to the region. Ibrahim *et al.* [40] examines how memory pages are updated and transferred in a pre-copy live migration for HPC applications. They found that there are three patterns: (a) memory update rate is far beyond memory transfer rate thus continuing live migration is impossible, (b) the application sometimes enters into slow memory update phase (e.g. in a synchronization operation) thus the VM should be suspended and moved to the destination PM at this moment, and (c) memory transfer rate is competitive to memory update rate thus pre-copy live migration works well. They detect a pattern applicable to the target VM by sampling memory update/transfer rates and change migration strategy depending on the detected pattern.

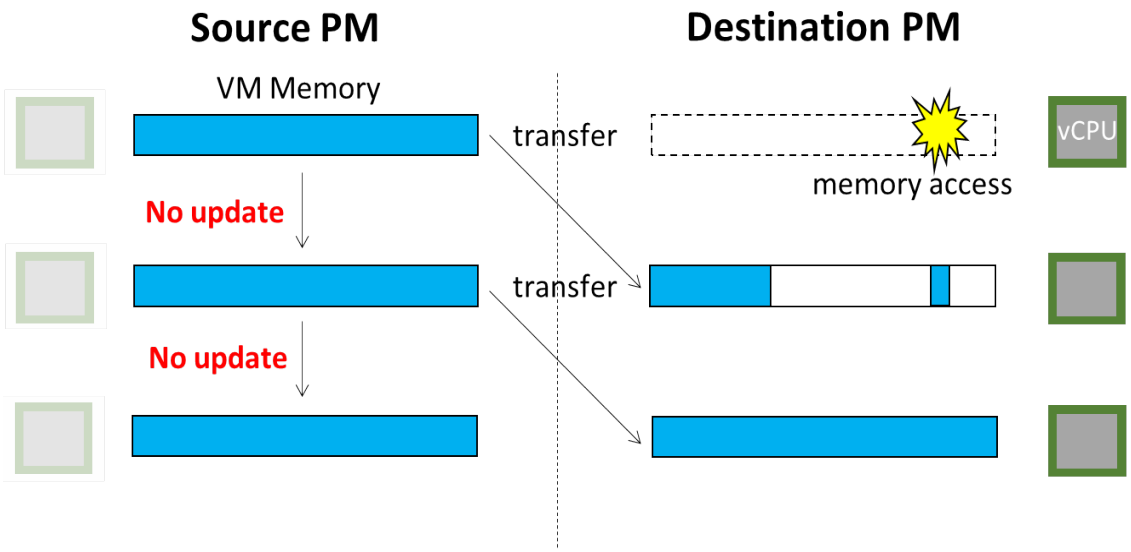


FIG. 2.3. Memory Transfer in Post-copy Live Migration

2.4 POST-COPY LIVE MIGRATION

Post-copy live migration was a big step toward more efficient live migration. It was proposed and implemented by some researchers almost at the same time. Mirkin *et al.* [35] achieved post-copy migration of containers, Hines *et al.* [36] achieved it for Xen virtual machine monitor, and Hirofuchi *et al.* [37] showed that it can be implemented with small modifications on the existing QEMU/KVM architecture.

Post-copy live migration accelerates live migration by omitting the iterative copy phase of pre-copy live migration entirely. It transfers device states of a target VM before transferring memory pages and the execution host of the VM is switched right after the migration is invoked. Memory pages of the VM is lazily fetched while the VM is running on the destination PM. The important point is that the memory pages left on the source PM are never updated thus a memory page is transferred only once.

A drawback of post-copy live migration is that it degrades performance of the migrated VM due to page faults occurred by lazy memory page transfers. In order to mitigate this issue, memory pages are pre-fetched using locality of memory accesses in [36] and [37]. Bryant *et al.* [53] clusters memory pages into kernel pages, user pages, and free pages by analyzing page table entries and OS specific information to find which memory pages should be transferred in priority.

2.5 OTHER DIRECTIONS OF MIGRATION RESEARCH

2.5.1 MIGRATING GROUP OF VMS

How to migrate a group of VMs efficiently has also attracted research interests. Typical use cases of migrating a group of VMs are moving all VMs hosted on a PM to escape from hardware faults/maintenance and migrating a logical set of VMs (e.g. set of web servers owned by the same user) from a PM to another.

Deshpande *et al.* [28] propose an efficient way to migrate a group of VMs. The basic idea is that VMs running the same/similar OSes or applications have many memory pages identical/similar to each other. They exploit this characteristic by transferring identical memory pages only once and transferring only the difference between the target page and a similar page that has already been transferred.

Ye *et al.* [41] survey the impact of various live migration strategies on the performance of migrating a group of VMs. The strategies include migrating VMs one by one, migrating some VMs concurrently, and also some patterns for migration order of VMs; suppose there are two groups (VMs 1, 2, 3 and VMs a, b, c) to be migrated, the question is weather two groups should be transferred sequentially (migrate 1, 2, 3 then a, b, c) or interleaved (1, a, 2, b, 3, c). Their claims include that concurrent migrations must be in a suitable granularity; i.e. migrating many VMs at once decreases the migration performance due to excessive network load, and that interleaved migration order should be avoided because it causes inter-PM communications by VMs belonging to the same group but hosted temporarily in different PMs.

2.5.2 VMS WITH PASS-THROUGH DEVICES

Pass-through devices (e.g. SR-IOV [54]) greatly decrease IO performance overhead of VMs thanks to reduced guest/host switching, but they make live migration much difficult.

Zhai *et al.* [42] enable migrating VMs using pass-through NICs. Device states of NICs inside the migrated VM are first saved into the VM memory by utilizing PCI hotplug technology. NICs are plugged off on the source PM before a migration, and then replugged on the destination PM after the migration finishes. In order to guarantee network connectivity while pass-through NICs are plugged off, Linux bonding driver is

used to automatically switch to virtualized NICs.

Takano *et al.* [43] push this technique a step forward; they migrate VMs using pass-through Infiniband interfaces into PMs that do not have Infiniband. A typical usage of their proposal is to migrate VMs from an Infiniband-capable cluster into a spare cluster with ethernet networking in an emergency. The most important issue in this work is that switching from Infiniband to ethernet (on which TCP/IP can be used) means switching from one networking protocol to another. Therefore, distributed applications that consist of multiple VMs should aware of the fact that one of the VMs switch from Infiniband to ethernet. They mitigate this issue by forcing all VMs composing a distributed application switch to ethernet at the same time with the help of Symbiotic Virtualization technique [55].

2.5.3 INTROSPECTION-BASED APPROACHES

VM Introspection is a technique to allow the host OS of a VM to analyze the contents of the VM memory without injecting any program inside the VM. This technique has been applied to many purposes, and live migration is one of the applications.

Bryant *et al.* [53] uses VM Introspection for VM cloning instead of migration, but the main task is the same as VM migration (i.e. how to transfer memory pages efficiently). They use page table information and OS-specific information (namely the list of page structures in Linux) to distinguish memory pages of the target VMs into five categories: kernel data, kernel code, user code, user data, and page cache. The categories are used to find the best number of pages to pre-fetch in their post-copy based VM cloning.

Chiang *et al.* [29] uses VM Introspection to detect free memory pages to accelerate live migration. The point here is that free memory pages cannot be detected by merely observing the binary because a free memory page is not necessarily filled with zero values. Their technique is advanced than existing ones in the sense that they do not need OS-specific hard coding to achieve VM Introspection.

2.5.4 OTHER ACCELERATION TECHNIQUES

Recent researches try to accelerate live migration by breaking through the pre-copy/post-copy schema.

Liu *et al.* [56] and Gerofi *et al.* [39] utilize checkpoint and restart technology to achieve fast live migration. The first phase of their methods are the same as normal pre-copy live

migration; i.e. all memory pages are transferred once in the beginning. However, they do not transfer updated memory pages but they transfer execution logs of the migrated VM instead. An execution log is smaller than the actual memory content that the log reproduces in design, therefore the methods take less time than pre-copy live migration.

Kim *et al.* [30] switch the execution host of the target VM right after a migration is invoked, but they keep the VM on the source node running even after that. The VM left on the source node is used to *guide* memory transfer during the migration; the memory access pattern of the left VM is used as a hint to know which memory pages have stronger possibility to be accessed by the migrated VM.

2.5.5 HETEROGENEOUS PLATFORM LIVE MIGRATION

Live migration between heterogeneous platforms is also an important issue. For example, live migration between KVM and Xen enables moving VMs between two data centers hosted by different organizations.

Liu *et al.* [57] propose a broker-based approach to achieve heterogeneous platform live migration. They analyze abstraction and migration protocol differences among KVM and Xen, and implement a system that buffer the differences and enable live migration between them. Takahashi *et al.* [58] propose WinKVM, a KVM port onto Windows by providing an abstraction layer that emulates needed Linux kernel APIs. They claim that WinKVM enables live migration a VM into laptops when the user must be offline during commuting or so.

2.5.6 HETEROGENEOUS ISA LIVE MIGRATION

Leveraging characteristic differences of multiple instruction set architectures (ISAs), most commonly x86 and ARM, is a traditional methodology to reduce energy cost. x86 has been adopted for high power servers because of its performance, while ARM has been adopted for SoC, mobile phones, or network facilities because of its energy efficiency.

Virtualization and live migration are also catching up this direction. DeVuyst *et al.* [18] theoretically achieve process migration between x86 and ARM architectures on a simulator. Although their work is still on a simulator due to the lack of actual hardware that supports multiple ISAs, the approach is cutting-edge to achieve more energy efficient data centers. Dall *et al.* [44] propose KVM on the ARM architecture, and their outcome

has already been adopted by the mainline Linux kernels. Combining these two techniques could bring heterogeneous ISA live migration, which enables a VM to migrate from an x86 server to an ARM server. This direction will surely open a new frontier for energy efficient data centers.

2.6 LIVE MIGRATION V.S. DATA CENTER ENERGY SAVING

A big motivation to accelerate live migration is to help reduce energy consumption of cloud data centers, as focused in this thesis.

2.6.1 MODELING LIVE MIGRATION IN TERMS OF ENERGY

Model energy consumption of live migration is one direction. Liu *et al.* show extra energy consumption caused by the pre-copy live migration depends only on the amount of transferred memory [38]. Even though they figure out faster network bandwidth consumes much energy per time unit, the accumulated sum during the total migration time is shown to be the same. Aikema *et al.* compare how extra energy consumption of live migration changes depending on workload type and transport type of memory data (encrypted and non-encrypted) [59].

2.6.2 CONSIDERING COST OF LIVE MIGRATION

It is also important to considering cost of live migration when designing VM relocation algorithms.

Hossain *et al.* [60] propose a VM relocation algorithm that considers extra energy consumption of live migration. They reduce a data center's overall energy consumption by 12% compared to existing algorithms.

Goiri *et al.* [26] figure out cost of live migration must be considered as a penalty when conducting VM relocation. The novelty of their algorithm is that it considers the time required to create a new VM and to migrate an existing VM when calculating energy-efficient VM placement.

Borgetto *et al.* [61] figure out during a live migration the VM stays both on the source and the destination PM. They modify existing VM relocation algorithms to more energy-aware ones by incorporating this idea.

CHAPTER 3

MIYAKODORI

3.1 INTRODUCTION

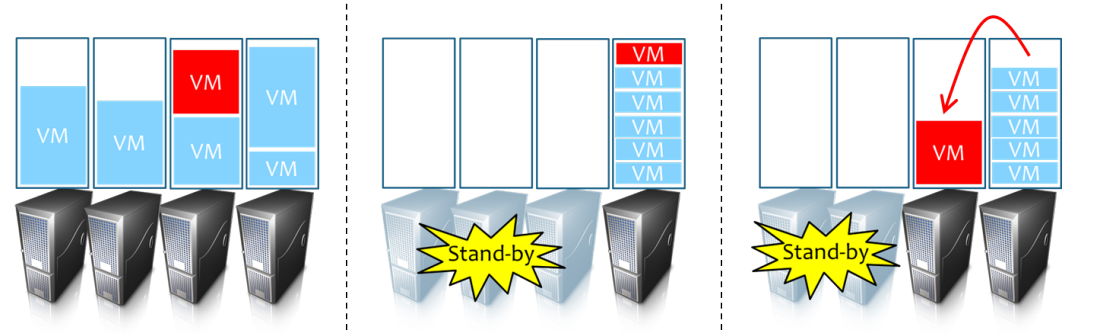
As described in Chapter 1 (Introduction), efficiency of live migration largely impacts the overall energy saving of a data center under aggressive VM relocation. In this chapter, *MiyakoDori*, a memory reusing technique to significantly shorten total migration time and reduce amount of transferred memory in a migration is proposed.

The core idea of MiyakoDori is that VMs migrate among a certain set of PMs (for example, PMs belonging to the same rack) again and again under aggressive VM relocation at runtime. In this situation, the method accelerates live migration of a VM when it migrates back to the PM on which it was executed before by keeping old memory image on the PM and reuse non-updated memory region. This idea significantly reduces amount of transferred memory for migrating back, and in turn reduces total energy consumption of data centers by being combined with dynamic VM consolidation.

This rest of this chapter is structured as follows. Section 3.2 describes the core ideas of MiyakoDori, namely *migration back* and *memory reusing*. Section 3.3 proposes MiyakoDori, a live migration acceleration mechanism that leverages the core ideas. Section 3.4 explains evaluation methodology and the results. Section 3.5 discusses various overhead of the system, the limitations of its applicability, and future directions. Section 3.6 refers related work and Section 3.7 summarizes this chapter.

A part of this chapter has been published in refereed papers. The copyright of each paper is hold by Information Processing Society of Japan (IPSJ) and IEEE Computer Society, respectively.

1. Soramichi Akiyama *et al.* MIYAKODORI: Optimization for Sequence of Live Mi-



t=t_0: VMs are load balanced

t=t_1: VMs are consolidated after becoming idle

t=t_2: A VM gets busy again, migrates back to the PM where it was hosted at t_0

FIG. 3.1. Migration Back

grations by Reusing VM Memory. *IPSJ Transactions on Advanced Computing Systems*, 5(2):74-85, March 2012 (in Japanese, awarded **IPSJ Computer Science Research Award for Young Scientists**).

2. Soramichi Akiyama *et al.* MiyakoDori: A Memory Reusing Mechanism for Dynamic VM Consolidation. In *Proceedings of The 2012 IEEE 5th International Conference on Cloud Computing*, pages 606 - 613, 2012.

3.2 CORE IDEAS

3.2.1 MIGRATION BACK

Migration back or *migrating back* refers a live migration whose destination host is one of the PMs where the migrated VM has ever executed before. Figure 3.1 illustrates migration back of a VM. There are six VMs in the system and one of them is colored differently (red) to distinguish it from other ones. At $t = t_0$, the VMs are load balanced across PMs because they are busy. At $t = t_1$, the VMs get consolidated into one of the PMs because they become idle. At $t = t_2$, the red VM becomes busy again and migrates "back" to the PM on which it used be executed at $t = t_0$.

Exploiting heterogeneity to reduce energy consumption is a promising idea. A similar technique has been proposed and proved to be efficient in a single machine level. In [62] and [63], they propose to equip two types of cores, high power/performance and low power/performance, and migrate processes between them in response to the states of processes. Even more aggressive approach is proposed in [18], where process mi-

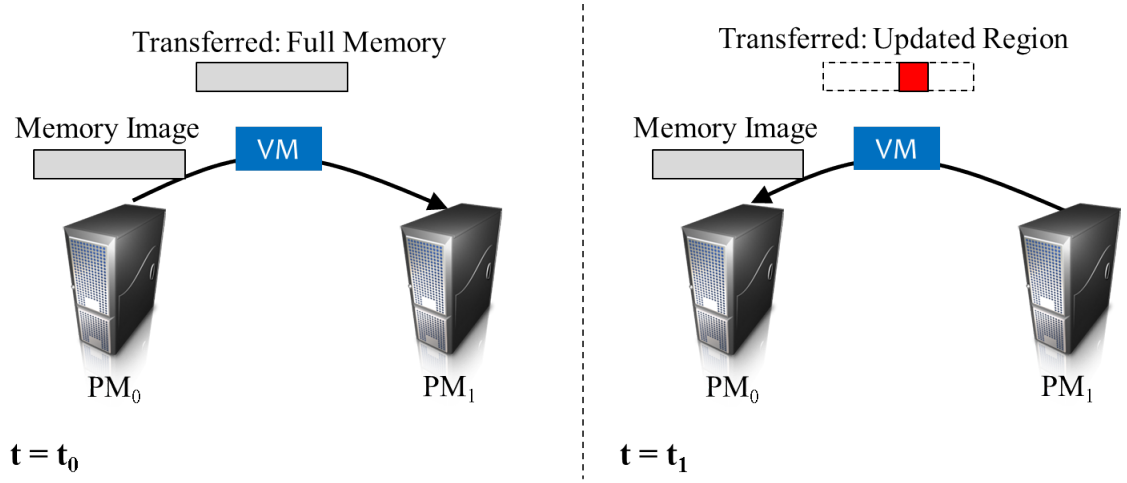


FIG. 3.2. Memory Reusing The upper row illustrates the first migration from PM_0 to PM_1, in which memory reusing cannot be applied. The lower row illustrates a migration back from PM_1 to PM_0, in which the memory image of the VM is kept on PM_0 and unchanged memory pages are not transferred.

gration between x86 architecture (high power/performance) and ARM architecture (low power/performance) is achieved in simulations. The author believes that this kind of heterogeneousness researches should be extended and applied to cloud data centers as well, and migrating back will become one of the fundamental techniques that support the heterogeneousness.

3.2.2 MEMORY REUSING

Memory reusing is an idea that a large portion of VM memory does not need to be transferred when the VM migrates "back", to shorten total migration time. In memory reusing, the memory image of a VM that is about to migrate keeps stored on the source PM in order to reuse it when the VM migrates back from another PM at a later stage. Figure 3.2 illustrates memory reusing:

1. At $t = t_0$, a VM is migrating from PM_0 to PM_1. The memory image of the VM keeps stored on PM_0 for future reuse. Note that this migration (*migration out*) is not accelerated by memory reusing thus all the memory pages of the VM are transferred.
2. At $t = t_1$, the VM is migrating back from PM_1 to PM_0. Many memory pages in the VM memory are not updated during t_0 and t_1, although some are updated due to the VM execution. In this migration, only the updated memory pages (red part

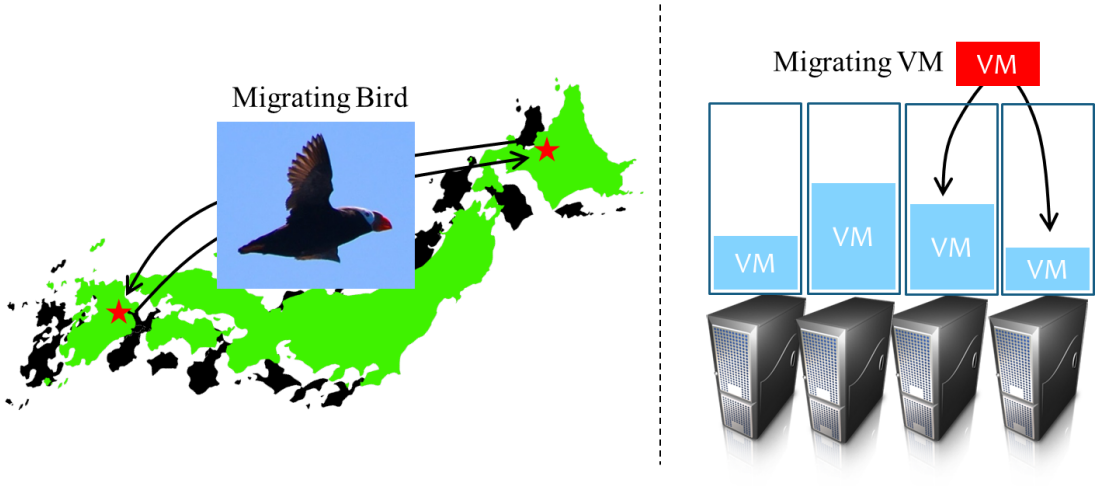


FIG. 3.3. A Metaphorical View of MiyakoDori: VMs Migrate as Birds Do

in the figure) needs to be transferred and other memory pages are *reused* from the memory image stored in PM_0.

3.3 PROPOSED SYSTEM: MIYAKODORI

This section proposes and describes live migration mechanism that enables VMs to migrate back with the memory reusing technique. The developed system is called MiyakoDori. In this section, the design policy and the implementation of MiyakoDori are described.

The name MiyakoDori is based on an old Japanese name of a migrating bird (black-headed gull in modern English). The bird appears in an old Japanese poetry as follows:

名にし負はば　いざ言問はむ　都鳥　わが思ふ人は　ありやなしやと （伊勢物語）

Translating an ancient poetry into modern English might break the atmosphere of it, but the author dare to do it for the readers' convenience:

Let me ask, how is the dearest doing, as you are a MiyakoDori (The Tails of Ise)

Note that *miyako* and *dori* mean the capital and a bird in Japanese respectively, and the author of the poetry found the migrating bird on the way from the capital to the east.

The idea of naming the system is that a VM in our system migrates out and back across PMs, just as a migrating bird flies forth and back across places (Figure 3.3).

3.3.1 DESIGN CRITERIA

There are two important design issues on implementing MiyakoDori.

MEMORY UPDATE DETECTION/EXPRESSION

The question here is how to detect and express that the current and saved versions of a memory pages are different. Two choices are possible for memory detection:

1. Detecting memory update immediately when the update occurs using dirty page tracking functionality. This method has a performance overhead on target VM; i.e. detecting memory updates every time one occurs can slow down memory-intensive workload inside the VM.
2. Calculating has values of each memory page just before a migration and compare them between the current memory and a memory image to be reused. This method has a time overhead on total migration time; i.e. calculating hashes of large memory can take time of the order of seconds.

MiyakoDori adopts the former approach to detect memory update. An experiment showed that the performance overhead to the target VM is negligibly small in real workload. The details of the experiment are shown in Section 3.5 (Discussion). Also the details of the dirty page tracking functionality is described in the next subsection (Section 3.3.2).

IMAGE DIFFERENCE MANAGEMENT

The question here is how to manage differences of memory pages between two memory images. There are two possible software architectures to manage:

1. A client-server architecture (Figure 3.4) where a central server manages all information about memory updates and PMs ask which memory pages can be reused to the server. The advantage of this architecture is that existing dynamic VM consolidation algorithms can be easily modified to use MiyakoDori, because the central server knows every information that MiyakoDori newly introduces.
2. A peer-to-peer architecture (Figure 3.5) where each PM manages the memory updates information and the source and destination PMs exchange the information in a peer-to-peer manner. The advantage of this architecture is that a single point of failure does not exist thus a better scalability is achieved.

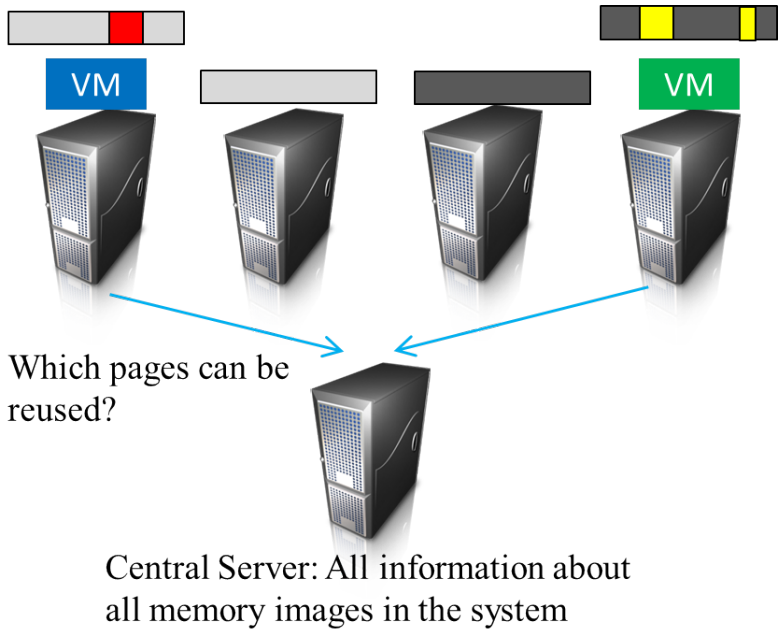


FIG. 3.4. Client-Server Architecture for MiyakoDori: A central server manages all update information about all memory images, and source PMs of live migrations ask which memory pages can be reused to the central server.

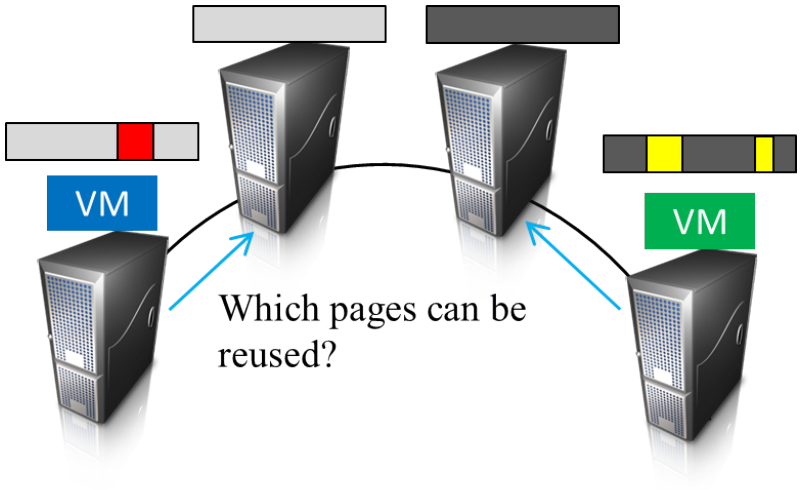


FIG. 3.5. Peer-to-peer Architecture for MiyakoDori: Source PMs of live migrations ask which memory pages can be reused to the destination PMs.

MiyakoDori adopts the client-server architecture. The central server holds a single integer named *generation* for each memory page of every memory images in the system. Therefore the central server is called a *generation server*. As mentioned above, this architecture allows to easily develop advanced dynamic VM placement algorithms integrated with MiyakoDori. For example, the generation server can easily find the PM that has the latest memory image of the target VM, because the generation server has all memory

reusing information. The single point of failure problem can be avoided by simply replicating a generation server because the system does not require simultaneous writes to a generation by two PMs. The detailed algorithm is explained in Section 3.3.3.

3.3.2 DIRTY PAGE TRACKING

Dirty page tracking is a functionality provided by the original QEMU/KVM. A user-level function of QEMU detects via dirty page tracking which memory pages are updated. In x86 and x64 architectures, when a memory page is updated, the CPU sets the dirty bit of the corresponding page table entry (to 1). Dirty page tracking acquires this bit and makes it available to user-level functions.

Since the page table entry is touched only once when the first update to the page occurs, there is practically negligible overhead in using dirty page tracking permanently, as describe in Section 3.5.

3.3.3 LIVE MIGRATION ALGORITHM WITH MIYAKODORI

This section explains the algorithm to migrate a VM with MiyakoDori. As described in Section 3.3.1, a migration involves a generation server. Note that a generation server should not necessarily be an designated PM because can be co-located with VMs, but a generation server is illustrated using a PM in the figures below for simplicity of explanation.

Figure 3.6 illustrates the behavior of MiyakoDori when a VM, named A, is migrated from PM_0 to PM_1 for the first time. In this migration, memory reusing does no occur but operations required for future memory reusing do occur. $\{PM_0, A\}$ in the figure represents the *generation table* of VM A associated with PM_0, that is, the set of all generations of A's memory pages stored in PM_0.

1. Page frame numbers (PFNs) of updated memory pages of the target VM are transferred from PM_0 to the generation server. The updated memory pages are detected during runtime of the VM using dirty page tracking as already described.
2. $\{PM_0, A\}$ is updated: Generations for the memory pages with the PFNs received in the previous step become 1 from 0.
3. Detection of memory pages to be transferred: $\{PM_0, A\}$ and $\{PM_1, A\}$ are compared and memory pages that have different generations in the two generation tables are transferred. In this case, all memory pages are regarded to have different

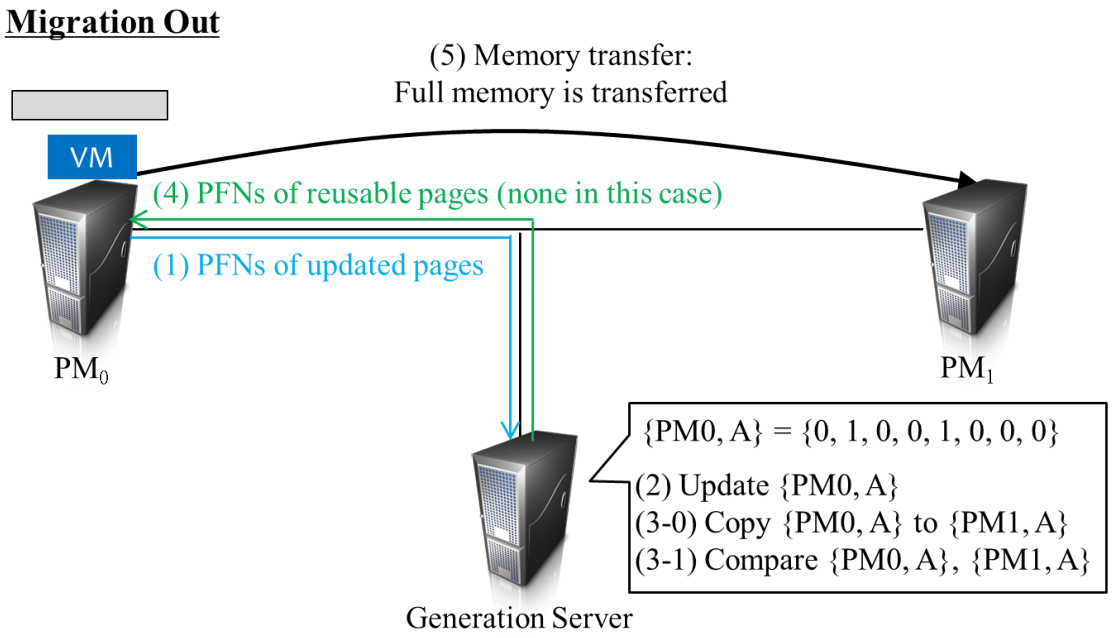


FIG. 3.6. First migration of a VM: The memory image is cached on the source PM. The generation table is updated and MiyakoDori knows later which pages can be reused.

generations because $\{PM_1, A\}$ does not exist yet (which means that this migration is the first time to PM_1).

4. PFNs of reusable memory pages are transferred to PM_0. In this case, all memory pages are not reusable thus nothing is transferred.
5. Live migration is kicked-off. The procedure is almost the same as well-known pre-copy live migration, but the only difference is that the memory image of VM A is not deleted after the migration.
6. After the migration is finished, $\{PM_1, A\}$ is created by copying $\{PM_0, A\}$.

During steps 1–4, VM A is suspended to maintain consistency between memory pages and generations. However, this procedure takes only a short period and does not incur large performance penalty to the VM.

Figure 3.7 illustrates the behavior of MiyakoDori when VM A migrates **back** from PM_1 to PM_0. In this migration, the memory image in PM_0 is reused to accelerate live migration.

1. PFNs of updated memory pages of the target VM are transferred from PM_1 to the generation server. The updated memory pages are detected during runtime of the VM using dirty page tracking as already described.

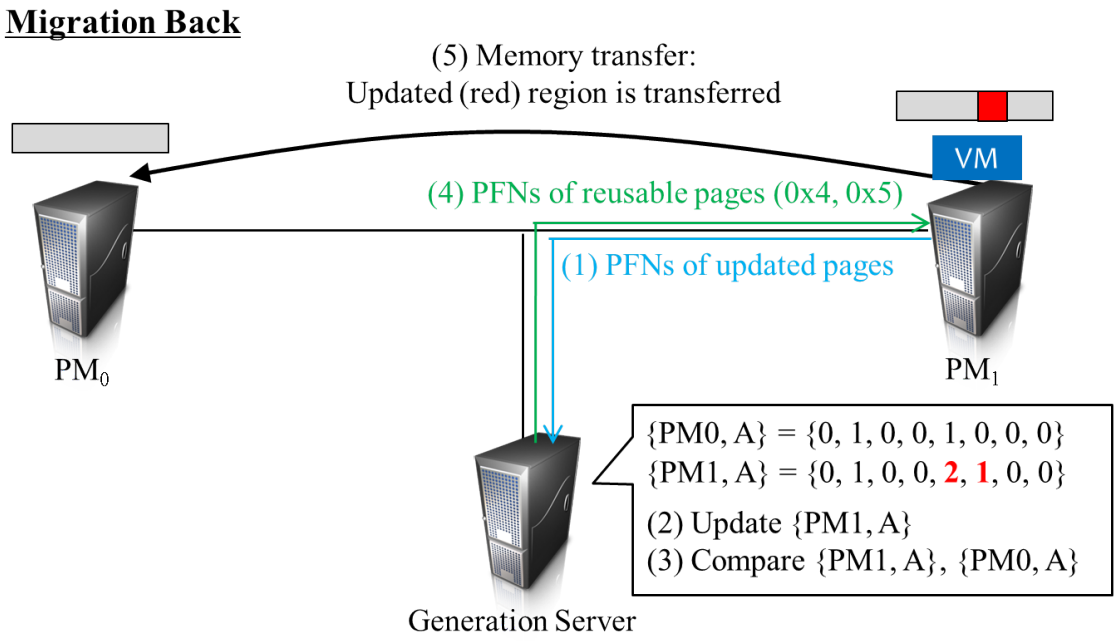


FIG. 3.7. Migration back to a PM on which the VM has once been executed: All the memory pages that have different generations in the source and the destination are copied. Other pages are reused and do not need to be copied.

2. $\{PM_1, A\}$ is updated: Generations for the memory pages with the PFNs received in the previous step become either to 2 (from 1) or to 1 (from 0).
3. Detection of memory pages to be transferred: $\{PM_1, A\}$ and $\{PM_0, A\}$ are compared and memory pages that have different generations in the two generation tables are transferred. In this case, the 4^{th} memory page (whose generation became 2 from 1) and the 5^{th} memory page (whose generation became 1 from 0) are to be transferred.
4. PFNs of reusable memory pages are transferred to PM_1. In this case, PFNs other than the 0x4 and 0x5 are transferred.
5. Live migration is kicked-off. In the 1^{st} phase of the pre-copy live migration, memory pages that are detected to be reusable are not transferred. The procedure is exactly the same as the pre-copy one after the 2^{nd} phase.
6. After the migration is finished, $\{PM_0, A\}$ is updated to be the same as $\{PM_1, A\}$.

3.4 EXPERIMENTS

3.4.1 METHODOLOGY

MiyakoDori is evaluated with application benchmarks. Total migration time and amount of transferred memory with/without the use of MiyakoDori are compared.

TABLE. 3.1. Hardware and Software used for Evaluating MiyakoDori

CPU	Intel Xeon X5460 (4 cores)
Memory	8 GB
Disk	250GB HDD
Network	GigaBit Ethernet NIC
OS	Debian GNU/Linux 6.0
kernel	Linux 2.6.32
KVM	kvm-kmod-2.6.38.6
QEMU	0.13.0

The specifications of used hardware and software are shown in Table 3.1. We set up a VM with 1GB of RAM and one virtual CPU. We used Ubuntu 10.10 server as a guest OS. Disk images are located on the generation server, accessible from the VM on any PM via Network File System (NFS).

TABLE. 3.2. Workloads used for Evaluating MiyakoDori

Name	Intensity	Detailed description
Busy Loop	CPU	Infinite busy loop
WebServer	Network, Disk IO	Read 1,024 static HTMLs (256 KB each) with 100 Mbps
Video	Memory, Disk IO	Transcode MPEG2 video into ogg theora format using vlc media player
TPC-C	CPU, Memory, Disk IO	DB access benchmark that simulates transactions of an online shop

Table 3.2 shows workload used for evaluating MiyakoDori. Detailed description of each workload is as follows:

BUSY LOOP This workload emulates pure CPU workloads. The vCPU of the VM is fully loaded by an infinite loop. It is expected to be the ideal case for MiyakoDori, where almost no memory pages are updated by the workload.

WEBSERVER This workload emulates web servers under high load. An Apache web server hosts 1,024 HTML with 256 KB each, and a load generator reads the files

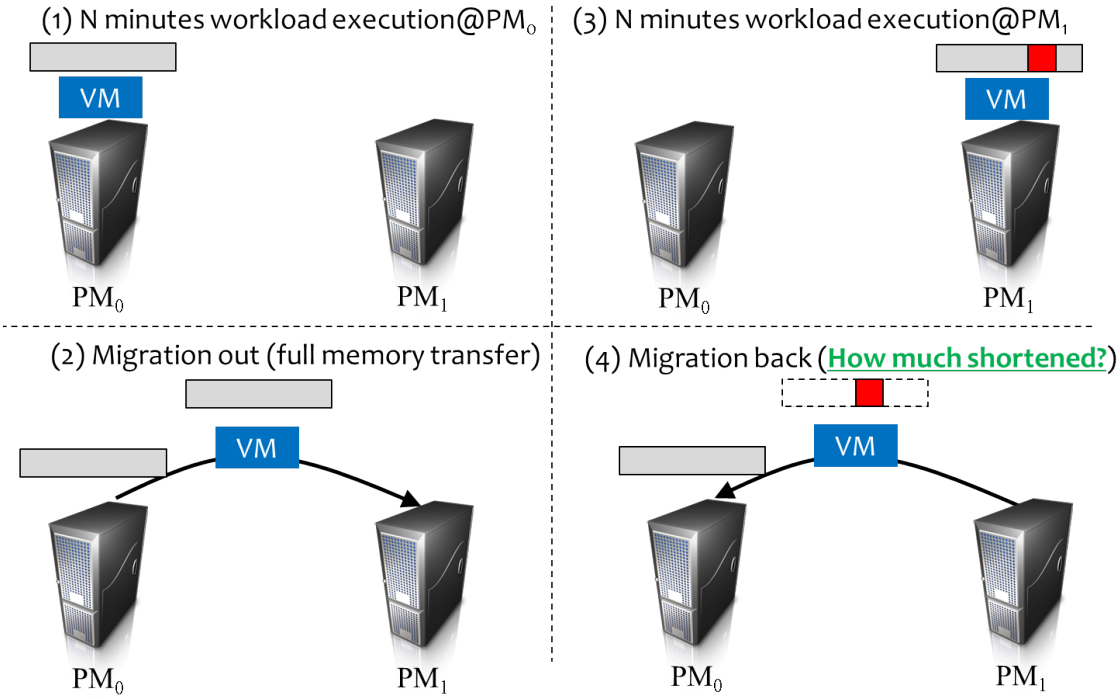


FIG. 3.8. Evaluation Methodology of MiyakoDori

with the speed of 100 Mbps. This workload is network and disk IO intensive.

VIDEO This workload emulates a media server which provides on demand video streaming in response to user requests. An MPEG2 video (extracted from a video DVD) is transcoded into Ogg Theora [64] format using VLC Media Player [65]. This workload is memory and disk IO intensive.

TPC-C This workload emulates a database server for web applications. TPC-C generates a typical read/write pattern for shopping web sites. This workload is CPU, memory and disk IO intensive. TPC-C refers a specification of a workload, thus as an implementation the one from Percona[*1] [66].

In order to compare total migration time and memory transfer in migration "back", the evaluations is done as follows (Figure 3.8):

0. Launch a VM on PM_0
1. Run a workload on the VM for N minutes
2. The VM migrates out from PM_0 to PM_1
3. Run the workload on the VM for **another** N minutes

[*1] http://www.percona.com/

4. The VM migrates back from PM_1 to PM_0. Total migration time and the amount of transferred memory in this migration are recorded and compared between MiyakoDori and non-modified QEMU/KVM.

3.4.2 TOTAL MIGRATION TIME

Total migration time achieved by MiyakoDori and taken by pre-copy live migration implemented un-modified QEMU/KVM are compared. The difference is that only the updated memory pages in the latter N minutes (Step 3 in Figure 3.8) are transferred with MiyakoDori, but all memory pages are transferred with the original pre-copy live migration. N was chosen from 5, 10, or 15 minutes.

Figures 3.9, 3.10, 3.11, and 3.12 show total migration time under Busy Loop, WebServer, Video, and TPC-C workload, respectively. The x-axis shows the interval between two migrations ($= N$ minutes) and the y-axis shows total migration time in each setting. Note that the maximum values of the y-axis are not the same in all the figures. Bars in right-hand side in the result pairs (labeled as "Original") describe the results with original pre-copy migration and ones in the left-hand side (labeled as Proposed) describe the results with MiyakoDori. In the following, the results and reasons that the results are yielded are discussed:

BUSY LOOP In this workload only a small number of memory pages were updated as expected. Thus the reduction ratio of total migration time is large. The results show that MiyakoDori is highly effective for pure CPU intensive workloads.

WEBSERVER This workload consumes more memory than Busy Loop but the reduction ratio of total migration time is larger than in Busy Loop. The reason is that reading files updates almost no memory pages, even if the total size of read files is large. The guest operating system kept all static HTML files in the page cache (file cache) during the experiments, and memory pages used for this file cache were not updated. The results show that MiyakoDori is again highly effective for a workload that has read-only large data.

VIDEO In this test workload, the largest amount of memory is consumed compared with the other three workloads, therefore total migration with the original pre-copy migration is longest among the all workloads. The difference between Video and WebServer workloads is that under Video workload large amount of data (e.g.

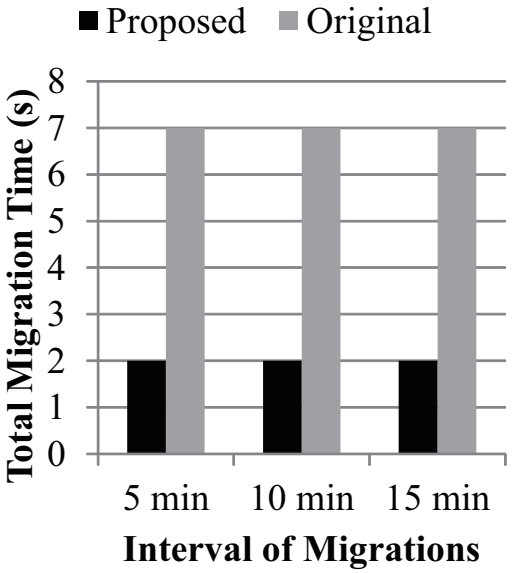


FIG. 3.9. Total Migration time under Busy Loop Workload

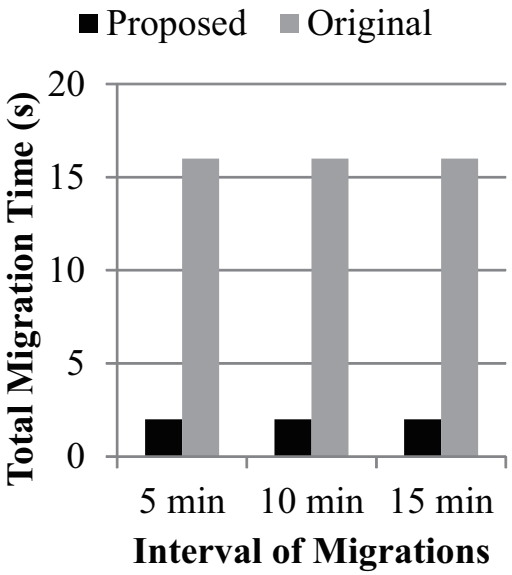


FIG. 3.10. Total Migration time under Web-Server Workload

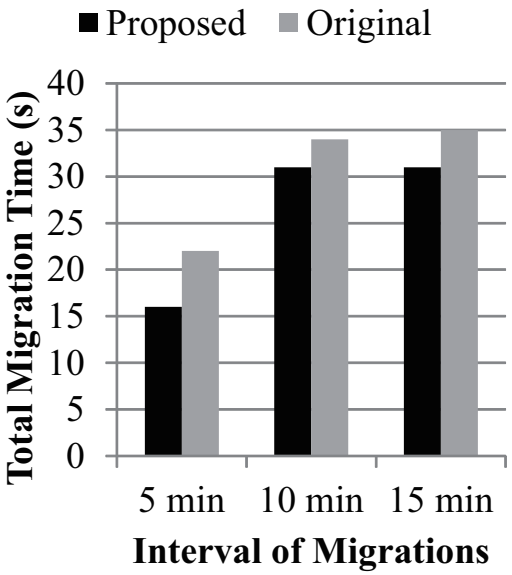


FIG. 3.11. Total Migration time under Video Workload

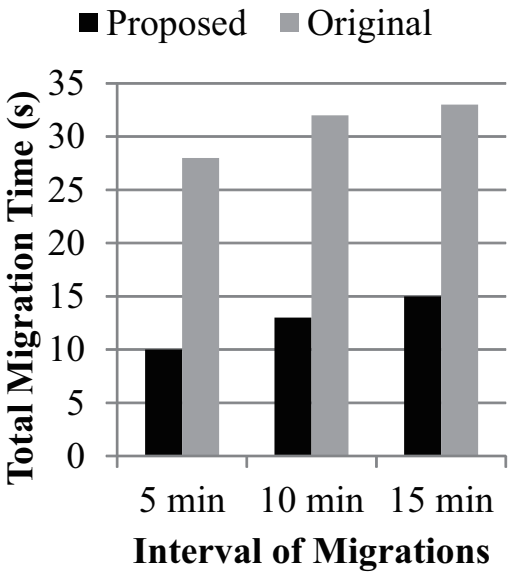


FIG. 3.12. Total Migration time under TPC-C Workload

converted video) is written as well as read. The operating system caches the written data in the memory for accelerating future access and buffering for the slow storage, resulting in many memory pages updated in this workload. The results show that MiyakoDori is not much effective for workloads that produce large output data in the memory. A small portion of memory reused in this workload is

used for purposes other than video data, such as text segment of loaded program or kernel data maintained by the guest OS.

TPC-C This workload is a compound workload of CPU, memory and disk IO and the memory access patter in also complex. Although the reduction ratio of total migration time slightly degrades, it is still high even when N is 15 minutes. The results indicate that MiyakoDori is beneficial for real compound workloads.

3.4.3 AMOUNT OF TRANSFERRED MEMORY

Figures 3.13, 3.14, 3.15, and 3.16 show the amount of transferred data under Busy Loop, WebServer, Video, and TPC-C workload, respectively. The x-axis shows the interval between two migrations ($= N$ minutes) and the y-axis shows the amount of transferred data in each setting. Note that the maximum values of the y-axis are not the same in all the figures.

An important point is that the figures look almost the same with the figures for total migration time. This is because in general total migration time of live migration is dominated by the time cost for transferring the memory data.

3.4.4 SUMMARY OF THE EVALUATION

This section summarizes the evaluation and draws important conclusions from it. The evaluations show that:

1. MiyakoDori can reduce the amount of transferred memory in a live migration with all of the four workloads.
2. Total migration time is reduced in proportion to the amount of transferred data in MiyakoDori.
3. Workloads that have small working set (e.g. Busy Loop) or large read-only data (e.g. Apache) are especially suitable for MiyakoDori.
4. Workloads that produce large data (e.g. Video) is not so much effective with MiyakoDori, although it can still reduce a small portion of total migration time and the amount of transferred data.

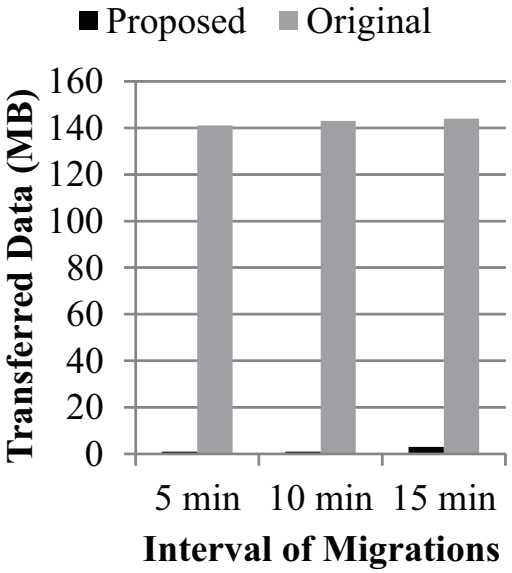


FIG. 3.13. Amount of Transferred Data under Busy Loop Workload

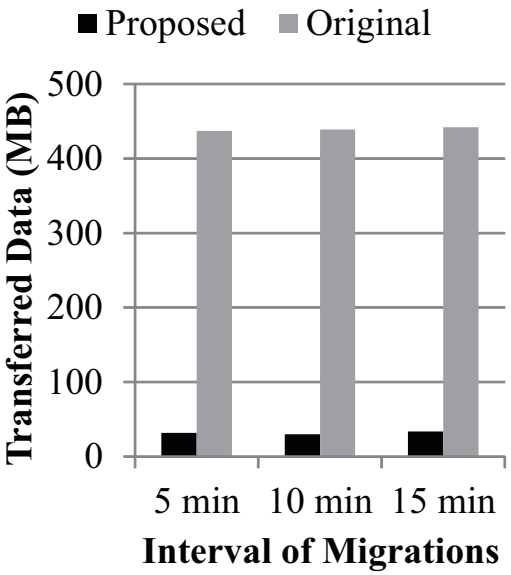


FIG. 3.14. Amount of Transferred Data under WebServer Workload

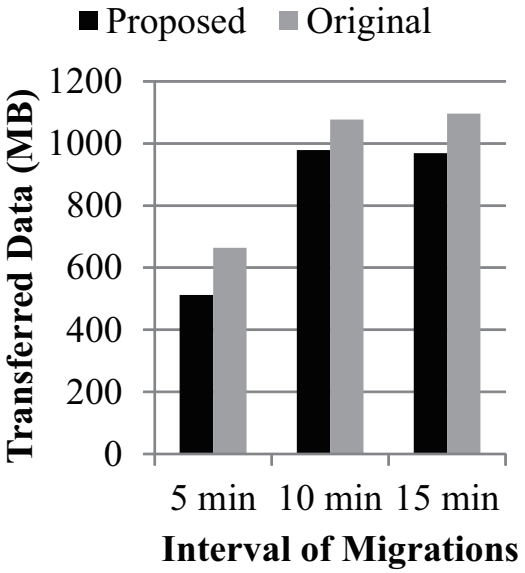


FIG. 3.15. Amount of Transferred Data under Video Workload

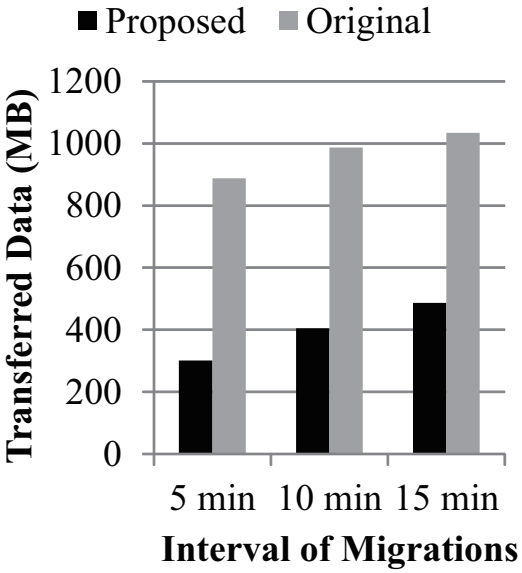


FIG. 3.16. Amount of Transferred Data under TPC-C Workload

3.5 DISCUSSION AND FUTURE WORK

3.5.1 OVERHEAD INCURRED BY MIYAKODORI

Even though MiyakoDori introduces two types of overhead into existing systems, we claim that in the real world this overhead is negligible for the following reasons:

MEMORY ACCESS SLOWDOWN

Setting the dirty bit of a memory page requires the page table to be checked in each memory access even if the virtual address of the page is cached on the Translation Look-aside Buffer. Therefore, dirty page tracking slows down memory accesses. In our experiments, dirty page tracking slows down only the first access to a page for approximately 20%, but no slowdown was observed in the completion time of the CG benchmark from NAS Parallel Benchmarks [67].

NETWORK OVERHEAD

MiyakoDori requires extra data to send updated page numbers to the generation server and to send reusable page numbers to the PMs. This overhead is very small because we only need 1 bit to represent whether a page is updated or not. Suppose that a VM has 4GB of memory, then we need to send only $4\,\text{GB} \div 4\,\text{KB/page} \times 1\,\text{bit/page} = 1\,\text{Mbits}$ for updated page numbers (the same calculation is also applied for reusable page numbers).

CPU DOWNTIME

The virtual CPU of a VM needs extra suspension by MiyakoDori, as described in Section 3.3. This suspension occurs during which the generation server receives PFNs of updated memory pages, comparing two generation tables, and sending PFNs of reusable memory pages. However, this suspension short enough because the data transferred during the suspension is enough small, and comparing two generation tables involves merely a single scan of two small arrays.

3.5.2 LIMITATION: SCALABILITY

MiyakoDori has two bottlenecks on its scalability:

GENERATION SERVER As described in Section 3.3.1, a client-server architecture where the generation server manages all information is chosen. This architecture is superior to a peer-to-peer architecture in the sense that it is easy to develop a new dynamic VM consolidation algorithm that aggressively exploits migrations back. However, it can be a scalable bottleneck when thousands of servers access the generation server in a short period.

LARGE MEMORY CACHES Memory caches stored in PMs can be large when there are bunch of VMs in the system or long term migration back (e.g. migrating back a

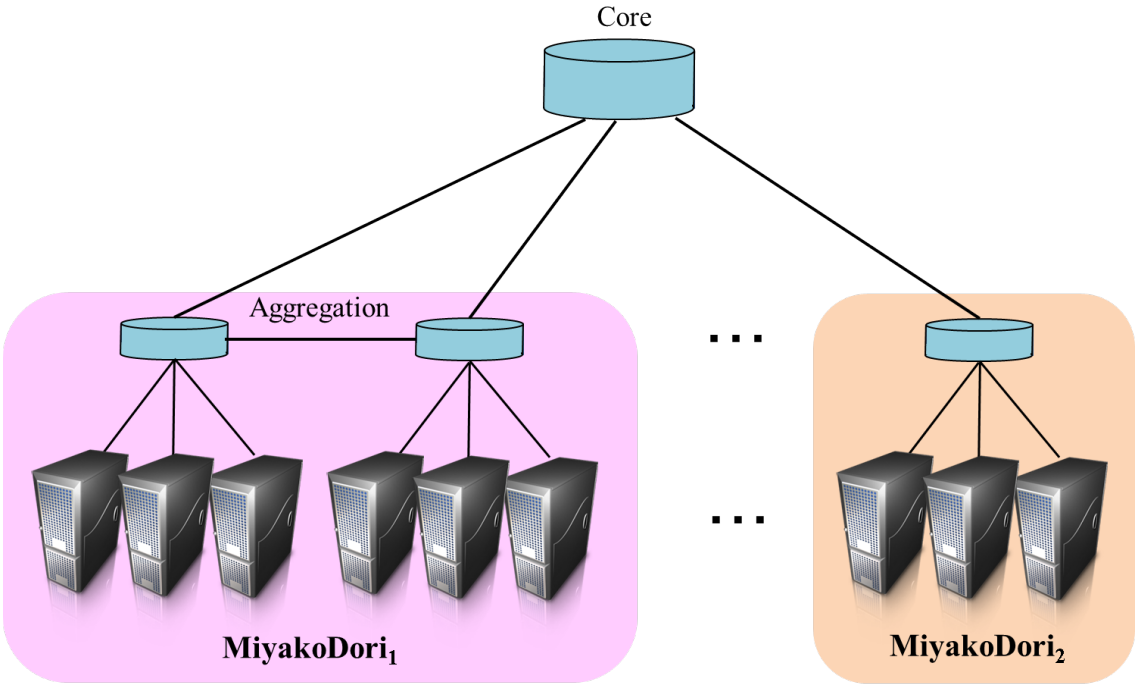


FIG. 3.17. Real usage of MiyakoDori. MiyakoDori is deployed for one rack or a group of several nearby racks. This usage does not weaken the applicability of MiyakoDori, because migrating VMs between faraway racks requires communications through core switches thus should be avoided.

week after migrating out) is required. Even with possible memory cache management mechanism (which is discussed in Section 3.5.3), this problem cannot be completely mitigated in a very large scale environment with thousands of PMs/VMs.

Due to these scalability bottlenecks, a practical use of MiyakoDori is to deploy it within one or several nearby racks in a data center. Figure 3.17 shows the use case. A MiyakoDori system is deployed across two neighboring racks in the leftside of the figure, and another MiyakoDori system is deployed for the right-most rack.

This usage limits the applicability of MiyakoDori, but the the limitation is not so severe because migrating VMs between faraway racks does not frequently occur in real environments. Communications between faraway racks go thorough core switches and can affect a large part of the whole data center. Therefore, placing highly related VMs in nearby racks/PMs is a well-known way to reduce network overhead in the data center [68]. This means that migrating a VM between faraway racks is a high-cost operation even compared to normal migration and is not desired.

3.5.3 FUTURE WORK

This section discusses future extending directions of MiyakoDori, namely memory image management and proactive memory reusing.

MEMORY IMAGE MANAGEMENT

Memory image management is an important factor of extending MiyakoDori. In the current implementation, all memory pages of all memory images are kept in the memory of the PMs. This may cause memory starvation if many memory intensive VMs are executed and the operating system of the PMs can swap the images out to external storages.

The easiest way to mitigate this issue is to use modern storage devices that provide fast read throughput. SSDs that are attached to an PIC-Express interface or IO-fusion provide several or even 10 times faster read throughput compared to traditional HDDs. However, this way cannot completely solve the issue, because under CPU-intensive workloads the number of reused memory pages is much larger than the number of non-reused memory pages. For example, suppose all memory images are swapped out to an external SSD, then in Figure 3.14 case nearly 400 MB must be swapped in from the SSD while 30 MB is transferred vie Ethernet.

A more effective but non-off-the-shelf way is to develop a memory image management algorithm/mechanism. For example, a page that is updated on a PM is no longer needed to be stored on any other PMs because that page is no longer reusable. Other types of unnecessary pages can be detected by further analysis of memory access pattern of the workload. A memory page that has strong possibility of update in a near future does not need to be cached. For example, analysis of memory access patterns of a workload indicates an assumed size of the working set of the workload [69]. Memory pages in the working set are highly possible to be updated in the near future, thus an aggressive decision can be not to store them in memory images at all.

PROACTIVE MEMORY REUSING

Proactive memory reusing means to predict destination PMs of future migration and to transfer memory image before an actual migration occurs. In proactive memory reusing, memory images can be reused even in the first migration of the VM to the destination PM. A similar technique has been proposed in the context of cluster computing and remote process migration.

3.6 RELATED WORK

Takahashi *et al.* [70, 71] propose a similar idea to accelerate storage migration, which is a technique to transfer the disk image of the migrated VM on a live migration. This thesis focuses on live migration used within a data center, thus storage migration is not required because disk images are located in shared filesystem such as NFS in practice. A technical difference is that they calculate hash values of disk blocks to detect which disk blocks are updated, while MiyakoDori uses dirty page tracking. This difference stems from a difference of performance requirements of two technologies. storage migration takes much more time than memory migration thus the time cost of calculating hash values is hidden behind the long transfer time. On the other hand, MiyakoDori can leverage dirty page tracking because it targets memory pages, whose updates are already detected by VMMs.

An important note is that even post-copy live migration and trace-and-replay based methods optimize the 2^{nd} phase. Post-copy live migration requires no repeated transfer of memory pages, thus it can be regarded as the extreme of optimizations on the 2^{nd} phase. However, post-copy live migration still needs to transfer all memory pages at least once as the 1^{st} phase of pre-copy live migration does.

3.7 SUMMARY OF THIS CHAPTER

In this chapter, MiyakoDori, an efficient live migration technique that is effective under aggressive VM relocation is proposed. MiyakoDori reduces the the amount of transferred memory by reusing non-updated memory pages when a VM migrates "back" to a PM where it has been executed before. The idea of migrating "back" is based on an assumption that live migration is aggressively used in a cloud data center, which is the goal of this thesis. Evaluations showed that the proposal reduces the amount of transferred memory and total migration time of live migration. Integrated evaluations of MiyakoDori and aggressive VM relocation are given in Chapter 5.

CHAPTER 4

PAGE CACHE TELEPORTATION

4.1 INTRODUCTION

Page cache is a widely-adapted mechanism to improve performance of disk IO operations. It is equipped in many modern operating systems such as Linux, Windows and BSDs. A VM running a workload that treats large data has large amount of page cache.

Transferring large amount of page cache prolongs live migration and results in longer PM active time under aggressive VM relocation than desired. However, existing methods that skip the transfer of page cache to shorten total migration time [72, 73] greatly degrade IO performance of target VMs and can violate service level agreements of clouds. Therefore, a novel technique that accelerate live migration of VMs with large page cache without large IO-performance degradation is required.

In this chapter, an advanced memory transfer mechanism for live migration of IO-intensive VMs is proposed. The method skips transferring the page cache to shorten total migration time while restoring it transparently from the guest OS via the SAN to prevent IO performance penalty. To start a migration, our mechanism collects the mapping information between page cache and disk blocks. During a migration, the source host skips transferring the page cache but transfers other memory content, while the destination host transfers the same data as the page cache from the disk blocks via the SAN. The technique shortens total migration time by simultaneously utilizing the SAN along with the general purpose network and achieves smaller IO performance penalty by transparently transferring the page cache rather than deleting it.

Experiments using WebServer, Postmark, and TPC-C workloads showed that our method greatly reduced total migration for various IO-intensive workloads. We also show that our system achieves the shortest migration time without manually tuning

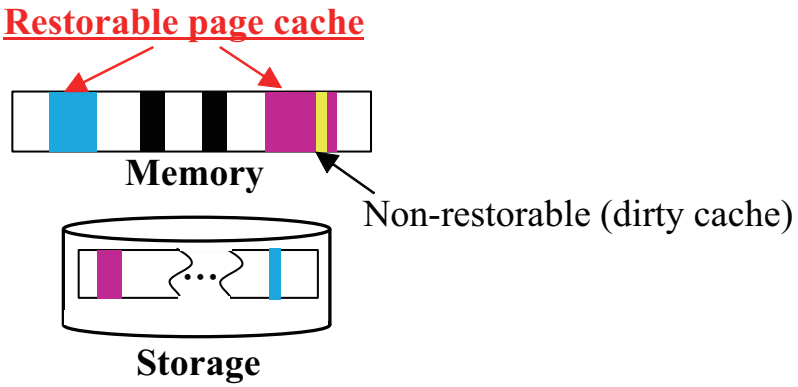


FIG. 4.1. Restorable Page Cache. Memory and storage have exactly the same data due to the page cache mechanism. Note that dirty cache (write cache that has not yet been flushed) is not restorable even though it is included in page cache.

parameters even when the amount of page cache or the location of the bottleneck are different in each workload. We also give mathematical analysis of our method to understand how our system works in various situations.

This chapter is structured as follows. Section 4.2 explains the page cache problem in more detail. Section 4.3 describes the core ideas in this chapter. Section 4.4 shows how the system works to reduce total migration time. Section 4.5 illustrates technical contributions. Section 4.6 explains the implementation details. Section 4.7 shows the evaluation results. Section 4.8 gives further discussions. Section 4.9 refers related work and Section 4.10 summarizes this chapter.

A part of this chapter has been published in refereed papers. The copyrights of the papers are hold by IEEE Computer Society.

1. Soramichi Akiyama *et al.* Fast Wide Area Live Migration with a Low Overhead Through Page Cache Teleportation, In *Proceedings of The 13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 76 - 82, 2013.
2. Soramichi Akiyama *et al.* Fast Live Migration with Small IO Performance Penalty by Exploiting SAN in Parallel, In *Proceedings of The 2014 IEEE 7th International Conference on Cloud Computing*, pages 40 - 47, 2014.

4.2 RESTORABLE PAGE CACHE

A VM running a workload with large data can have many memory pages identical to disk blocks due to *restorable page cache* because the operating system uses as many free

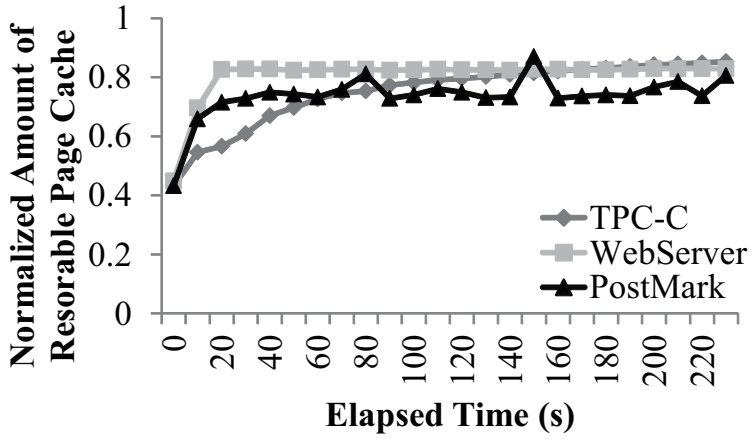


FIG. 4.2. Normalized Amount of Restorable Page Cache in the Memory under WebServer, Postmark, and TPC-C, Workloads. The values are normalized by being divided by total memory usage of the VM. More than 70% of the memory is occupied by restorable page cache for most of the time.

memory pages as possible for page cache. [*1] Page cache is an on-memory cache mechanism to hide the gap between the accessing speed of memory and storage and is implemented in many modern operating systems such as Linux, Windows and BSDs. When an IO operation is requested for a disk block, the read/written data is stored in the page cache to accelerate future requests for the same disk block. Restorable page cache refers memory pages whose data can be restored from the identical disk blocks even if it is deleted. Figure 4.1 illustrates restorable page cache. The same contents exist both in the memory and the storage (rectangles with the same color). Note that memory pages containing write-cache which has not yet been flushed are not restorable although they are included in page cache.

Figure 4.2 shows the amount of restorable page cache contained in the memory of a VM running workloads with large amount of data. The x-axis shows the elapsed time from the beginning of the workload, and the y-axis shows the normalized amount of restorable page cache. The values are normalized by being divided by the total memory usage of the VM. WebServer is a workload that simulates a web server under high load. A load generator outside of the VM accesses the web contents with high access rate. Postmark is a workload to measure IO performance of small files to estimate server performance for web and mail services. TPC-C is a workload that simulates the typical database access pattern for an online shopping web site. The detailed descriptions of

[*1] Page cache and restorable page cache themselves are not unique phenomena of virtual machines, but this thesis focuses on page cache inside VM memory.

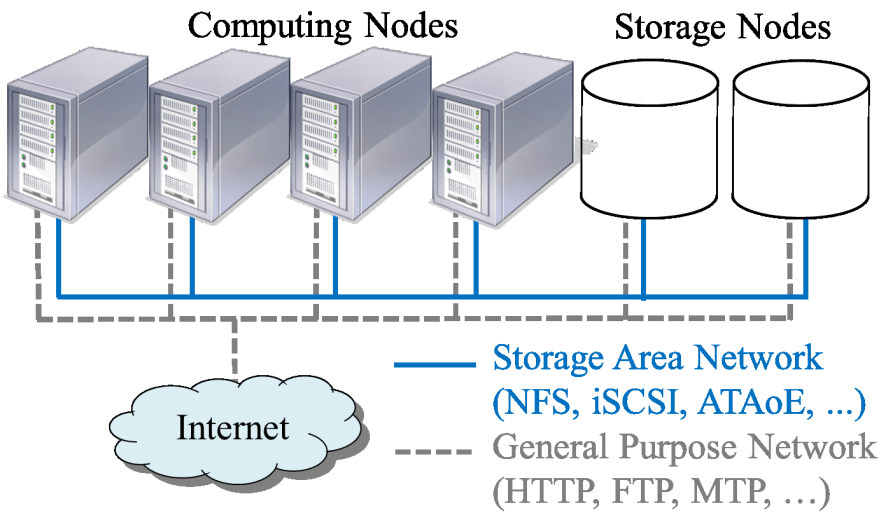


FIG. 4.3. Network Architecture of Cloud Data Center. Storage nodes are connected with a designated storage area network (SAN).

WebServer, Postmark, and TPC-C workloads are given in Section 4.7. The figure shows that in all three workloads many memory pages (more than 70% of all pages) are identical to disk blocks due to the restorable page cache most of the time during the workload execution. The values are small in the beginning but this does not weaken our claim because the periods are warming-up phases of the workloads.

Existing live migration mechanisms lack an efficient way to transfer the large page cache described in the previous two paragraphs. Large page cache increases the amount of memory to be transferred in the 1^{st} phase of a migration. Thus, live migration researches focusing on the 2^{nd} phase (how to mitigate iterative copies) can do nothing on this issue. Some studies [72, 73, 74] focus on reducing amount of transferred page cache. Differences of our work and the existing studies are discussed in Section 5.9 (Related Work) in detail.

4.3 CORE IDEAS

4.3.1 NETWORK ARCHITECTURE OF CLOUD DATA CENTER

An important characteristic of a data center networking that we exploit is explained here. Figure 4.3 illustrates a simplified view of the network architecture of a typical data center. The main point is that storage nodes are connected with a designated SAN along with a general purpose network. For example, CISCO suggests a data center networking archi-

tecture that includes Storage Networking and Business Continuance Networking [75]. Nodes might have another link for management purposes. Descriptions of each network are as follows:

Storage Area Network (SAN): It is used to communicate with the storage nodes in the data center. A shared filesystem is built on top of this link and IO requests and data from/to the storage nodes go through this link. An important notice is that we do not assume IP-capability of this link. The shared filesystem can be built with any networking such as Ethernet or Infiniband without IP.

General Purpose Network (GPN): It is used to deal with any network packets other than storage-related ones. For example, HTTP requests sent from the Internet or sent between services running in the data center go through this link.

4.3.2 TRANSFERRING PAGE CACHE FROM STORAGE

The core idea of our work is that restorable page cache can be transferred from the disk image using the SAN, rather than the general purpose network. This is technically possible because disk blocks identical to restorable page exist in the disk image. Live migration of a VM with large restorable page cache is accelerated by conducting two transfers in parallel:

1. Transferring restorable page cache via the SAN.
2. Transferring other normal data via the general purpose network.

Figure 4.4 shows how live migration is accelerated by our proposal. The source and the destination hosts are connected by the GPN, and the storage node is connected with them by the SAN. The rectangles in the VM memory and the disk image indicate memory pages and disk blocks. The black that is included only in the memory is the normal data page. The grey ones that are included both in the memory and the disk image contain restorable page cache. On a migration, the normal data page is transferred via the general purpose network, at the same time as restorable page cache is transferred via the SAN.

4.3.3 DIVIDING PAGE CACHE

Transferring all the restorable page cache via the SAN is not enough to mitigate the problem we tackle. Suppose the memory usage of a VM to be migrated is 1 GB and 90% of it is filled with restorable page cache (as the case in WebServer workload). In this case,

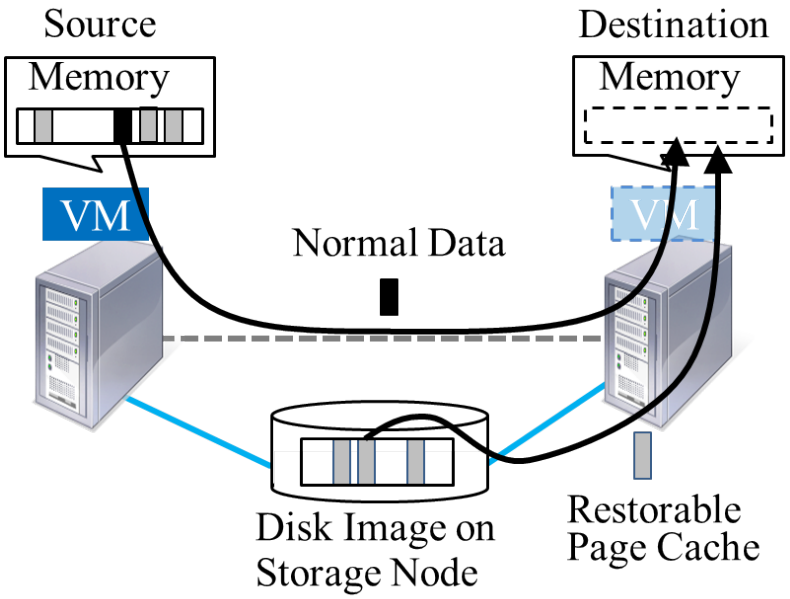


FIG. 4.4. Simple View of How Live Migration is Accelerated by Our Proposal. Rectangles in the VM memory and the disk image indicates memory pages and disk blocks. The black one is a normal data page, and the gray ones that appear both in the memory and the disk image contain restorable page cache. Restorable page cache is transferred from the storage node via the SAN to use two networks in parallel.

the amount of memory to be transferred via the SAN is 900 MB while the one to be transferred via the GPN is 100 MB, resulting in a new bottleneck at the SAN.

In order to mitigate this issue, we propose *adaptive page cache transfer*. This method transfers a portion of restorable page cache via the GPN but not via the SAN, to balance the load of the two network links. "Adaptive" means that the method does not require to manually specify how much portion of restorable page should be transferred via the GPN. This adaptiveness is highly important in a cloud data center, where network usage by other VMs are unpredictable/uncontrollable. For example in the case in the previous paragraph, dividing the restorable page cache into 500 MB and 400 MB results in 500 MB transfers both via the GPN and the SAN. However, the SAN might be more congested than the GPN because other VMs sharing the same storage PM is executing heavy IO, or the other way around because web servers in other VMs are under high volume of requests. Our adaptive page cache transfer mitigates this issue and the details are further described in Section 4.5.4.

4.4 PROPOSED SYSTEM

4.4.1 DESIGN OVERVIEW

We propose an advanced memory transfer mechanism that exploits the core ideas explained in the previous section. The design criteria of the system is as follows:

1. The system must have as little performance interference as possible to the target VM because dynamic VM placement is a background operation which users of the VMs do not want to have affected from.
2. IP-capability is not assumed to the SAN because in real data centers it can be non IP networking such as Infiniband without IP.
3. Implementation must be easy in terms of guest OS dependency because types of guest OS in a cloud has much variety.

4.4.2 MIGRATION PROCEDURE WITH OUR MECHANISM

The procedure of a live migration with our mechanism is illustrated in Figure 4.5. A VM is being migrated from the source (top-right) to the destination (bottom-left). The VM on the destination is not running yet thus it is grayed out. The dotted arrows show commands, the dashed ones show metadata to realize our method, and the solid ones show transfers of memory data. Detailed descriptions are as follows:

1. The source VMM receives a request to execute a migration. The requests is passed to our user-space program inside the VM. Then the user-space program requests our kernel module to fetch the page frame numbers (PFNs) of the restorable page cache and the block numbers of the identical disk blocks.
2. The kernel module detects the PFNs of restorable page cache and the identical disk blocks. They are sent to the source and the destination VMMs with the help of the user-space program.
3. The memory transfers are kicked-off. The SAN is exploited to accelerate live migration.
   (a) Restoring Thread in the destination VMM transfers restorable page cache from the disk image via the SAN.

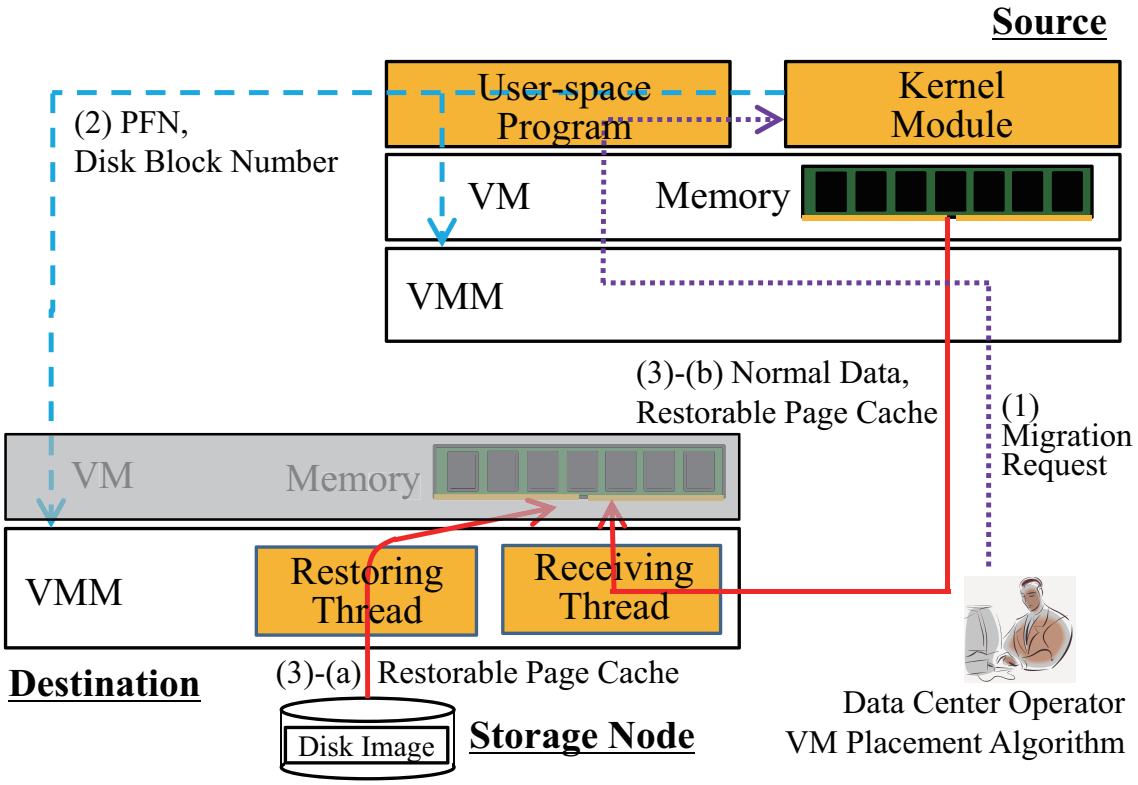


FIG. 4.5. Procedure of Live Migration with Our Mechanism. Each arrow shows a data flow to achieve the proposal. (1) The source VMM receives a migration request and it passes the request to kernel module inside the guest OS with a help of the user-space program. (2) The user-space program sends retrieved PFNs and disk block numbers of restorable page cache to the source and the destination VMMs. (3) Normal data is transferred via the GPN, and restorable page cache is transferred both via the SAN and the GPN.

(b) Receiving Thread in the destination VMM receives normal data pages from the source VMM via the GPN. Once the number of normal data pages to be transferred becomes sufficiently small, the thread starts receiving restorable page cache via the GPN (adaptive page cache transfer).

4. Memory pages updated during above steps are transferred again via the general purpose network.
5. Once the amount of remaining memory becomes sufficiently small and all the restorable page cache is copied, the execution host of the VM is switched.

4.5 TECHNICAL DETAILS

4.5.1 OVERVIEW

The technical contributions of this paper is described in this section. The challenges are how to achieve the parallel transfers of normal data and restorable page cache, and how to achieve adaptive page cache transfer.

Memory Consistency: Transfers of normal memory and restorable page cache are done while the VM keeps running. This causes updates to restorable page during a migration. Updated restorable page cache must be detected and transferred as normal memory pages because there is not guarantee that the pages are flushed into the disk image.

Writing Algorithm: It is possible that one memory page is written by the two threads. The writing algorithm must be carefully designed to deal with this case.

Adaptive Page Cache Transfer: Receiving Thread and Restoring Thread must cooperate in adaptive page cache transfer to select appropriate network to transfer each memory page.

4.5.2 MEMORY CONSISTENCY

The memory consistency of the VM during a migration must be carefully dealt with. The issue is that a memory page containing restorable page cache can be updated and can turn into non-restorable. Suppose a memory page is updated during a migration after it has been detected as restorable page cache. In this case, the updated and latest data must be transferred via the general purpose network because there is no guarantee that the updated data has been flushed into the disk. There are two cases in which a memory page used for the restorable page cache is updated: the cached data contained in the page is updated or the guest OS frees the page and use it for another purpose than page cache because of memory pressure.

We solve this issue with the dirty page tracking functionality of the VMM. The x86 architecture has a dirty bit for each memory page that is set when the page is updated. The VMM provides a functionality to read the dirty bits from the software level. Dirty page tracking is enabled at the source host when a migration starts and all the memory writes after that are tracked. A memory page updated during the tracking is transferred via the general purpose network, even if the memory page was restorable when the

Algorithm 1 Receiving Thread

```
repeat
  ReceiveMemoryPages(Buffer)
  i ← AddressOfFirstPage(Buffer)
  repeat
    TryLock(i)
    if Lock is acquired then
      ReceivedFlag[i] ← 1
      MemoryOfVM[i] ← GetMemoryData(Buffer, i)
      ReleaseLock(i)
    end if
    i ← AddressOfNextPage(Buffer, i)
  until Buffer is all flushed
until Number of received pages becomes sufficiently small
```

Algorithm 2 Restoring Thread

```
for i ∈ AddressesOfResorablePageCache do
  TryLock(i)
  if Lock is acquired then
    if ReceivedFlag[i] == 0 then
      b ← IdenticalDiskBlockNumber(i)
      MemoryOfVM[i] ← DiskImage[b]
    end if
    ReleaseLock(i)
  end if
  /* Skip the page if the lock cannot be acquired */
end for
```

kernel module detected restorable page cache.

4.5.3 WRITING ALGORITHM

The simultaneous transfers are implemented by two threads and a buffer in the destination VMM. Receiving Thread receives and buffers normal memory pages transferred via the general purpose network. The buffer is flushed into the VM memory once it is filled. Restoring Thread fetches the restorable page cache from the storage node via the SAN and copies the data into the VM memory without buffering.

A lock mechanism is used to deal with two simultaneously transfers because the two threads can write to the same memory page. This happens when a memory page on the source host is updated after it has been detected to be restorable page cache as described in Section 4.5.2. The destination VMM has a *received flag* and a mutex for each memory page. The memory footprint during a migration by the flags and mutex is enough small because a flag is 1 byte and a mutex is 40 bytes (in Linux pthread implementation) and

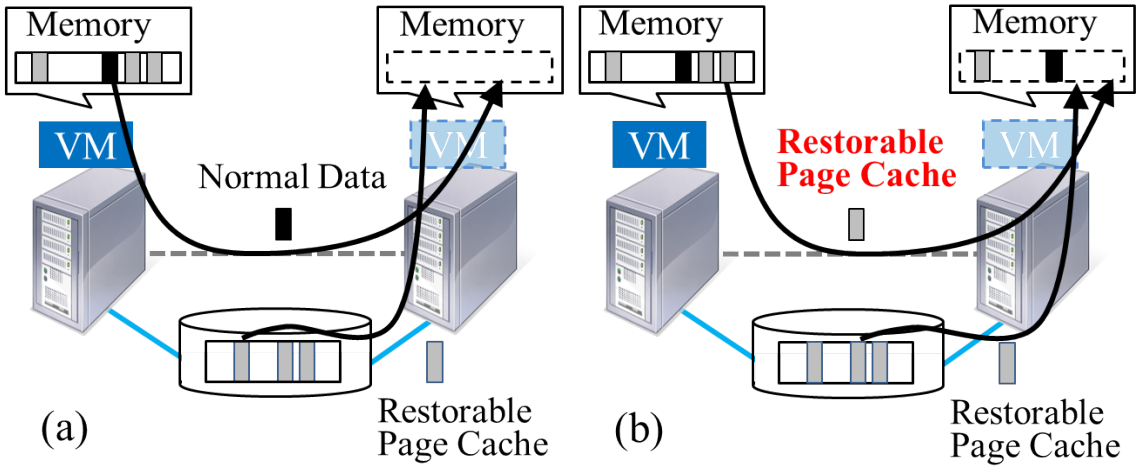


FIG. 4.6. Adaptive Page Cache Transfer. Example with 1 memory page for normal data and 3 memory pages for restorable page cache. (a) First, the normal data page (black rectangle) is transferred via the general purpose network and a restorable page cache page is transferred via the SAN. (b) Second, one of the remaining two restorable page cache pages is transferred via the general purpose network while the another one is transferred via the SAN.

the sum is 4 % of the size of a memory page. The flags and mutex are no longer required after a migration thus there is no extra memory required during non-migration time. The working algorithms of Receiving thread and Restoring thread are as follows:

Receiving Thread: It tries to acquire the lock for a memory page before writing to the page. If the lock cannot be acquired it processes the next memory page in the buffer (or the first memory page if it reaches to the end of the buffer). If the lock is acquired, it enables the received flag for the page and copies the transferred data into the VM memory, and then releases the lock. The buffer prevents the receiving thread from being blocked upon a lock conflict.

Restoring Thread: It also tries to acquire the lock for the page before writing to a memory page. If the lock cannot be acquired, it skips processing the page because a lock conflict means that the updated and latest data is being written to the memory page by the receiving thread. If the lock is acquired and the received flag is disabled, it copies the identical disk block to the memory page, and then releases the lock.

4.5.4 ADAPTIVE PAGE CACHE TRANSFER

We propose adaptive page cache transfer to balance the load of the two networks. This imbalance happens when transferring normal data pages finish earlier than transferring restorable page cache, because restorable page cache dominates the memory usage in

our target cases. As described in Section 4.3.3, manually mitigating this imbalance is infeasible thus our adaptive page cache transfer is essential.

The adaptive page cache transfer has three steps:

1. Our system **tries** to transfer all restorable page cache via the SAN, and all other normal memory pages via the GPN.
2. Transferring of normal data pages ends first before transferring restorable page cache.
3. The remaining of restorable page cache is transferred using the two networks in parallel. The transfer via the normal network is done from memory to memory, and the one via the SAN is done from storage to memory.

Figure 4.6 illustrates the adaptive page cache transfer. The figure shows a simple example where only 1 memory page contains normal data (black rectangle) and 3 memory pages (grey rectangles) contain restorable page cache. In Figure 4.6(a), the normal data page is transferred via the GPN and a restorable page cache page is transferred via the SAN. Once transferring the normal data is finished, in Figure 4.6(b), a restorable page cache page is transferred vie the GPN along with another one being transferred via the SAN.

In the step (3), the source host and the destination host must tell each other which memory pages are already transferred. This is because pages transferred via the GPN are "pushed" from the source host while ones transferred via the SAN are "pulled" from the destination host in order not to make any change to the storage hosts.

This mechanism is implemented without any extra data structures by utilizing the received flag (described in Section 4.5.2) and a characteristic of underlying storage. Figure 4.7 describes the algorithm and it works as follows:

3.1) The pairs of PFNs and disk block numbers containing restorable page cache are sorted in the order of the block numbers.

3.2) At $t = t_0$, the restoring thread is transferring restorable page cache in the ascending order of the disk block numbers, and the receiving thread is transferring it in the descending order of the disk block numbers.

3.3) At $t = t_1$, both threads reach to the same memory page and all memory pages that contain restorable page cache are transferred to the destination host.

3.4) Between $t = t_1$ and $t = t_2$, the restoring thread skips transferring all memory pages

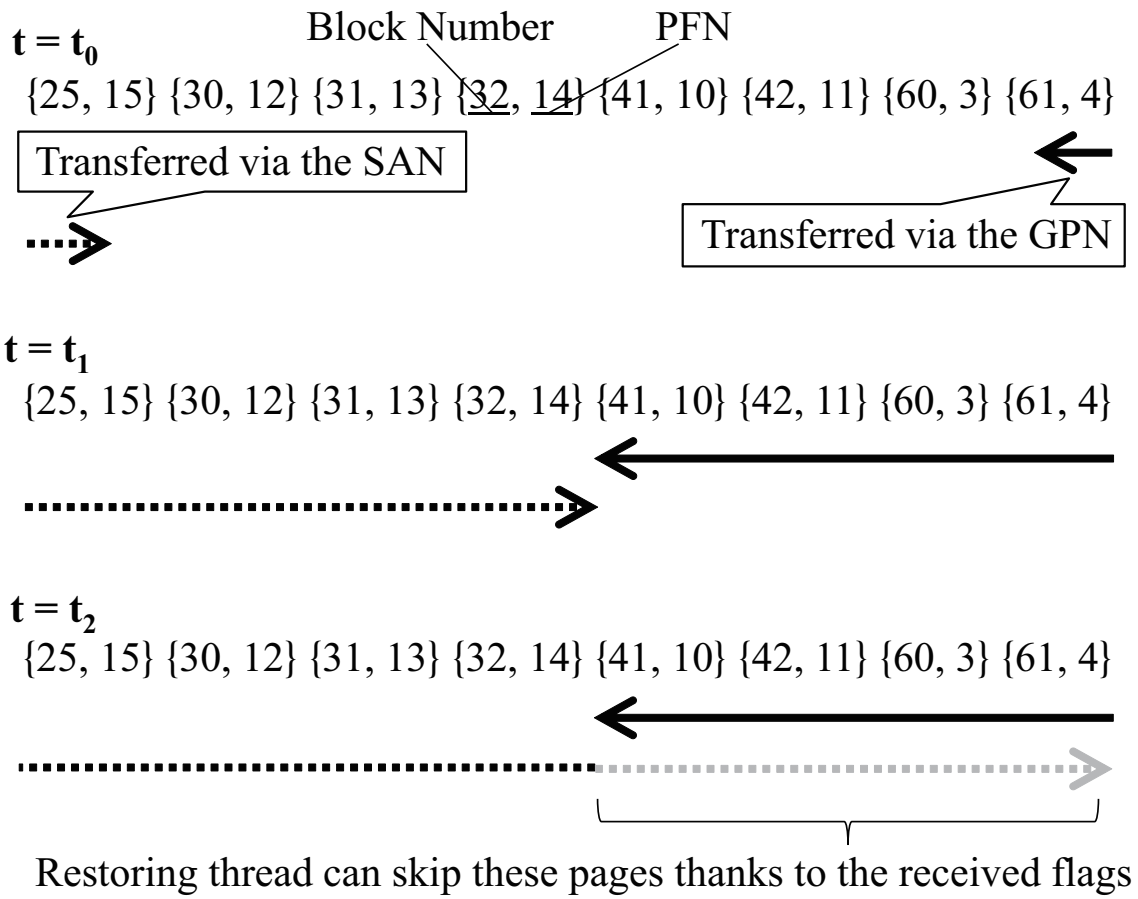


FIG. 4.7. Adaptive Page Cache Transfer Algorithm. Pairs of PFNs and disk block numbers to be transferred are sorted in the order of disk block numbers. **t=t**$_0$: Restoring thread receives restorable page cache via the SAN in the ascending order, while Receiving thread receives it via the GPN in the descending order. **t=t**$_1$: The two transfers reach up to the same memory page. **t=t**$_2$: Restoring thread can skip already transferred pages thanks to the received flag (Section 4.5.3), and the destination VMM finds that all memory pages have been transferred.

whose corresponding disk block numbers are more than 32, because the received flags for them are on.

3.5) At $t = t_2$, the restoring thread reaches to the end of the list, and the destination VMM notifies the source VMM of it. While the restoring thread is skipping already transferred pages, some memory pages ({14, 32} in Figure 4.7) are transferred again by the receiving thread. However, the number of memory pages that are transferred twice is negligible because skipping already transferred pages take small amount of time (it only requires to acquire open locks and check if the flags are on).

4.6 IMPLEMENTATION

4.6.1 COMPONENTS

Our system is implemented with three components. The details of each component are described in the following subsections.

Kernel module: It detects the PFNs of the memory pages containing restorable page cache and the block numbers of the identical disk blocks to the memory pages.

User-space program: It sends the PFNs and the disk block numbers to the modified VMMs.

Modified VMM: It transfers the restorable page cache via the SAN, and at the same time transfers the other memory pages via the general purpose network.

4.6.2 DETECTING RESTORABLE PAGE CACHE WITH KERNEL MODULE

Our kernel module detects the PFNs of the memory pages containing restorable page cache and the block numbers of the identical disk blocks to the restorable page cache. It utilizes OS dependent kernel functions and data structures to easily detect them. Our current implementation requires Linux guest, but we believe it is easy to implement it for other guest OS (Windows provides similar kernel functions to the ones we use in Linux). The size of the module is 155 KB only and it takes less than a second to detect restorable page cache from 1 GB of memory and 20 GB of disk.

The use of kernel functions and data structures greatly reduces the implementation cost to detect restorable page cache. In Linux, `pfn_to_page` kernel function takes an integer as the parameter and returns `struct page` kernel data of a memory page whose PFN is the integer. If the page contains page cache, the `struct page` includes a flag indicating whether the page is flushed back to the disk, which means this page is restorable. The disk block number that has the identical data to the page is retrieved by passing the `struct page` to another function `bmap`.

4.6.3 USER-SPACE PROGRAM

Our user-space program works as a broker between the VMM and the kernel module. When a migration is invoked, the VMM in the source host sends a request to get PFNs of restorable page cache and the identical disk block numbers. The user-space program

receives the request and invokes the `ioctl` operation of the kernel module. Once it gets the PFNs and the identical disk block numbers from the module, it sends them to the VMMs in the source and destination hosts.

The mapping information sent from the user-space program to the VMMs is an array of disk block numbers indexed by memory page numbers, that is, the n^{th} disk block number in the array describes information about the n^{th} memory page of the VM. A non-zero number represents the disk block number containing the identical data to the memory page. A zero in an array means that the memory page is not restorable. For example, an array $\{1234, 0, 10, 0, ...\}$ means that the 1234^{th} disk block has the identical data to the 1^{st} memory page and the the 10^{th} disk block has the identical data to the 3^{rd} memory page, while the 2^{nd} and 4^{th} memory pages are not restorable, and so forth. The size of an array is given by $8 \times \mathrm{NumberOfMemoryPages}$ because a disk block number is represented with an 8-byte unsigned integer in modern Linux. If a VM has 4 GB of memory, the size of the mapping information of this VM is: $8 \times (4\ \mathrm{GB} \div 4\ \mathrm{KB/page}) = 8\ \mathrm{MB}$.

4.6.4 MODIFIED VMM

We modify QEMU/KVM to add three functionalities. First, it has a new migration command whose arguments are the IP address of the VM, the IP address of the destination host, and the IP address of the target VM to be migrated. Second, it communicates with the user-space program inside the target VM to fetch the PFNs of restorable page cache and the identical disk block numbers before it start transferring the VM memory. Third it invokes two threads to simultaneously receives the restorable page cache and the other memory pages at the destination.

Retrieving the PFNs of restorable page cache and the identical disk block numbers uses our kernel module and our user-space program installed inside the VM. This requires users of VMs to install them, but we believe this requirement is lightweight because of two reasons. First, the kernel module and the user-space program are simple enough (both have approximately 200 lines of code in C) so that the admins can verify our modules are innocent. Second, our method helps not only cloud providers but also data center users. Fast live migration achieves lower energy consumption (resulting in lower pricing) and faster load balancing.

Receiving thread and Restoring Thread are invoked to achieve simultaneous trans-

TABLE. 4.1. Evaluation Environments

	Host	Guest
CPU	Intel Xeon X5460	1 core vCPU
Memory	8 GB	3 GB
Storage	256 GB HDD (read: 90 MB/s)	20 GB (raw disk image)
Network	1 Gbps NIC $\times$ 3	1 Gbps vNIC (bridged)
OS	Debian GNU/Linux 6.0.5 (Linux 2.6.32)	
VMM	QEMU 0.13.0, KVM 2.6.32	

fers. Receiving thread receives normal memory pages via the general purpose network, and Restoring thread transfers restorable page cache via the SAN. Restoring Thread uses normal read/write system calls to fetch the restorable page cache, therefore our implementation does not require any change to the underlying network settings. Normal read/write are automatically rerouted by the underlying filesystem because disk images are on a shared filesystem (such as NFS, ATAoE, and iSCSI) as normally done in cloud data centers. This means that our mechanism can be applied even if the target data center uses Infiniband for the SAN while using IP for the general purpose. On the other hand, merely bonding the SAN and the general purpose network for faster migration requires IP-capability of the SAN and changes to existing data center network settings.

4.7 EVALUATION

4.7.1 METHODOLOGY

Our evaluation consists of three parts and the metric each part measures is as follows:

1. Total migration time under various workloads, with all our proposals enabled.
2. Total migration time under various workloads, without the adaptive page cache transfer but with a pre-defined parameter R that specifies how much portion of restorable page cache is transferred via the SAN.
3. IO performance penalty to the VM after a migration.

The 1^{st} part shows that our proposal successfully shortens total migration time. The 2^{nd} part shows that the adaptive page cache transfer automatically yields the optimal results by comparing with manually tuned cases. The 3^{rd} part shows that our proposal causes

negligible IO performance penalty to the target VM after a migration because page cache of the VM is fully warmed up by our method.

The evaluations environments are shown in Table 4.1. The physical hosts have three 1 Gbps network interface cards (NICs), two of which are used for the SAN and the general purpose network. The read throughput of the storages are measured using bonnie++ in the host OSes. The VMM is composed of QEMU 0.13.0 and KVM 2.6.32. The KVM is the default version of the host OS. Each measurement uses three servers: two computing nodes and a storage node shared across the cluster via Network File System (NFS). A VM running a workload is migrated from a computing node to another computing node. The disk image of the VM is stored in the storage node. The read/write block size of NFS is tuned to 8 KB because it achieved the best throughput in our environment. [*2]

The evaluation is conducted with three workloads: WebServer, TPC-C and Postmark.

WebServer simulates a web server under high load. Apache web server has static files. The number of files is 10,000 and the size of each file is 300 KB. A load generator, httperf, fetches the files with the speed of 50 files/s. The load generator runs on a designated host (not the same neither as source nor destination) and accesses the files via the third NIC to avoid interference to the migration process. The migration is executed 250 seconds after the workload started, where all the files has been cached in the page cache.

Postmark [77] is a benchmark that measures IO performance of small and short-lived files to simulate load of mail, net news, or web servers. A survey on file system benchmarking tools [78] reports that Postmark is the 3rd most popular in research during 2009–2010. We set up the parameters of Postmark as follows: repeat 80,000 transactions in the speed of at most 500 per second with 1MB–5MB files and 4KB of read/write buffers.

TPC-C [79] is a benchmark that measures the performance of a database system. It generates database access patterns that simulates an online shopping web site. The total size of content of the database is 1.9 GB. The migration is executed 270 seconds after the the workload started, where the warming up phase of TPC-C has been finished.

4.7.2 TOTAL MIGRATION TIME

Figure 4.8(a), 4.8(b) and 4.8(c) show total migration time under WebServer, Postmark, and TPC-C workloads, respectively. The light-colored bars indicated "original" are the results with un-modified live migration implemented in QEMU 0.13.0, and the dark-

[*2] In our previous paper [76], the block size was 4KB.

colored bars indicated "proposed" are the results with our proposal. Each value is calculated by averaging over 10 runs.

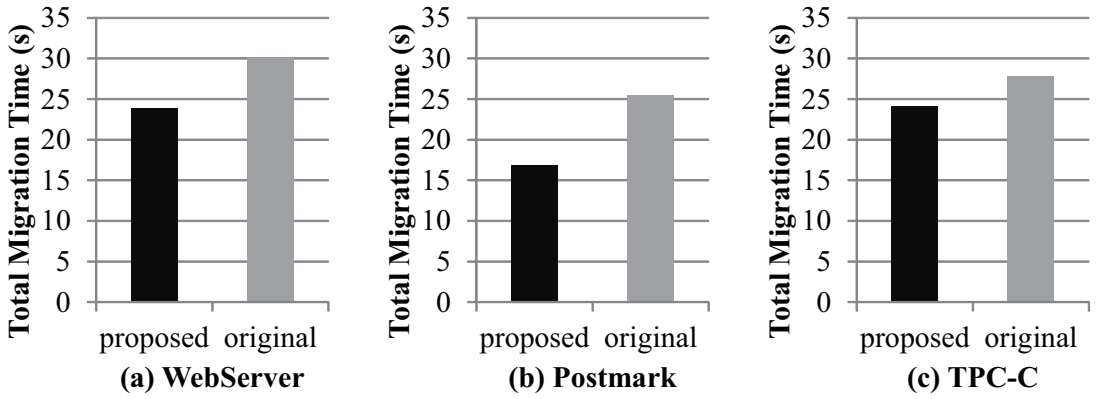


FIG. 4.8. Total Migration Time with and without our proposal under (a) WebServer (b) Postmark (c) TPC-C workloads.

TABLE. 4.2. Reduction Ratio of Total Migration Time with Our Proposal.

Workload	Reduction Ratio
WebServer	20.3 %
Postmark	33.9 %
TPC-C	13.3 %

Table 4.2 shows the reduction ratios of total migration time achieved by our proposal, against total migration time achieved by non-modified QEMU. The reduction ratios of the total migration time are 33.9% under Postmark workload, 20.3% under WebServer workload, and 13.3% under TPC-C workload. These results show that our proposal efficiently reduces the total migration time under various IO-intensive workloads.

There are large difference between the reduction ratios depending on each workload. The difference stems from characteristics of IO-operations in each workload. The details are discussed in Section 4.8.4.

4.7.3 EXPERIMENTS WITHOUT ADAPTIVE PAGE CACHE TRANSFER

To confirm that our adaptive page cache transfer mechanism achieves the optimal result, the total migration time in Section 4.7.2 are compared with manually tuned results without using the adaptive page cache transfer. The adaptive page cache mechanism is turned off and a pre-defined parameter R is given. R represents the ratio of restorable

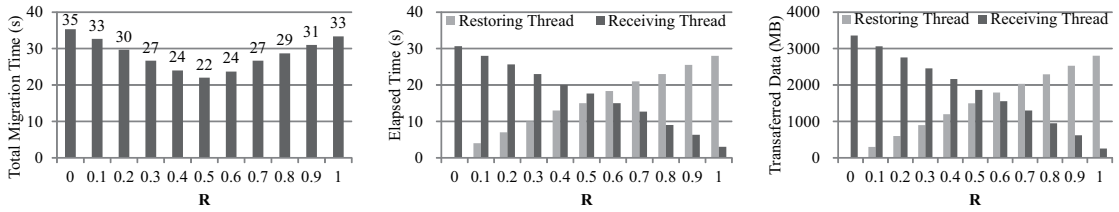


FIG. 4.9. Evaluation Results with WebServer Workload. Left: Total Migration Time, Middle: Elapsed Time Consumed by Receiving Thread and Restoring Thread, Right:Amount of Data Transferred by Each Thread.

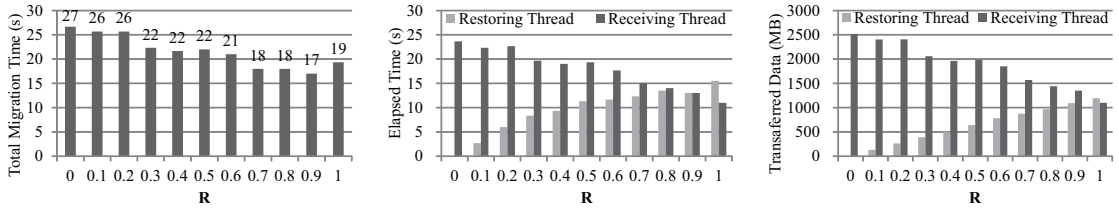


FIG. 4.10. Evaluation Results with Postmark Workload. Left: Total Migration Time, Middle: Elapsed Time Consumed by Receiving Thread and Restoring Thread, Right:Amount of Data Transferred by Each Thread.

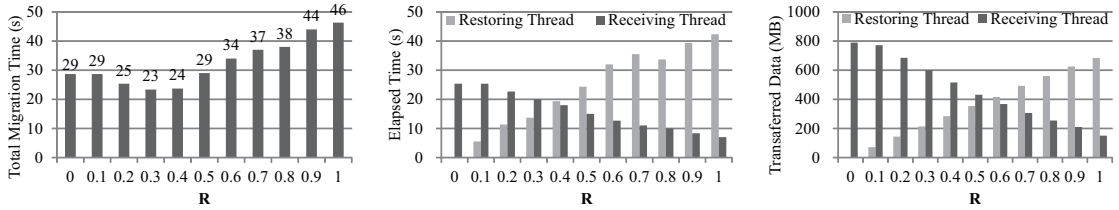


FIG. 4.11. Evaluation Results with TPC-C Workload. Left: Total Migration Time, Middle: Elapsed Time Consumed by Receiving Thread and Restoring Thread, Right:Amount of Data Transferred by Each Thread.

page cache transferred via the SAN to all the restorable page cache. That is, $R = 0.6$ means that 60% of the restorable page cache is transferred via the SAN and 40% is transferred via the general purpose network. Detailed analysis focusing on each thread (Receiving and Restoring) is also given in this section.

The left figures of Figure 4.9, Figure 4.10, and Figure 4.11 show the total migration time without the adaptive page cache transfer under WebServer, Postmark, and TPC-C workloads, respectively. The x-axis of each figure shows a value of R and the y-axis shows the total migration time for the R. Note that total migration time with $R = 0$ are not the same as the values shown in Section 4.7.2, because the values with $R = 0$ include overhead of retrieving the locations of restorable page cache. This overhead is discussed later in Section 4.8.2. For all workloads, the shortest total migration time achieved in this

experiment matches the one in Section 4.7.2. This means our adaptive page cache transfer mechanism achieves the optimal result without manually tuning any parameters.

The middle figures of Figure 4.9, Figure 4.10, and Figure 4.11 show the elapsed time consumed by Receiving Thread and Restoring Thread. The x-axis shows a value of R and the y-axis shows the elapsed time of each thread for the R. The dark-colored bars are for Receiving Thread and the light-colored bars are for Restoring Thread. For all workloads, the shortest total migration time is achieved when two bars have almost the same lengths. It means that when the total migration time becomes the shortest the SAN and the general purpose network are utilized throughout the migration, and neither thread waits for the other one to finish.

The right figures of Figure 4.9, Figure 4.10, and Figure 4.11 show the amount of data transferred by each thread. The dark-colored bars are for Receiving Thread and the light-colored bars are for Restoring Thread. An interesting point is that the trend is not necessarily the same as the trend in the elapsed time of each thread. For example in Figure 4.11, the elapsed time by the two threads are almost the same at $R = 0.4$ (where the shortest total migration time is achieved), but the amount of data transferred by the two threads are largely different at $R = 0.4$. This is because the bottleneck is the throughput of the HDD in the storage node, but not the networks. The reason why the HDD can become the bottleneck is discussed in Section 4.8.4. In WebServer and Postmark benchmarks the trends of the elapsed time and the amount of transferred data are the same because the network throughput the bottleneck in these cases.

Summary: The adaptive page cache transfer achieves the shortest total migration time which is nearly the same as the one manual parameter tuning achieves. The shortest total migration time is achieved when the two networks are utilized for the same duration of time.

4.7.4 SMALL IO PERFORMANCE PENALTY TO THE VM

Our mechanism has small IO performance penalty to the VM after a migration because it keeps the page cache warmed up transparently from the VM. Figure 4.12 shows the file read throughput of a VM on the HDD cluster. The x-axis shows the elapsed time in second from the beginning and the y-axis shows the throughput in blocks/second. The VM with 3GB of memory is migrated using our method with $R = 0.5$ during $50 < x < 62$ to measure the IO performance penalty by our method. The file is 2 GB large and cached

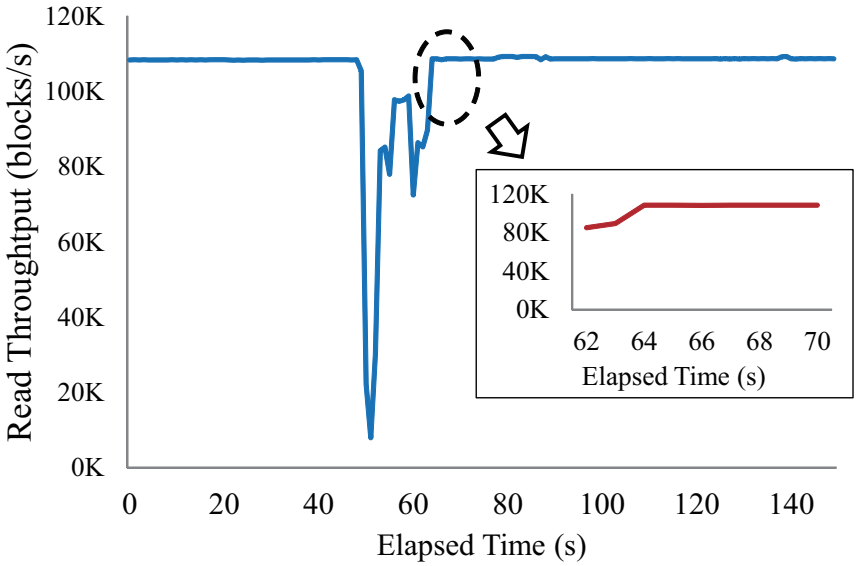


FIG. 4.12. File Read Throughput of a VM Reading a Large File. Blue line shows the result of the whole measurement and red line shows enlarged result for $62 \leq x \leq 70$. The VM is migrated during $50 < x < 62$ by our method. The throughput recovers to the maximum 2 seconds after the migration completes ($x = 64$). The degradation during the migration is out of our scope.

in the page cache before the measurement. Once the end of the file is reached, the file is read from the head again. The blue line is the result of the whole measurement and the red line is the result during 9 seconds after the migration completes ($62 \leq x \leq 70$).

The read throughput fully recovers 2 seconds after the migration completes, because there is no loss of page cache after the migration in our method. We did not conduct the same measurement with existing methods because the implementations are not provided, but deleting page cache causes much larger IO performance penalty because reading a 2 GB file not on page cache takes 14 seconds in our HDD cluster. The degradation during the migration ($50 < x < 62$) is a general phenomenon of migration [80] and out of our scope.

Summary: Our mechanism has negligible IO performance penalty to a migrated VM. File read throughput of a migrated VM fully recovers 2 seconds after a migration completes.

4.8 DISCUSSION

4.8.1 MATHEMATICAL ANALYSIS OF THE METHOD

We give mathematical analysis on how dividing page cache impacts on the total migration time, using the case that restorable page cache is manually divided with the parameter R as done in Section 4.7.3. We assume that there is no update to the VM memory during a migration to make the analysis simple and easy to understand. The optimal value of R that achieves the shortest migration time is denoted as R^* hereafter.

R^* depends on four factors: the amount of restorable page cache M_p, the amount of normal memory M_n, the throughput of transferring restorable page cache S_p, and the throughput of transferring normal memory S_n. R^* and the four factors satisfy:

$$\frac{R^* M_p}{S_p} = \frac{M_n + (1 - R^*) M_p}{S_n} \tag{4.1}$$

because both transfers must end at the same time to utilize the SAN and the general purpose network equally. Therefore, R^* is given by:

$$R^* = \frac{M_p + M_n}{M_p} \times \frac{S_p}{S_p + S_n} \tag{4.2}$$

Substituting (4.2) into (4.1) gives the time T consumed to finish both transfers:

$$T = \frac{R^* M_p}{S_p} = \frac{M_p + M_n}{S_p + S_n} \tag{4.3}$$

This means that the two network links can be regarded as an integrated network link with $S_p + S_n$ bandwidth.

We discuss how the balance of M_n and M_p affects R^* and how it results in total migration time. Suppose the two networks have the same available bandwidth ($S_n = S_p = S$) during migration, then we get:

$$R^* = \frac{M_p + M_n}{M_p} \times \frac{S}{S + S} \tag{4.4}$$

$$= \frac{M_p + M_n}{2M_p} \tag{4.5}$$

$$= \frac{1 + \frac{M_n}{M_p}}{2} \tag{4.6}$$

R^* must holds $0 \leq R^* \leq 1$ by the definition. This means that for our system to work efficiently, M_p and M_n must holds:

$$0 \leq \frac{M_n}{M_p} \leq 1 \tag{4.7}$$

because

$$0 \leq \frac{1 + \frac{M_n}{M_p}}{2} \leq 1 \tag{4.8}$$

When the equation (4.7) holds, the reduction ratio of total migration time is given by dividing T by the time consumed to transfer all memory pages with one link, which is:

$$\frac{T}{\frac{M_n+M_p}{S}} = \frac{\frac{M_n+M_p}{2S}}{\frac{M_n+M_p}{S}} \tag{4.9}$$

$$= \frac{1}{2} \tag{4.10}$$

This means that, if $S_p = S_n$ and $M_n \leq M_p$, total migration time is shortened to the half of the original in theory.

However, Table 4.2 shows that actual reduction ratio is smaller than 50% in all workloads although restorable page cache dominates more than 70% of the memory usage (Figure 4.2). The reason is that S_p and S_n are not equal, more concretely, S_p is non-negligibly smaller than S_n. Even if the available network bandwidth for the SAN and the GPN are the same, S_p can be smaller than S_n. Further discussion is given in Section 4.8.4 with real data from the three workloads.

4.8.2 SENDING PFNS AND DISK BLOCK NUMBERS WITH TCP/IP

Transferring PFNs of restorable page cache and disk block numbers with TCP/IP is the easiest method and applicable to any practical guest OS, but has overhead on total migration time. In the WebServer benchmark, the transfer takes 3 seconds even though the size of data to transfer is just 6 MB. This is calculated by multiplying the number of memory pages with 8 (in Linux a disk block number is 8-bytes long). This is because the web sever inside the VM arises many hardware interruptions to the network interface and our user-space program cannot use the network functionalities efficiently. Without the web server, it takes less than 1 second to send the PFNs and disk block numbers even if the vCPU usage is 100%. The overhead was 1.6 seconds for Postmark workload and

TABLE. 4.3. Comparison of Methods to Detect Restorable Page Cache

	Kernel Module	VM Introspection	IO Monitoring
Implementation	*Easy*	Hard	Middle
Runtime Overhead	*None*	*None*	Small
Guest OS Limitation	Module Installed	Specific Version	*None*
Page Cache History	*R/W*	None	Write

1.2 seconds for TPC-C workload.

Mechanisms that do not use network to for communications between a host and a VM can be alternatives. Examples are Symbiotic Virtualization [55] and the shared memory space used in [72], although they require much implementation cost.

4.8.3 DETECTING RESTORABLE PAGE CACHE IN OTHER METHODS

Installing a kernel module into the guest OS is the most feasible method to detect restorable page cache, although it is not the only one. Other possibilities to detect restorable page cache includes using VM introspection technique and monitoring IO operations to/from the storage.

VM introspection (e.g. [81]) is a technique that enables the host OS to understand the memory content of a VM running on it. The host OS requires no help of the guest OS, but the guest OS kernel must be a specific version that the host OS expects.

IO monitoring in [74] captures all IO operations between the guest OS and external storages. The method is implemented in the layer of VMM and emulated hardware thus it required no modification to the guest OS. However, it incurs small overhead while the VM is executed because of the IO capturing.

Table 4.3 shows comparison between our method (kernel module), VM introspection, and IO monitoring. The best characteristics among each row are showed *italic*. Kernel module is the easiest to implement among three methods. As shown in Section 4.6.4, it has 200 lines of code in C (error handling excluded). Runtime overhead means the overhead incurred by each method to the VM performance during non-migration time. The kernel module incurs literary no runtime overhead because the module is invoked just before a migration and it even does not need to be loaded into the kernel during non-migration time. This characteristic is highly important as described in the design

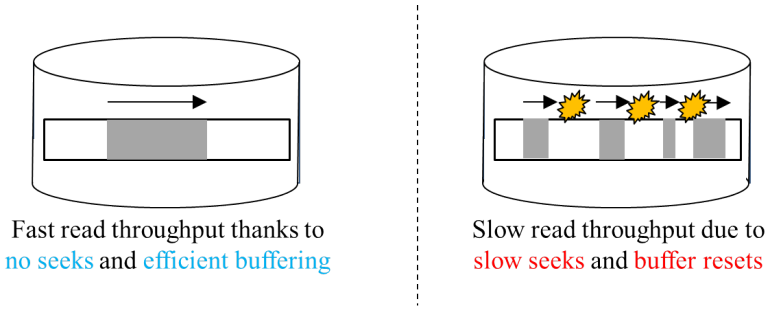


FIG. 4.13. Disk Block Sequentiality v.s. Read Throughput

criteria (Section 4.4.1).

An unique and promising characteristic of kernel module approach is that it can not only retrieve information from the guest OS but can also inject new behaviors to it. For example, a possible direction to extend our method is to exploit access history to page cache by the guest OS and to post-copy page cache that will not be used in the near future. Accepting small modification to the guest kernel allows to increment a counter each time page cache is hit, and to exploit that information for more efficient migration. VM introspection does not allow this extension because it can only get the information that is already embedded on an existing kernel. IO monitoring neither allows the extension because it cannot capture page cache hits, which are memory to memory data copy. The "Page Cache History" column shows this characteristics: kernel module can exploit read/write histories from/to page cache, while IO monitoring can only get write histories and VM introspection cannot get any history information.

4.8.4 BLOCK SEQUENTIALITY

This section analyses a reason of the difference on the efficiencies of our method among workloads. The total migration time in our method is largely affected by the sequentiality of disk blocks that contain restorable page cache. Random accessing to an HDD is slower than sequential accessing, because of slow seek operations and fruitless buffering (Figure 4.13). Therefore, when disk blocks to transfer are scattered across wide address range of the HDD, read throughput of the storage node can be the bottleneck of our method, instead of the networks.

Table 4.4 shows sequentiality of disk blocks to transfer in the workloads used in the evaluation. For each workload, we calculated average cluster size of restorable page cache for the workload. A cluster means a set of sequential disk blocks within the disk

TABLE. 4.4. Average Cluster Size of Restorable Page Cache.

Workload	Average Cluster Size
WebServer	57.1
Postmark	124.3
TPC-C	18.9

blocks to transfer. For example, if the disk blocks to transfer are {1, 2, 55, 56, 100,101,102}, the average cluster size is $\frac{2+2+3}{3} \approx 3.3$. Larger cluster size results in better read throughput of the blocks.

The results show that the average cluster size is the smallest in TPC-C workload and relatively the large in WebServer and Postmark workloads. These values obviously show why the reduction ratio of total migration time is not large under TPC-C workload. It can be expected that database-related workloads have small average cluster size in general and our proposal does not work efficiently under those workloads.

For another verification of the fact that disk block sequentiality largely affects total migration time for the proposed system, it is observed that accessing order to the disk blocks to transfer also affects total migration time. Three accessing orders, *ascending*, *descending*, and *PFN* are compared. Ascending order means that the disk blocks are read out in the ascending order of disk block numbers, which is the default of the proposed system. Descending order means that the disk blocks are read out in the descending order of disk block numbers. PFN order means that the disk blocks are read out in the ascending order of memory page frame numbers that have the same data with the disk blocks. Note that this is not the same as the ascending order as memory pages used for page cache are not necessarily straight-mapped to disk blocks. The total migration time was the shortest for the ascending order, which explains our hypothesis that reading disk blocks are largely affected by the accessing order. Therefore the system uses ascending order by default.

4.8.5 APPLICABILITY TO WIDE AREA LIVE MIGRATION

Wide area live migration is a technique to live migrate a VM from one data center to another far away data center and it has many use cases. Al-Kiswany *et al.* [82] achieve cross data center load balancing by using wide area live migration. This work is unique in the sense that it distributes the load of a data center to other distant data centers. Moghad-

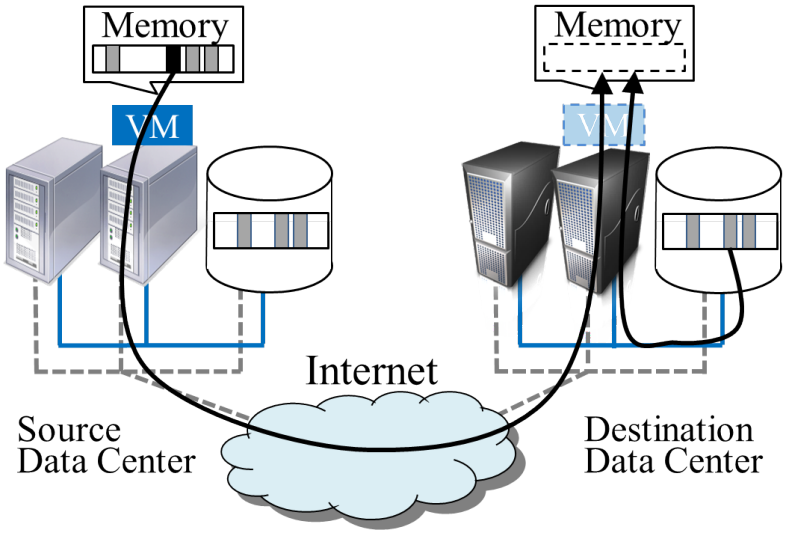


FIG. 4.14. Application of Page Cache Teleportation to Wide Area Live Migration

dam *et al.* [21] minimize the carbon footprint of a virtual private cloud through a live migration of VMs to a data center that uses clean energy sources as much as possible. Tsugawa *et al.* [83] propose to use wide area live migration for disaster recovery of indispensable IT systems. VMs can survive a disaster by using live migration of them to safe data centers when a disaster occurs. The work shows that when the great earthquake hit Japan in 2011, the network connectivity and the uninterruptible power supply were kept alive for dozens of minutes at Tohoku University (150 km from the epicenter).

The proposed system in this chapter is also applicable to wide area live migration. Figure 4.14 shows how it can be used for wide area live migration. The figure shows when a VM is about to migrate from the source data center to the destination data center. In each data center, computing nodes are connected with a GPN that is also connected to the Internet. Storage nodes are connected with a SAN that is not connected to the outside of each data center. We assume that disk images stored in the two data centers are periodically synchronized to achieve fast live migration within the two data centers, because the two data centers are hosted by the same organization, or they are a contract to do so for disaster recover. Undet this assumption, a wide area live migration is accelerated with our system as follows:

1. Normal data pages are transferred via the internet from the source data center to the destination data center.
2. Meanwhile, restorable page cache is transferred via the SAN **within** the destination data center, because the disk image is periodically synchronized during non-

migration time.

Synchronization of large files among geographically distant data centers has been approached by some studies. DRBD [84] is a block-device level synchronization mechanism that is already included in the Linux mainstream. It has a synchronous mode where a write to the local host cannot finish until the remote replica finishes writing the same data, and an asynchronous mode where a write can finish without waiting for an ack from the remote replica. To use DRBD with page cache teleportation, either using the synchronous mode during the whole data center operation, or by switching to the synchronous mode from the asynchronous mode before a migration is required. dsynch [85] provides periodic synchronization of VM disk images between two data centers with less CPU overhead and less cache pollution than merely calculating block-wise hash values on every synchronization time to find updated blocks. To use dsynch with our page cache teleportation, a mechanism to track which disk blocks are not yet synchronized is required to prevent transferring old data from storage on the destination data center.

4.8.6 LIMITATION

This section explains the limitation of applicability of Page Cache Teleportation. Page Cache Teleportation has two important assumptions:

1. The target data center has a SAN as a separated link from the GPN, and
2. The congestion of the SAN is similar to the one of the GPN.

The 1st assumption holds for many modern, big data centers (as mentioned in Section 4.3, CISCO suggests to do so [75]). However, it does not necessarily holds small private clouds that cannot have much budget. They might have a virtual SAN that is created by software (VLAN is commonly used in data centers), but Page Cache Teleportation does not work for a virtual software SAN.

The 2nd assumption does or does not hold depending on workload characteristics of the target data center. For example, if the target data center is designated for processing huge amount of data such as DNA analysis, SAN is most probable more congested than the GPN thus there is no room to transfer page cache via the SAN.

4.9 RELATED WORK

It has been pointed out that the large amount of page cache slows down total migration time thus it must be approached. Koto *et al.* discuss computational and time costs of live migration (referred as *migration noise*) in [72]. The work reduces the migration noise by skipping the transfer of restorable pages including page cache, free pages, and kernel objects that can be regenerated from other data. This method degrades IO performance of the VM due to the loss of page cache after a migration. Hines *et al.* also skips the transfer of the page cache by using the balloon driver of Xen [73]. In a paravirtualization environment with Xen, a guest OS returns unused memory pages to the Xen using the balloon driver. Hence, Xen can skip the transfer of the deleted page cache in a live migration. This method also degrades the IO performance of the VM after a migration because the VM must reload the deleted page cache from the disk.

Transferring page cache from storage to achieve fast live migration without deleting the cache has been proposed [74, 86]. The main advantage of our work is that we proposed the adaptive page cache transfer. Existing studies do transfer restorable page cache via the SAN in parallel with normal data transfer via the GPN, but they use the GPN exclusively for normal data. To mitigate the load imbalance between the SAN and the GPN, [86] introduces *lazy fetch* mechanism, whose core idea is similar to well-known post-copy live migration. The lazy fetch mechanism switches the execution host of the VM as soon as transferring normal data is finished. After the execution host of the VM is switched, the remaining restorable page cache is fetched on demand in response to the memory accesses at the destination host. This mechanism has the same problem as post-copy live migration has: accesses to the page cache which is not transferred yet takes long time and degrades the overall performance of the workload running in the VM. Our adaptive page cache transfer mechanism mitigate the load imbalance problem without introducing new overhead and is more suitable for IO-intensive VMs than existing works.

There is a body of knowledge on accelerating live migration, but our technique is complementary with most of them because Page cache teleportation accelerates the 1^{st} phase of live migration. Detailed explanation of this matter is in Section 3.6 in Chapter 3.

4.10 SUMMARY OF THIS CHAPTER

IO-intensive VMs have large amount of page cache in the memory, and this results in inefficient live migration by long total migration time. Recent studies tackled this problem by merely skipping the transfer of page cache in live migration, but this method penalizes the performance of the IO-intensive workloads running on the target VM. In this chapter, page cache teleportation, an advanced memory transfer mechanism for live migration of IO-intensive is proposed. The idea is that modern data centers have a dedicated storage area network (SAN), which has never been utilized for live migration before. Page cache teleportation shortens total migration time of an IO-intensive VM by transferring a portion of page cache via the SAN from a storage PM to the destination PM of the migration, not via normal network from the source PM. The method mitigates the problem with significantly smaller IO performance penalty to the VM than the existing method, because the the VM has no need to re-fetch the page cache from a storage PM. The experiments showed that the method shortened total migration time with by 13-33% for various IO-intensive workloads with negligible IO performance penalty. Integrated evaluations of Page cache teleportation and aggressive VM relocation are given in Chapter 5.

CHAPTER 5

EVALUATING ENERGY IMPACT OF LIVE MIGRATION

5.1 INTRODUCTION

As discussed in Chapter 1 (Introduction), the amount of energy consumed by data centers is becoming enormous in response to today's trend of using cloud computing. 1.5% of the overall electricity consumption in the US was due to data centers in 2006 [14], Google pays $1.4 million per month for its five data centers in the US [15], and energy consumption of data centers grew by 16.7% per year during 2000–2005 over the world [13].

Many studies have been done to reduce energy consumption of data centers both from dynamic VM consolidation side and from live migration side. However, actual amount of energy reduced by these researches in real data centers is still not revealed. This is because they lack integrated evaluation of dynamic VM consolidation and live migration, thus they cannot evaluate how much extra energy consumption and short sleep time are incurred by live migration in real data centers with real workloads. Dynamic VM consolidation researches calculate energy reduction with cost of live migration ignored, on the other hand live migration researches evaluate just a single migration and never discuss impact on a whole data center.

Given this situation of existing researches, this chapter conducts integrated evaluation of live migration methods proposed in this thesis and dynamic VM consolidation algorithms. First, MiyakoDori and Page cache teleportation are implemented onto a cloud simulator, SimGrid, by modeling their performance and energy characteristics. Second,

integrated evaluation of the two methods and a state of the art dynamic VM consolidation algorithm, FFD, is conducted with realistic PM energy and workload settings.

This chapter is structured as follows. Section 5.2 clarifies the problem tackled in this chapter. Section 5.3 explains how the problem is approached and by integrated simulations. Section 5.5 gives experimental setups and results. Section 5.9 describes related work and future directions. and Section 5.10 concludes the chapter.

A part of this chapter has been published in a refereed paper. The copyright of the paper is hold by IEEE Computer Society.

1. Soramichi Akiyama *et al.* Evaluating Impact of Live Migration on Data Center Energy Saving. In *Proceedings of 6th IEEE International Conference on Cloud Computing Technology and Science*, pages 759 - 762, 2014.

5.2 THE PROBLEM AND APPROACH

5.2.1 EXTRA ENERGY CONSUMPTION OF LIVE MIGRATION

Extra energy consumption of live migration is incurred by the load increase due to a live migration process. The whole memory of the migrated VM must be transferred from the source PM to the destination PM during live migration. All the memory pages of the VM are accessed and pushed into the network on the source PM, and they are received from the network and written into the memory on the destination PM. These procedures increase load of CPU, memory, network and buses and result in increased energy consumption.

Figure 5.1 shows power of PMs when a VM is migrated between them. The PMs are rack-mounted servers, each of which has a 4-core Intel Xeon X5460, 8 GB of memory, three 1 Gbps NICs, and an HDD. The VM has 1 GB of memory and is updating 128 MB of its working set with the speed of 100 MB/s. The lines show the power consumption of the source PM, the destination PM, and the storage PM that hosts NFS to save the disk image of the VM. The x-axis shows elapsed time in second from the beginning of the measurement and the y-axis shows power consumption in Watt at each time point. The first value increase before $x = 30$ is because the VM is being booted during that time. The second value increase during $120 < x < 140$ is because of the migration. Both the source PM and the destination PM has non-negligible extra energy consumption during

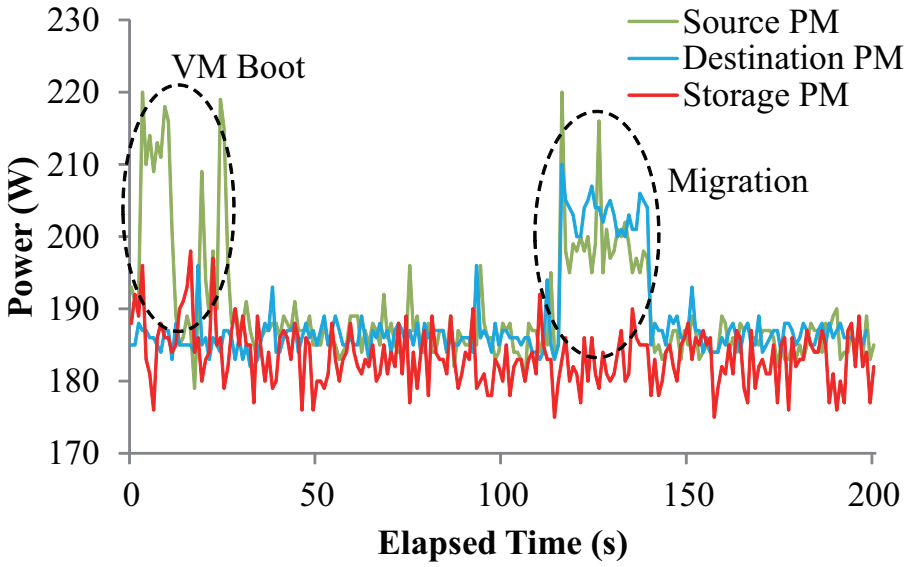


FIG. 5.1. Extra Energy Consumption by Live Migration. Both the source PM and the destination PM consume non-negligible amount of extra energy during a migration. The second value increase ($120 < x < 140$) is due to a live migration and the first value increase ($x < 30$) is due to booting the VM.

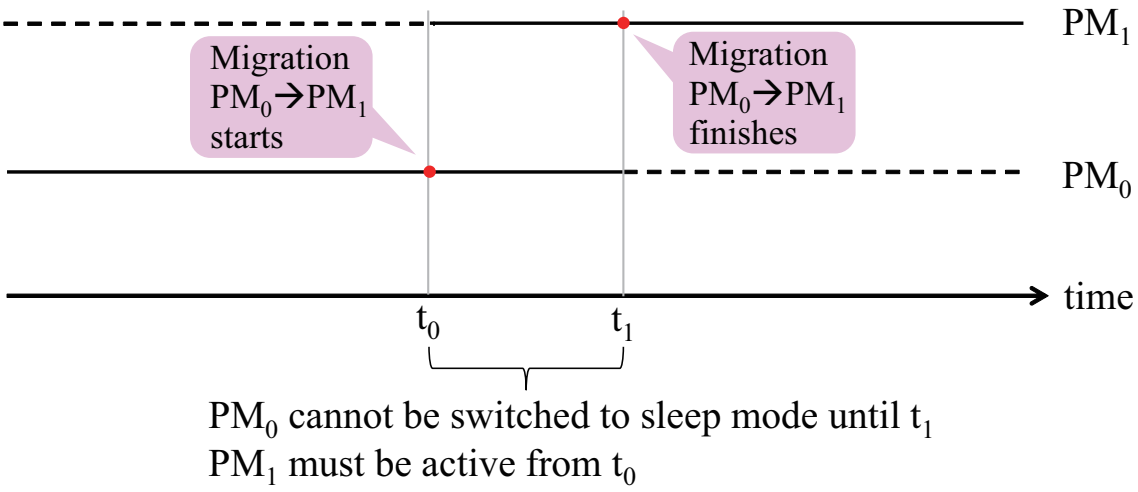


FIG. 5.2. Short Sleep Time of PMs due to Live Migration

the migration. It is a well known fact that booting a PM/VM consumes much energy, and this figure gives us an intuition that live migration consumes comparable amount of energy to booting.

5.2.2 SHORT SLEEP TIME DUE TO LIVE MIGRATION

The length of time that a PM can be in a sleep mode in dynamic VM consolidation system largely depends on how quick a migration finishes, as well as on workload characteristics running in the data center. This is because even when the only one VM hosted in a PM is migrated to another PM, the source PM cannot be switched into into sleep mode until the migration finishes. Figure 5.2 shows a typical example of this situation. The horizontal axis shows the time, and lines denoted PM_0 and PM_1 show if the PMs are active or in sleep at each time. A solid line means that the PM is active during the period, and a dashed line means that the PM is in sleep. In this figure, a VM is running on PM_0 at first and then it starts a migration to PM_1 at t_0. In an ideal situation where live migration can be finished with zero time cost, PM_0 can be switched into sleep mode at t_0, as soon as the VM starts a migration. However, live migration does take some time thus PM_0 cannot be in sleep until t_1, where the migration finishes after transferring all the memory of the VM. This means that longer and longer the migration takes, larger and larger amount of energy is consumed to keep PM_0 active until the migration finishes.

5.2.3 THE PROBLEM: ENERGY OVERHEAD V.S. ENERGY REDUCTION

Evaluating energy saving given by a dynamic VM consolidation algorithm in a real data center requires not only analyzing or simulating the algorithm itself (energy reduction), but also considering extra energy consumption by live migration to execute it (energy overhead). Existing studies of dynamic VM consolidation discuss the former and ones of live migration mechanisms discuss the latter, but integrated evaluations of them are missing. Studies of dynamic VM consolidation either consider the time and energy cost of live migration negligible [23, 24, 25], or take only the time cost of live migration into account [26, 27]. However, the energy overhead is not negligible but worth being evaluated as shown in this chapter.

Accelerated live migration mechanisms [87, 76, 31, 30, 74] are widely researched. However, both the energy overhead of them and the energy reduction given by a combination of them and dynamic VM consolidation are not researched. Use of accelerated live migration mechanisms depending on workload characteristic and data center operation policies is considered a promising method to improve data center efficiencies. Therefore, it is important to approach energy overhead and energy reduction with accelerated live

migration mechanisms as well.

In summary, estimating the amount of energy the dynamic VM consolidation researches can reduce in a real data center is difficult because of two reasons:

1. They lack integrated evaluations of the energy reduction they give and the energy overhead by live migration.
2. Many researches have proposed accelerated live migration mechanisms and they affect both the energy reduction by dynamic VM consolidation and energy overhead of live migration.

5.3 MODELING LIVE MIGRATION

5.3.1 PERFORMANCE MODEL AND ENERGY MODEL

Performance and energy models of live migration are built in this thesis to conduct integrated evaluation of dynamic VM consolidation and live migration.

PERFORMANCE MODEL describes how long a live migration takes under a given environment (e.g. VM memory size, network bandwidth, workload characteristic running on the VM). It is used to calculate how long each server can be turned into low-power states during data center operations.

ENERGY MODEL describes how much extra energy a live migration consumes under a given environment. It is used to calculate how much energy is lost by conducting live migrations to execute dynamic VM consolidation algorithms.

5.3.2 SIMGRID

SimGrid "is a scientific instrument to study the behavior of large-scale distributed systems such as Grids, Clouds, HPC or P2P systems. It can be used to evaluate heuristics, prototype applications or even assess legacy MPI applications" (cited from the official website [88]). It was originally developed as a grid simulator to evaluate such as the amount of transferred data in a given MPI workload, but recent versions also have cloud supports such as creating a VM on a PM and migrating a VM from one PM to another.

SimGrid is an event-based simulator and the user of it writes events (e.g. VM creation, workload execution on the VM, VM migration) to simulate and evaluate things the user

wants. The simulator can be accessed through normal programming languages such as C, Java, and Ruby, and the user can create events using function/method calls of that language. For example, a function call to `MSG_vm_migrate(v, dst)` migrates the VM named 'v' to the PM named 'dst'. The simulations in this thesis are conducted by modifying and using Simgrid as follows:

1. First, SimGrid implementation was modified to support MiyakoDori and Page cache teleportation. Details of the implementation are explained in Section 5.3.4 and Section 5.3.5.
2. Next, required events to simulate aggressive VM relocation are written using the simulator functions SimGrid provides.

All implementations in this thesis are based on the latest version of SimGrid (as of April 2014) retrieved from its git repository [89]

5.3.3 MODELING NAÏVE LIVE MIGRATION

This section explains how to model and implement the naïve pre-copy live migration into a simulator.

PERFORMANCE MODEL

Performance model of pre-copy live migration is discussed in [90] and is implemented into a well-known simulator SimGrid. This model considers not only the allocated memory size of a migrated VM but also memory updates due to workloads running on the VM and network resource contention from migrations of other VMs. The paper shows pre-copy live migration is well simulated by its model, thus we use the model implemented in SimGrid.

ENERGY MODEL

Extra energy consumption of pre-copy live migration has been discussed in literatures [38, 60, 59, 91]. Characteristics of the extra energy consumption are:

1. Both the source PM and the destination PM undergo large increase of energy consumption as shown above, Figure 4(b) in [38], and Figure 1(a) in [60].
2. Not only CPU load increase but also memory and network load increase contributes to the extra energy consumption because memory and network occupy non-negligible parts of energy consumption within a server [92, 93].

3. The extra energy consumption is largely depends on the amount of transferred memory during migration as shown in Figure 5 in [38] and Figure 5 in [91]. This is because total migration time increases almost linearly as the amount of transferred memory increases.

In [38], extra energy consumption of PMs conducting live migration is measured with various VM memory size and network bandwidths. They show that it depends only on the total amount of transferred memory and is formulated as:

$$E_{mig} = \alpha V_{mig} + \beta, \tag{5.1}$$

where E_{mig} is the extra energy consumption in Joule, V_{mig} is the amount of transferred memory in the migration in Megabytes, and α and β are coefficients that vary depending on specific hardware (equation (15) in [38]). In their environment, the coefficients are 0.512 and 20.165 and equation (5.1) is instantiated as follows (equation (17) in [38]):

$$E_{mig} = 0.512 V_{mig} + 20.512. \tag{5.2}$$

Building a model that matches to our own servers is not a concern of this paper. Our severs use 185W when idle and 235W when all 4 cores are fully loaded, while their ones use around 190W when hosting an idle VM and 250W when loaded with memtester benchmark (Figure 4 in [38]). Thus, we assume our servers and their ones have similar energy characteristics and equation (5.2) can be applied as is for the energy model of pre-copy live migration in this paper.

A similar model is given in [91], although it claims that the model also slightly depends on available network bandwidth for a migration. We believe this difference is not a big issue, but it stems from the characteristics of their hardware. Broader bandwidth available for a migration results in faster access to the memory, NICs and buses inside the PMs. Extra energy consumption of a migration is constant regardless of available network bandwidth if the power consumption of them increase near linearly to the increase of the accessed speed, because the time taken for the migration decreases near linearly to it. Discussing this type of differences is not our purpose, thus we adopt the easy model described in [38] in this paper.

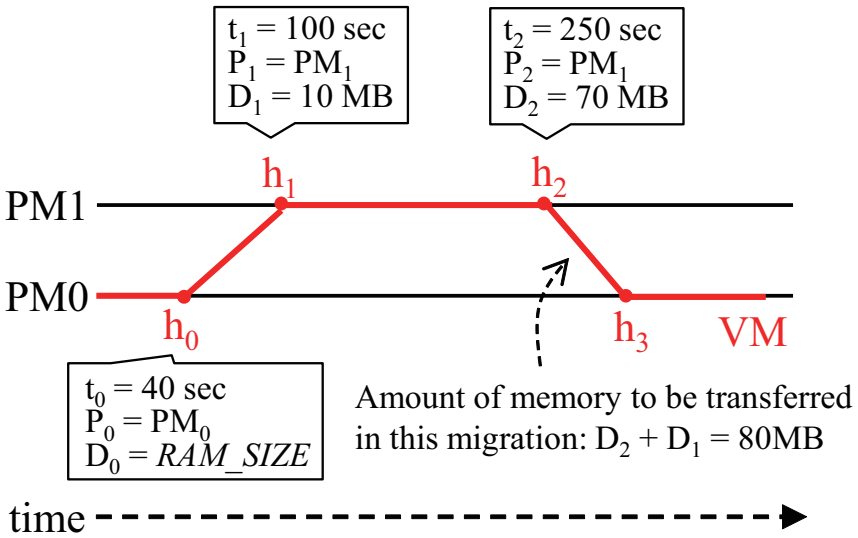


FIG. 5.3. Migration History. The VM is migrated from PM_0 to PM_1 and then from PM_1 to PM_0. h_i has a timestamp t_i, the VM's execution host P_i, and the amount of updated memory D_i during t_{i-1} and t_i. The amount of non-reusable memory in the second migration is $D_2 + D_1$. The same mechanism can be applied when the number of PMs is more than two.

5.3.4 MODELING MIYAKODORI

This section explains how to model and implement MiyakoDori into a simulator.

PERFORMANCE MODEL

We build a performance model of MiyakoDori to simulate how much energy it saves when it is used with dynamic VM consolidation. To build the model requires:

1. Performance model of pre-copy live migration
2. Amount of memory that can be reused in MiyakoDori

MiyakoDori can be simulated subtracting amount of reusable memory from the total memory size of the migrated VM and simulating pre-copy live migration. This is because MiyakoDori works totally the same as pre-copy live migration after reusing non-updated memory in the initial memory transfer.

Performance model of pre-copy live migration is already implemented in SimGrid, thus we use it as the base of our model implementation. Amount of reusable memory is calculated by tracking updated memory pages during VM execution in the real MiyakoDori implementation. SimGrid does not support page-wise operations such as dirty page, thus emulating this mechanism in Simgrid is infeasible.

To solve this issue, we introduce migration history h_i ($i \in \{1, 2, ..., n\}$) of each VM into SimGrid. The elements of h_i are a timestamp t_i, the VM's execution host P_i at t_i, and the amount of VM's updated memory D_i during t_{i-1} and t_i. A history is recorded every time a migration is started/finished. Figure 5.3 shows how migration histories are recorded. Migration histories are recorded at h_0, h_1, h_2, and h_3 in the figure. Each horizontal line (PM0, PM1) shows a slot of the PM that can host a VM. The bent red line shows a VM migrated from PM_0 to PM_1 during t_0 and t_1, and then from PM_1 to PM_0 during t_2 and t_3 (elements of h_3 are not drawn due to the space limit). Calculating the amount of reusable memory in the second migration requires the amount of updated memory since last time the VM was hosted on PM_0. This is equal to $D_2 + D_1$ because D_i is the amount of updated memory during t_i and t_{i-1}.

We confirm that our performance model simulates actual total migration time well. Live migrations are conducted using the real implementation and the simulator to compare the total migration time. The procedure of the experiments is:

1. VM V is booted on PM_0.
2. V is migrated to another PM_1.
3. V executes a workload that dirties W MB of working set with D MB/s for T seconds. W is chosen from {128, 256, 512, 1024}, D is chosen from {2, 4, 8} and T is chosen from {10, 30, 60, 300} thus there are $4 \times 3 \times 4 = 48$ combinations in total.
4. V is migrated back to PM_0 after the workload execution. The total migration time for this migration is compared.

Figure 5.4 shows comparison of total migration time across the simulation and the real implementation. Each point in the figure represents one parameter combination of W, D amd T. The x-value of each point is the total migration time given by the simulator and the y-value is the total migration time the real implementation takes. The main concern here is to confirm that our model simulates the real implementation well but not to check the actual values, thus we do not explain which point represents which parameter combination. The best fitting line given by least square analysis is: $y = 0.953x + 3.6487$. The fact that the coefficient of x (0.953) is nearly 1 means the model properly simulates the total migration time. The constant factor (+3.6487) cannot be 0 because the real implementation takes 2–3 seconds for the preparation phase before starting actual memory transfer. Therefore we add 2.5 seconds of waiting before starting a migration in the simulation.

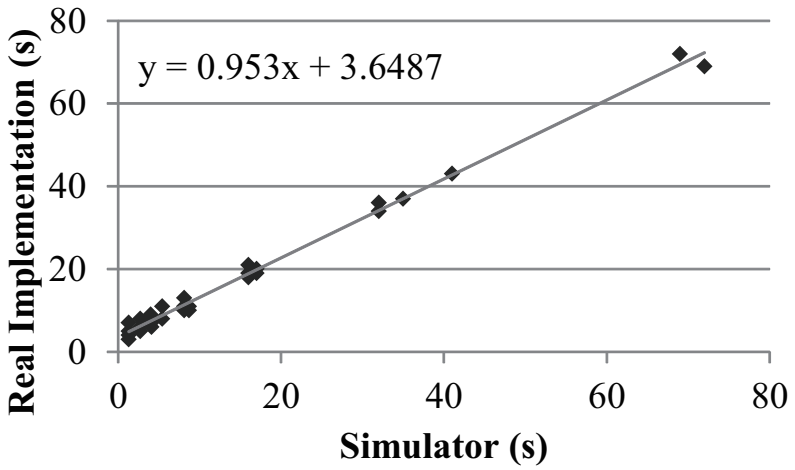


FIG. 5.4. Validation of the Performance Model of MiyakoDori. Each point has simulated total migration time as the x-value and real total migration time that MiyakoDori takes as the y-value. The best fitting line ($y = 0.953x + 3.6487$) shows our model well simulates the real values.

ENERGY MODEL

MiyakoDori has two different procedures from pre-copy live migration, thus they must be considered to build an energy model. For the details please refer Section IV-A in [87].

1. Dirty page tracking is enabled even during non-migration time.
2. Generations of memory pages are exchanged and compared between the source PM and the destination PM to detect which memory pages must be transferred.

Procedure (1) incurs negligible amount of CPU and memory load increase, and we confirm that enabling dirty page tracking increases no energy consumption even when the VM is running a memory intensive workload, which invokes dirty page tracking more frequently than non memory intensive workloads. Procedure (2) requires to transfer data from and to the source PM, but the amount of data is negligible compared to the amount of memory transferred in a migration.

Therefore, we conclude equation (5.2) can be used as is for MiyakoDori because the resource usage of it can be regarded the same as pre-copy live migration. Note that the amount of transferred memory, V_{mig}, does decrease with MiyakoDori thus the total energy consumption for a migration also decreases.

5.3.5 MODELING PAGE CACHE TELEPORTATION

This section explains how to model and implement Page cache teleportation into a simulator.

PERFORMANCE MODEL

In this thesis, page cache teleportation is modeled as a method to reduce the amount of transferred memory in the 1^{st} phase of its process. A reduction ratio, P, is introduced to specify how many percentage of memory is transferred via the SAN. For example, when $P = 0.25$, 25% of memory pages of the migrating VM is transferred via the SAN while the other 75% is transferred via the general purpose network. Note that P does not necessarily match the ratio of the amount of restorable page cache within VM memory, because page cache teleportation has the novel adaptive page cache transfer technique.

The limitation is that this modeling is applicable when (and only when) it is possible to acutually migrate a VM running the target workload with page cache teleportation in the target environment. P depends not only on the amount of restorable page cache within VM memory (workload characteristics), but also on the storage performance and available network bandwidths of the SAN and the general purpose network.

ENERGY MODEL

An important notice about evaluating page cache teleportation with regard to energy consumption is that it does not reduce amount of transferred memory while it shortens total migration time. This is because page cache teleportation is not a method to reduce the amount of transferred memory for live migration, but it is a method to offload a part of the transferring task into the SAN from the general purpose network.

Given the fact above, this thesis assumes that migrating a VM with page cache teleportation consumes the exactly same amount of extra energy consumption as the VM migrates with normal pre-copy live migration. More concretely, the same energy model as the normal pre-copy live migration shown in Section 5.3.3 is used. Note that in MiyakoDori even the model equation is the same the extra energy consumption is reduced thanks to smaller amount of transferred memory, but in page cache teleportation the amount of transferred memory is also the same as normal pre-copy live migration.

5.4 DYNAMIC VM CONSOLIDATION ALGORITHMS

5.4.1 A SIMPLE ALGORITHM: WAREHOUSE-HIGHPOWER

The warehouse-highpower strategy is a simple dynamic VM consolidation algorithm. PMs are divided into two categories in this strategy: warehouse server (WHS) and high-power server (HPS). A WHS hosts idle VMs that are overcomitted; i.e. a 4-core WHS hosts more than four 1-core VMs. A HPS hosts busy VMs that are not overcomitted; i.e. a 4-core HPS cant hosts no more than four 1-core VMs, or two 2-core VMs, or so. A VM is migrated from a WHS to a HPS when the load changes to heavy from light and the other way around when the load changes to light from heavy. The simulated data center has 32 HPSs, each of which has 4 cores and can host four 1-core VMs, and one WHS. How to select a HPS to migrate a busy VM depends on policies of data center operations, thus we simulate three policies. Note that our intention is to cover various operation policies, but not to show a specific one is better than others.

Most Dense: A HPS that is hosting the largest number of VMs (but not exceeding the capacity) is chosen. It simulates a data center where reducing the total energy consumption is the primary concern.

Least Dense: A HPS that is hosting the smallest number of VMs is chosen. It simulates a data center where reducing performance interference from other VMs is the primary concern.

Random: A HPS that is hosting the largest number of VMs or one hosting the smallest number of VMs are chosen equally (50% probability). It simulates a data center where some VMs can be co-located without any concern of interferences, while other VMs must be scattered as much as possible.

5.4.2 A STATE OF THE ART ALGORITHM: FFD

Finding the best VM placement in a dynamic VM consolidation system can be interpreted as a bin packing problem. In this model, a PM is represented as a bin with a capacity of resource and a VM is represented as a volume of resource usage. Some studies try to build more accurate models that map dynamic VM consolidation problems into vector packing problems [94, 95], but this thesis adopts the simple one-dimensional bin packing model because how to accurately model the dynamic VM consolidation problem

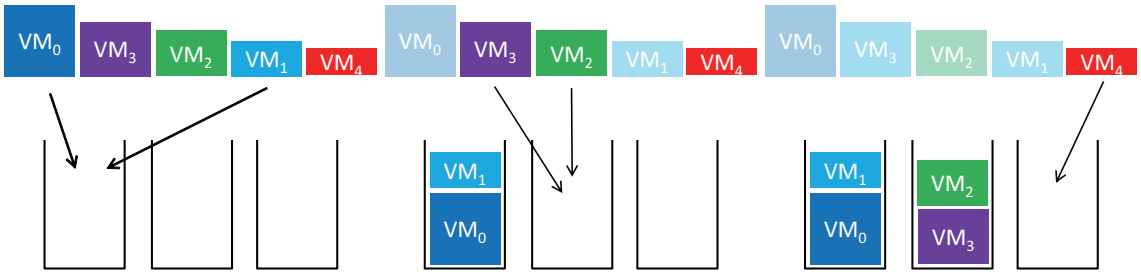


FIG. 5.5. The FFD Algorithm.

is under ongoing discussions in the research field.

First fit decreasing (FFD) is a fast huristic algorithm for bin packing problems. Here "fast heuristic" means that bin packing problems are NP-hard thus huristic algorithms are used to solve them in a practical time- and computation-cost. The FFD algorithm is described as follows:

1. Sort the bins in the descending order of their capacities.
2. Sort the volumes in the descending order of their sizes.
3. Starting from the largest bin and the largest volume, repeat the following until no volume remains un-packed.
   (a) If the current bin can include the current volume, put the current volume into the current bin.
   (b) Move to the next-largest volume and then try (a) again.
   (c) If the sequence reaches to the smallest volume, the current bin is supposed to be full. The sequence moves to the next largest bin and the largest un-packed volume.

The FFD algorithm is proved to give 11/9 worst-case approximation when all bins have the same size [24], which in turn means all PMs have the same capacity. In discussions hereafter, this chapter assumes that all PMs (bins) have the same capacity.

5.5 EVALUATION METHODOLOGY

5.5.1 METRICS

The impact of extra energy consumption and short sleep time incurred by live migration on overall energy saving is evaluated by integrated simulations with aggressive VM relocation. The metrics explained below are measured in the integrated simulations.

Slept Time Ratio: The ratio of the time during which a PM was in the low-power state against the time during which it was active. This metric shows how live migration impacts short sleep time. The values are averaged across all the PMs.

Saved Energy Ratio: The ratio of the amount of energy saved by dynamic VM consolidation. This metric shows how much actual energy a combination of a dynamic VM consolidation and a live migration mechanism save, with live migration overhead taken into account. The values are calculated by subtracting actual energy consumption from expected energy consumption in the naïve case, where all PMs are always active and live migration never occurs. Note that it is not the same as Slept Time Ratio because of energy consumption by live migration and sleeping PMs.

Energy Overhead: The ratio of extra energy consumption by live migration against the total energy used by all the servers in the data center. This metric shows how much extra energy consumption live migration incurs. This metric also means the amount of wasted energy used for conducting live migration but not for fruitful computation.

5.5.2 EXPERIMENTAL SETUP

The metrics described in Section 5.5.1 are evaluated in simulations conducted using modified SimGrid. This section explains experimental setup that are common for MiyakoDori and Page cache teleportation. Settings that are specific to each method are explained in the sections for each evaluation.

CONSOLIDATION ALGORITHM

VMs are dynamically consolidated across PMs using the Warehouse-highpower or FFD algorithms explained in Section 5.4. In the Warehouse-highpower algorithm, a VM migrates every time its load changes from heavy to light or other way around. In the FFD algorithm, consolidation is invoked every 300 seconds starting from 150 seconds after the simulation begins. Migrating only one VM is not capable in the FFD algorithm due to its nature.

WORKLOAD

Each VM on the simulated data center executes a given bursty workload. Three types of workload are used to conduct the simulation and are summarized in Table 5.1. The first one is named *Full-Zero*, where the load takes either 100% of vCPU or 0. The second and last ones are named *WebServer* and *TPC-C*, which simulates the WebServer and TPC-C

TABLE. 5.1. Details of the Simulation Workloads

Name	Full-Zero	WebServer	TPC-C
Load	Light: 0, Heavy: 100%	Low: 10%-30%, High: 80%-100%	
Working Set Size	128 MB		1024 MB
Busy/Idle Interval	10 mins - 20 mins		

workloads used in Chapter 3 and Chapter 4. The workloads have heavy and light CPU load phases that alternate in short time periods. Specifically, the load of each VM changes as follows in our experiments:

1. Heavy load continues for a random period chosen from a range of 10 mins to 20 mins, with granularity of 100 seconds; i.e. 600 sec, 700 sec, ..., 1200 sec.
2. Then the load becomes light for a random period chosen from the same range.

When the load is heavy (light), a vCPU is loaded to 100% (0%) in Full-Zero workload, and to a random number chosen from 80% to 100% (10% to 30%) in WebServer and TPC-C workloads. The memory of a VM is 4 GB large, and the working set is updated with the speed of 2MB/s when the load is heavy and has the size of 1024 MB, 128 MB, and 128 MB in TPC-C, WebServer, and Full-Zero workload, respectively.

ENERGY MODELS

The energy model of a PM is built upon power measured with a real server (a rack-mounted server with a 4-core Intel Xeon X5460, 8GB of memory, three 1 Gbps NICs, and an HDD). The server consumes 20 Watt, 185 Watt, and 235 Watt when it is in sleep mode, when it is on but idle, and when its 4 cores are fully loaded, respectively. Therefore, given the load and capacity of a simulated PM as l and c, the PM consumes P (Watt) of power that is formulated as follows:

$$P = \begin{cases} 20 & (l = 0) \\ 185 + (235 - 185) \times \frac{l}{c} & (\text{otherwize}) \end{cases} \tag{5.3}$$

where l is defined by adding the load of all VMs running on the PM (e.g. when the PM hosts a VM with 30% load and another with 60% load, l for that PM is 90) and c as 100×4, meaning that each of 4 cores can serve 100% capacity. In the Warehouse-highpower algorithm, the WHS (warehouse server) is assumed to be moderately loaded for the whole simulation and consumes $\frac{235+185}{2} = 210$ Watt of energy.

TABLE. 5.2. Simulation Settings for MiyakoDori

Parameter	Value
Number of PMs	32
Number of cores of a PM	4
Number of cores of a VM	1 or 2 or 4
Number of VMs	128 or 64 or 32
NW bandwidth between PMs	10 Gbps
Memory size of a VM	4 GB

5.6 EVALUATING MIYAKODORI

5.6.1 OVERVIEW

MiyakoDori is evaluated by integrated simulations with Warehouse-highpower and FFD algorithms. The simulation settings are described in Table 5.2. There are 32 PMs connected with 10 Gbps bandwidth, each of which have 4 cores. A VM occupies 4 GB of memory. The simulations are composed by two parts:

1. A simulation in a simple, ideal environment with Warehouse-highpower algorithm:. VMs run Full-Zero workload described in Section 5.5.2 where the load is either 0% or 100% of the VM's capacity. The number of VMs that can be hosted in a PM is bound by the number of cores a VM, which is either 1, 2 or 4. Memory images of the VMs used for memory reusing can be stored as many as possible in a PM.
2. A simulation in more complex, realistic environment with FFD algorithm: VMs run either WebServer or TPC-C workload described in Section 5.5.2, where the load is randomly decided and working set size and its update speed is defined based on real values. A PM can host VMs as many as the sum of their load becomes 400 (100% $\times$ 4 cores). Memory images of the VMs used for memory reusing can be stored either up to 8 images (assuming that a PM has 32 GB of memory), or infinitely.

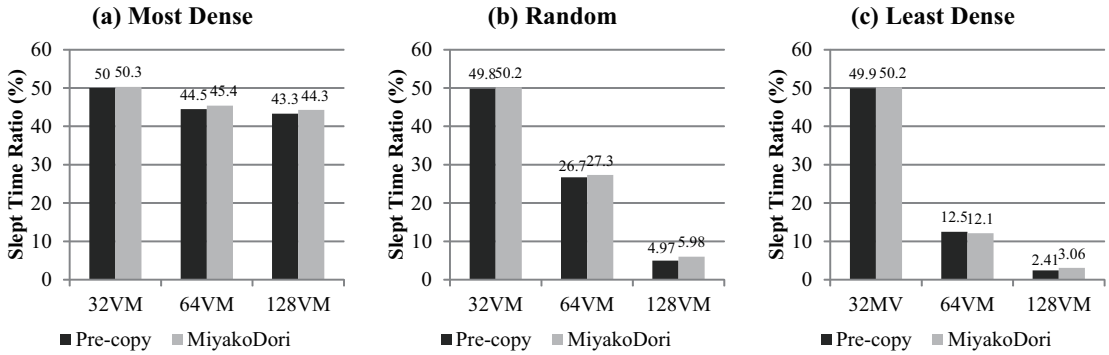


FIG. 5.6. Slept Time Ratio of Most Dense policy (Left), Random policy (Middle) and Least Dense policy (Right).

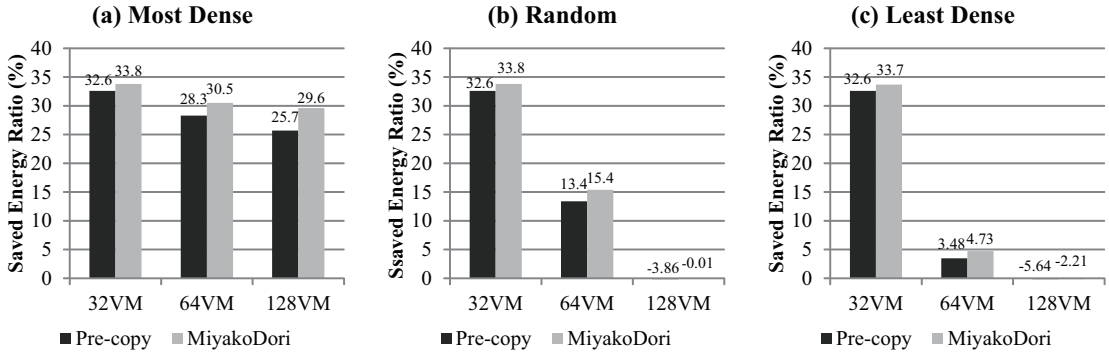


FIG. 5.7. Saved Energy Ratio of Most Dense policy (Left), Random policy (Middle) and Least Dense policy (Right).

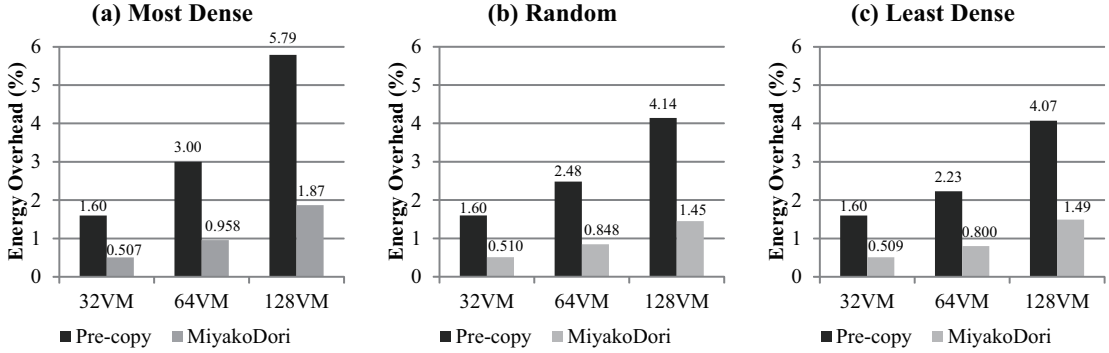


FIG. 5.8. Energy Overhead of Most Dense policy (Left), Random policy (Middle) and Least Dense policy (Right).

5.6.2 SIMULATION RESULTS WITH WAREHOUSE-HIGHPOWER

Figure 5.6, Figure 5.7 and Figure 5.8 show Slept Time Ratio, Saved Energy Ratio and Energy Overhead, respectively. Figures indicted (a) show the values for Most Dense policy, ones indicated (b) show the values for Random policy and ones indicated (c) show the values for Least Dense policy. Bars indicated 32, 64, 128 VM are the results

when each HPS can host 1, 2, or 4 VMs respectively because each VM has 4, 2, or 1 vCPUs. This simulates various usage of VMs by users. Experiments with mixed number of vCPUs are one of future work. All values are averaged across 30 simulations runs. Slept Time Ratio is calculated only across 32 HPSs. Saved Energy Ratio and Energy Overhead are calculated with energy consumption of the WHS taken into account. Note that the number of PMs is 33 under dynamic VM consolidation, but it is 32 when dynamic VM consolidation is not used.

SLEPT TIME RATIO

The values depend on number of VMs and each data center operation policy (Most dense, Random, Least dense). In the 32VM cases, all policies show the same value because a PM can host only one VM thus the difference of the policy makes no change. However in the 128VM cases, Most Dense policy yields more than 10 times better results than Least Dense policy. This is because Least Dense policy tries to distribute VMs as much as possible to prevent performance interference. In Least Dense policy all PMs become active when 32 VMs are busy, while in Most Dense policy 8 PM is enough to host 32 VMs. Note again that our intention is not to state Most Dense policy is better than Least Dense policy, but to evaluate the impact of live migration on various data center operation policies. For Slept Time Ratio, MiyakoDori does not contribute much because the network bandwidth between PMs is large (10 Gbps). However, it contributes much to Saved Energy Ratio and Energy Overhead as shown below.

SAVED ENERGY RATIO

The trend of the values are the same with the one in Slept Time Ratio. The most important point is that Saved Energy Ratio is greatly smaller than Slept Time Ratio all cases because of two reasons. First, live migrations conducted to consolidate VMs lose extra amount of energy as focused on this paper. Second, PMs consume non-negligible amount of energy even in the low-power state (20 Watt in our settings). This energy consumption is also unavoidable because totally shutting off a PM may requires even larger amount of energy to boot the PM again. In some cases (128VM in Least Dense and Random), the ratios are negative values even though PMs are in sleep mode for positive amount of time. It means in these cases always keeping all PMs active consumes smaller amount of energy than using dynamic VM consolidation. This is a good example of our idea that impact of live migration on data center energy saving must be considered carefully. Considering energy overhead of live migration reveals that the overall energy consumption can be

increased by dynamic VM consolidation if the efficiency of it is not large due to the data center operation policy.

ENERGY OVERHEAD

The values show the ratio of wasted energy for live migration to the energy used for fruitful computation. Note that the values are not equal to the differences between Slept Time Ratio and Saved Energy Ratio because the they include the amount of energy consumed by PMs in sleep mode and by the WHS. In the 128VM cases, Energy Overhead is 5.79%, 4.14% and 4.07% in Most Dense, Random and Least Dense policies, respectively. These values are not negligible at all because the overall amount energy consumption of data centers is huge as shows in Section 5.1. Using MiyakoDori decreases the Energy Overhead to less than 1.9% and saves 2.6% – 3.9 % of the energy consumption by the PMs in a data center. According to Barroso *et al.*, IT equipment consumes 50% of the overall energy consumption in a traditional data center (Figure 5.2 in [92]). This fact and our Energy Overhead measurements indicate that pre-copy live migration loses up to 2.9% of the overall data center energy consumption, and use of MiyakoDori decreases the overhead to less than 0.9%. These values can give important insights to power management or cost analysis of data centers, and we claim that it is our integrated evaluations that make it possible to draw the values.

5.6.3 SIMULATION RESULTS WITH FFD

Figure 5.9 shows Slept Time Ratio, Saved Energy Ratio, and Energy Overhead when FFD is used for the dynamic VM consolidation algorithm and the VMs execute WebServer workload. Figure 5.10 shows the same metrics when the VMs execute TPC-C workload. The number of PMs is 16 and the number of VMs is 64. The parameter C (8 or ∞) denotes number of memory images that a PM can save. Note that C has nothing to do with the results when normal pre-copy live migration is used, thus only one set of results is shown for pre-copy live migration.

SLEPT TIME RATIO

In both workloads, normal pre-copy migration yields the same results because it have to transfer all the memory of the target VM no matter how working set is smaller than the total memory usage (128 MB v.s. 4 GB in WebServer workload). MiyakoDori yields smaller Slept Time Ratio than normal pre-copy live migration, which betrays the intu-

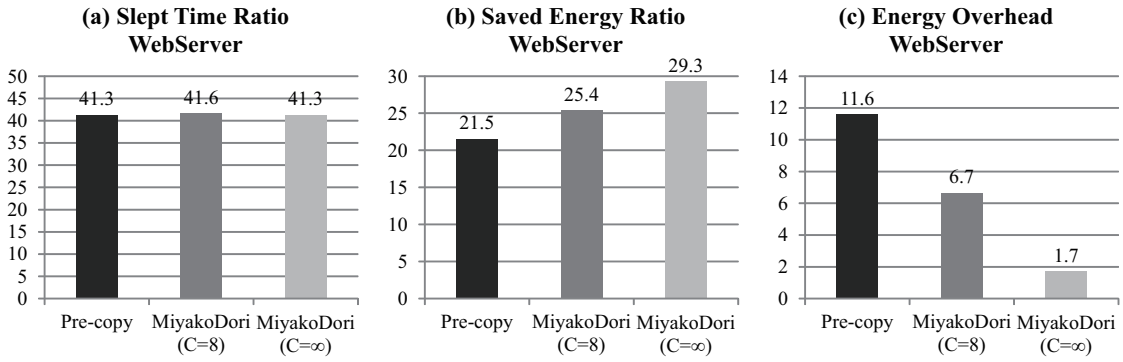


FIG. 5.9. Slept Time Ratio (Left), Saved Energy Ratio (Middle), and Energy Overhead (Left) with FFD Algorithm and WebServer Workload for 64 VMs and 16 PMs.

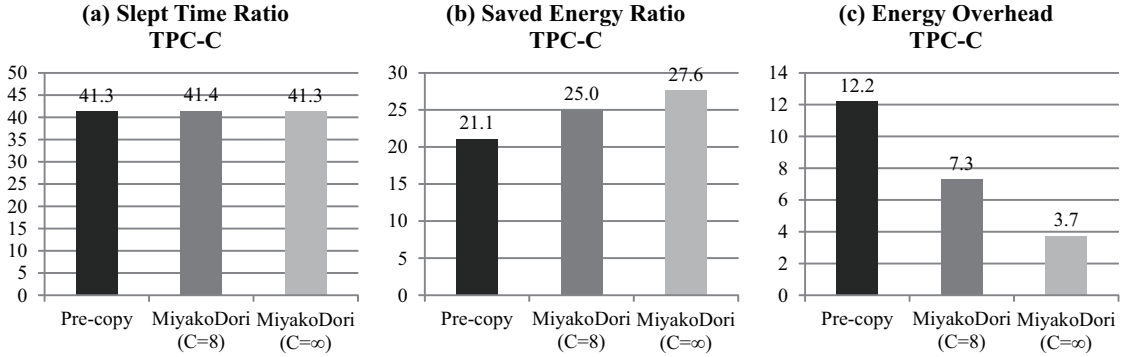


FIG. 5.10. Slept Time Ratio (Left), Saved Energy Ratio (Middle), and Energy Overhead (Left) with FFD Algorithm and TPC-C Workload for 64 VMs and 16 PMs.

ition and the fact that MiyakoDori takes shorter time than normal pre-copy migration.

SAVED ENERGY RATIO

In both workloads, using MiyakoDori saved more than 25% of the data center energy consumption while using normal pre-copy migration saved 21–22%. Using normal pre-copy migration yields the same results with the two workload, because it have to transfer all the memory of the target VM no matter how the working set is smaller than the total memory usage (128 MB working set against 4 GB total memory usage in WebServer workload). When $C = 8$, i.e. a PM has 32 GB of memory, MiyakoDori saved around 3.9 points in both workloads compared to the normal pre-copy cases. As described for the results in Section 5.6.2, these reduction ratios are really valuable as the energy consumption of a data center is huge. When $C = \infty$, i.e. a PM has an infinite amount of memory, MiyakoDori saves more than when $C = 8$ because memory reusing is highly utilized with full memory images of migrating VMs. An important point is that even when $C = 8$ with no sophisticated memory image management, MiyakoDori saves valuable amount

TABLE. 5.3. Simulation Settings for Page cache teleportation

Parameter	Value
Number of PMs	**16**
Number of cores of a PM	4
Number of cores of a VM	1
Number of VMs	**64**
NW bandwidth between PMs	**1 Gbps** or 10 Gbps
Memory size of a VM	4 GB

of data center energy consumption.

ENERGY OVERHEAD

Energy overhead shows the same trend as in the results with the Warehouse-highpower algorithm: MiyakoDori reduces it a lot which in turns improves the Saved Energy Ratio. When MiyakoDori is used, the values are largely different in WebServer and TPC-C workload. This is because in TPC-C workload the working set size is larger than in WebServer workload, thus amount of reusable memory is less.

5.7 EVALUATING PAGE CACHE TELEPORTATION

5.7.1 OVERVIEW

Page cache teleportation is evaluated by integrated simulations with FFD algorithm. The simulation settings are described in Table 5.3. Parameter values that different from Table 5.2 is **displayed in bold fonts**. The PMs are connected with either 10 Gbps of **1 Gbps** bandwidth. The reason why 1 Gbps setting is introduced is that page cache teleportation has positive impact on the total migration time but not on the energy overhead of live migration. Therefore, in 10 Gbps it is predicted that page cache teleportation does not help reducint the energy consumption of data centers because live migration is already very quick with 10 Gbps bandwidth. Number of PMs and VMs are **16** and **64** respectively (c.f. 32 and 128 in MiyakoDori case) because dealing with 128VMs with 1 Gbps can slow down live migration so much that load change of a VM might happen during a migration of that VM. [*1]

[*1] Note that this thesis is NOT to propose VM relocation algorithm, thus merely avoiding exceptional cases is not a defect of this work

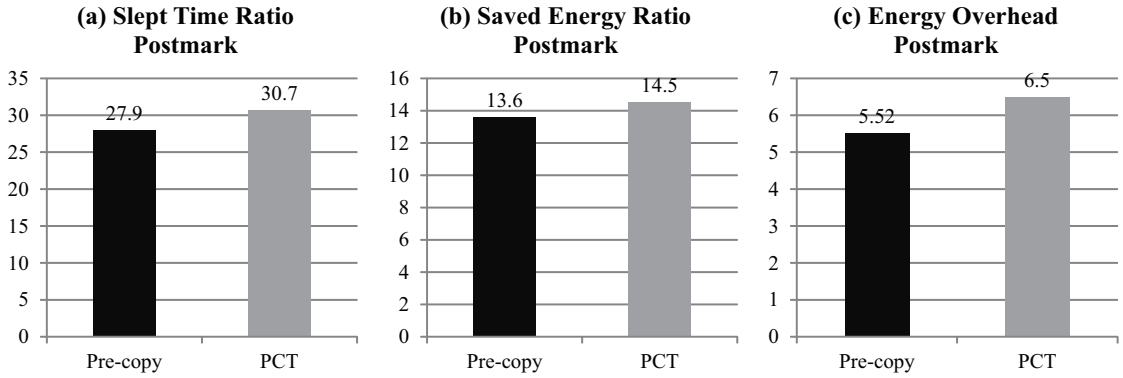


FIG. 5.11. Comparison of Page Cache Teleportation (denoted PCT in short) and Normal Pre-copy Live Migration under Postmark Workload with **1 Gbps** connections between PMs. Slept Time Ratio (Left), Saved Energy Ratio (Middle), and Energy Overhead (Right).

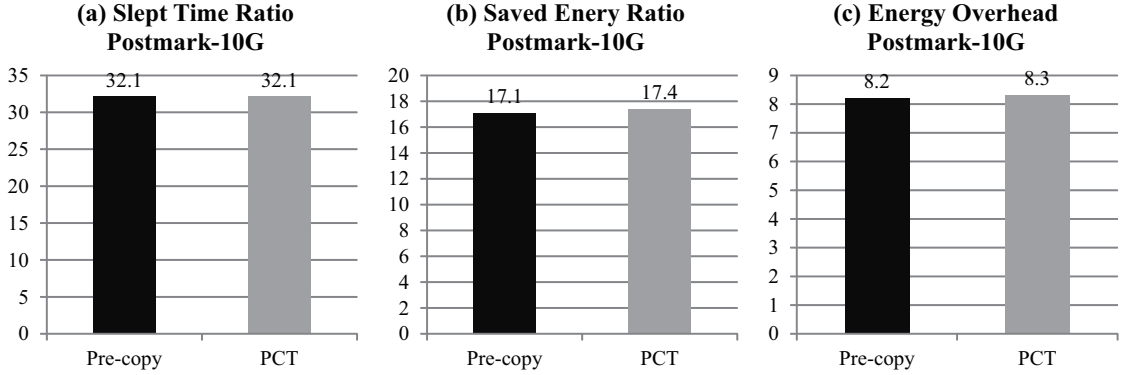


FIG. 5.12. Comparison of Page Cache Teleportation (denoted PCT in short) and Normal Pre-copy Live Migration under Postmark Workload with **10 Gbps** connections between PMs. Slept Time Ratio (Left), Saved Energy Ratio (Middle), and Energy Overhead (Right).

5.7.2 SIMULATION RESULTS WITH FFD

This section evaluates how page cache teleportation can reduce energy consumption of data centers compared to normal pre-copy live migration. An important notice is that page cache teleportation shortens total migration by utilizing two network links, but it does not reduce the amount of transferred memory. Thus, page cache teleportation has positive impact on the short sleep time, but it does not reduce energy overhead of live migration (both of which are explained in Section 5.2.1).

Figure 5.11 shows simulation results with page cache teleportation and normal pre-copy live migration. The number of VMs and PMs are 16 and 4 respectively, and the network bandwidth is 1 Gbps. The number of VMs/PMs are smaller than in MiyakoDori case because simulating a bigger data center in 1 Gbps may cause a situation that a migration takes longer time than a heavy load period when it coincides with migrations of

many other VMs. The left figure shows the slept time ratio, the middle figure shows the saved energy ratio, and the right figure shows the energy overhead. Here the network bandwidth is set smaller than in the simulation for MiyakoDori because short sleep time impacts the overall energy consumption when network within a data center is not very fast. Page cache teleportation is configure to take 34% shorter time and transfer the same amount of memory compared to normal pre-copy live migration. The ratio (34%) is defined based on a previous experiments shown in Table 4.2.

SLEPT TIME RATIO

The slept time ratio is 30.7% with page cache teleportation while it is 27.9% with normal pre-copy live migration. This is because PMs in the data center can become idle for longer time thanks to accelerated live migration by page cache teleportation.

SAVED ENERGY RATIO

The saved energy ratio is 14.5% with page cache teleportation and 13.6% with normal pre-copy live migration) This means that page cache teleportation saves 0.9% of total energy consumption of the simulated data center. The reason why the saved energy ratio is reduced only by 0.9% while slept time ratio is reduced by 3% is that PMs consume 20W of energy even in sleep mode. However, 0.9% is still a great reduction because the total energy consumption of a data center is huge and this reduction is from a sate of the art algorithm (FFD).

ENERGY OVERHEAD

Energy overhead is larger with page cache teleportation than with normal pre-copy live migration, but this phenomenon is as expected. This is because page cache teleportation does not reduce the amount of transferred memory, thus it neither reduces extra energy consumption of live migration. However, the total energy consumption is reduced as in Figure 5.11(b), thus energy overhead (extra energy consumption of live migration divided by total energy consumption of the data center) gets increased.

5.7.3 IO PENALTY OF PAGE CACHE DROPPING

A naïve counterpart of page cache teleportation that shortens total migration time of VMs with large page cache is to simply not transfer page cache at all and to make the VM re-load the data from storage after a migration. However, this method obviously

has significant IO performance penalty to the workload running on the VM. The fact a migrating VM has large page cache means that the workload running on the VM is IO-intensive, therefore this penalty must impact largely on the workload performance as well.

TABLE. 5.4. IO Performance Penalty of Page Cache Dropping to Postmark

	Normal	Dropped
Execution Time	565 sec	688 sec
Penalty	-	123 sec
Penalty / each drop	-	12.3 sec

Table 5.4 shows IO performance penalty to Postmark workload when page cache is dropped during the workload execution. Postmark benchmark is configured to execute 8000 transactions in the speed of at most 1024 per second with 1MB–5MB files and 4KB of read/write buffers. The benchmark is executed on the same machine as Table 4.1. "Normal" means that the benchmark is executed normally, and "Dropped" means that page cache of the machine is deleted 10 times with 60 seconds interval during the workload. The 1^{st} row shows execution time in each setting, the 2^{nd} row shows how dropping page cache prolonged the execution time, and the 3^{rd} row shows penalty per each page cache drop (note that page cache is dropped 10 times during the measurement to reduce jitters). The values are averaged over 10 runs in each setting.

Using the naïve method incurs large performance penalty to the workload running on the VM, therefore this thesis claims that it cannot be used for cloud data centers.[*2] For a concrete example, performance penalty to Postmark workload when two migrations occur within a workload execution is compared below:

NAÏVE CASE The workload execution is prolonged by $12.3 \times 2 = 24.6$ seconds, thus the total workload execution takes 589.6 seconds. This means 4% overhead compared to the original execution time is incurred by the two migrations.

PAGE CACHE TELEPORTATION CASE From the results in Chapter 4, a migration of VM running Postmark workload takes 17 seconds. Existing studies show that a migration degrades workload performance by 10%–30% [45, 96], therefore the two migrations prolong total execution time of the workload by $2 \times 17 \times 0.3 = 10.2$

[*2] Note that in terms only of energy consumption, it could result in smaller energy consumption thanks to shortened total migration time in some cases.

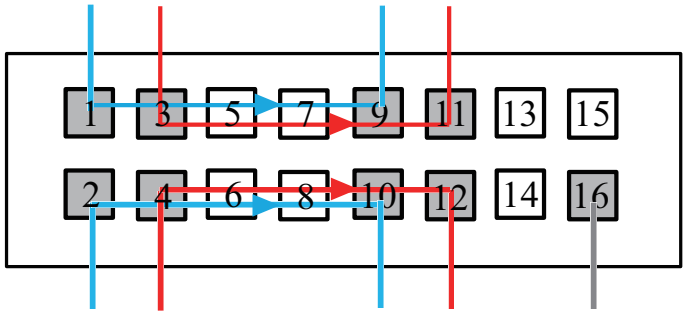


FIG. 5.13. Tested Switch Connections.

seconds at most (30% case). This penalty is less than the half of and is significantly smaller than in the naïve case.

To quantitatively discuss how these values are large/small with regard to service level agreements of clouds is preferable. However, it is unfortunately difficult as cloud providers do not claim service level agreements in a fine granularity. For example, the Amazon EC2 Service Level Agreement [97] promises "to make Amazon EC2 and Amazon EBS each available with a Monthly Uptime Percentage ... of at least 99.95%". It only promises how many percentage of time the cloud should be up, but does not say anything about performance degradation during the uptime due to sudden load change, resource starvation, or migrations.

5.8 DISCUSSION

5.8.1 ENERGY CONSUMPTION OF NETWORK SWITCHES

Aggressive VM relocation increases the amount of network traffic in the data center because of frequent live migration, thus energy consumption of network infrastructures might also negatively impact the energy saving. This section shows that it is not the case using a small experiment.

In order to show that network traffic increase does not affect the overall energy consumption of a data center, the relationship between energy consumption of an L2 network switch and its traffic is measured. Here an L2 switch is selected as a representative (not an L3 router or other network infractractures) because intra data center communications are mostly done within in the same segment built with many L2 switches. Note that the results in this section cannot be applied to a data center that uses software defined networks (SDN) such as OpenFlow.

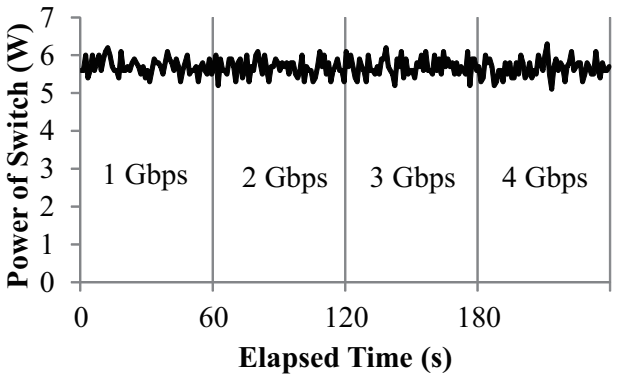


FIG. 5.14. Power of an L2 switch

Figure 5.13 illustrates an L2 switch and network connections made to it under the experiment. The switch is a Corega CO-BSW16GTX with 16 Gigabit ethernet ports. 8 servers are connected to Port 1, 2, 3, 4, 9, 10, 11, and 12 to generate heavy traffic, and another server is connected to Port 16 to control the 8 servers. The other 7 ports are not used in this experiment. The servers connected to Port 1, 2, 3, 4 are paired with the ones connected to Port 9, 10, 11, 12, respectively in terms of generated traffic (illustrated as connected lines in the figure). For example, the server connected to Port 1 sends data to to the server connected to Port 9.

Figure 5.14 shows the measured power of the L2 switch. The x-axis shows the elapsed time (seconds) from the beginning of the measurement, and the y-axis shows the power (Watts) of the L2 switch at each x. The four pairs of servers started exchanging data with 60 seconds interval; i.e. data is transferred between Port 1 and Port 9 during $0 \leq x \leq 60$, between Port 1 and Port 9, and Port 2 and Port 10 during $60 \leq x \leq 120$ etc. Each interval of the graph is denoted with the network traffic that the L2 switch is transferring (1 Gbps, 2 Gbps, 3 Gbps, and 4 Gbps). The actual throughput of each transfer was almost 1 Gbps, thus the denotation increases by just 1 Gbps for easy understanding.

The result strongly shows that the amount of traffic of an L2 switch does not affect the energy consumption of the switch. This in turn means that frequent live migration invoked by aggressive VM relocation does not increase the energy consumed by network infractractures in the data center without an SDN. Therefore, it can be concluded that energy consumption of network infrastructures can be ignored in the simulation conducted in this chapter.

5.8.2 ENERGY CONSUMPTION OF STORAGE SYSTEMS

Energy consumption of storage systems account for a non-negligible amount within a server and within a whole data center. Barroso *et al.* [92] report that disks consume 14.3% of energy of a data center, which is nearly equal to the amount consumed by cooling facilities (15.4%). Many researches have been done to reduce energy consumption of storage systems in data centers; Gurumurthi *et al.* [98] propose to dynamically control the rotating speed of HDDs to reduce the power for spinning the disks. Their main idea is "to dynamically modulate the speed at which the disk spins (RPM), thereby controlling the power expended in the spindle motor driving the platters" (cited from [98]). Zhu *et al.* [99] utilize this proposal to develop a power-aware cache management mechanism. They consider not only the number of misses but also the distribution pattern of the misses to efficiently switch rotating speed of HDDs.

Among the two efficient live migration methods proposed in this thesis, Page cache teleportation has relation to the load of storage nodes while MiyakoDori does not. In Page cache teleportation, transfer of restorable page cache via SAN requires read operations to HDDs in the storage nodes. Accesses to HDDs can increase the load of storage nodes and can also result in energy overhead. However, evaluating this energy overhead in general is difficult because access patterns to HDDs occurred by page cache teleportation largely depend on the actual data center in two aspects:

CACHE SYSTEM OF THE STORAGE NODES Number of read operations to the HDDs are not necessarily equal to the number of accesses to the restorable page cache. This is because storage nodes themselves have DRAMs and cache frequently accessed blocks. Therefore, evaluating energy overhead by storage nodes need to assume concrete physical and algorithmic aspects of the cache mechanism.

IO PATTERNS OF THE WORKLOAD Even if number of read operations to the HDDs are fixed, the next question is weather they require spinning up the HDDs or not because spinning up an HDD consumes much energy. Answering this question needs to assume/know the detailed IO patterns of the workloads running on a whole rack that shared the same storage nodes. An intuitive explanation is that under workloads with a great many of IO operations the HDDs are always spinning and no extra energy is consumed to spin up, but under workloads with few number of IO operations reading a block of restorable page cache always requires

spinning up.

5.9 RELATED WORK

Several studies have been done to model energy consumption of pre-copy live migration. Liu *et al.* shows extra energy consumption caused by the pre-copy live migration depends only on the amount of transferred [38]. Even though they figure out faster network bandwidth consumes much energy per time unit, the accumulated sum during the total migration time is shown to be the same. The paper gives detailed mathematical analysis but they never mention how the extra energy consumption impacts on the overall data center energy consumption. Aikema *et al.* compares how extra energy consumption of live migration changes depending on workload type and transport type of memory data (encrypted and non-encrypted) [59]. The paper concludes that conducting live migrations is "not always be advisable when looking to minimize power consumption", but they do not evaluate how much impact migrations have on the overall energy consumption.

Hossain *et al.* [60] proposes a dynamic VM consolidation algorithm that considers extra energy consumption of live migration. They reduce a data center's overall energy consumption by 12% compared to existing algorithms. Our novelty against this paper is that we show the energy overhead of both pre-copy and an accelerate live migration mechanism in detail, but they only show the overall energy saving given by their new algorithm. Goiri *et al.* [26] figures out cost of live migration must be considered as a penalty when conducting dynamic VM consolidation. The novelty of their algorithm is that it considers the time required to create a new VM and to migrate an existing VM when calculating energy-efficient VM placement. However, they do not care extra energy consumption of live migration that is focused on this paper. Borgetto *et al.* [61] figures out during a live migration the VM stays both on the source and the destination PM. They modify existing dynamic VM consolidation algorithms to more energy-aware ones by incorporating this idea, but they neither consider extra energy consumption of live migration incurred by memory and network load increase. Yang *et al.* [100] evaluates performance interference of co-located VMs, another overhead of VM consolidation besides live migration energy consumption. We believe negative impacts of VM consolidation must be cared more as done by us and Yang *et al*.

5.10 SUMMARY OF THIS CHAPTER

In this chapter, energy overhead of live migration within an aggressive VM relocation system is discussed and quantitatively evaluated using simulations. The core problem is these discussion and quantitative evaluation have never been done before although there are many researches on live migration optimization and VM relocation/consolidation algorithms. This chapter evaluates the two efficient live migration techniques given in this thesis, MiyakoDori and Page cache teleportation, in terms of their energy overhead within an aggressive VM relocation system. A cloud/grid simulator, SimGrid, is used and performance and energy models of each efficient live migration technique are implemented on SimGrid. The simulations showed that MiyakoDori and Page cache teleportation significantly reduce the energy overhead of live migration, and as a result they save the total energy consumption of a data center by several percent.

Future work includes two directions: (1) further analysis using the same models but different types of dynamic VM consolidation algorithms and real workload traces and (2) generalization of our methodology to other accelerated live migration mechanisms. The latter is more challenging because it requires modeling resource usage of them. Many of them use different types resources than the pre-copy live migration such as more storage IOs or multiple network interfaces.

CHAPTER 6

CONCLUSION

6.1 SUMMARY OF THIS THESIS

Cloud computing has become a common computing paradigm for both individual and enterprise uses, and energy consumption of cloud data centers has become huge in response to this trend.

Given this situation, the goal of this thesis is to achieve greener data center by aggressive VM relocation at runtime with efficient live migration. In order to make progress toward this goal, this thesis has three contributions:

1. First, this thesis discusses the requirements and the technical challenge to achieve the goal. The requirements include two factors; i.e. efficient live migration and integrated evaluation of live migration and aggressive VM relocation in terms of energy consumption. This thesis also figures out that the cost of live migration is still not fully reduced although there are many researches on efficient live migration. Two phases of memory transfer of the target VM are equally important with regard to the cost of live migration, but the 1^{st} phase (where all memory pages are transferred sequentially at first) is not yet well optimized.
2. Second, this thesis proposes two efficient live migration techniques, MiyakoDori and Page cache teleportation, to reduce the cost of live migration on the 1^{st} phase. MiyakoDori is based on an idea that VMs can migrate "back" to hosts where they have been executed before in a system that aggressively relocates VMs. MiyakoDori leverages this situation to reduce amount of tranferred memory and total migration time by reusing VMs' memory images on migrations "bask" and transferring only updated region of the VMs. Page cache teleportation mitigates a

problem that IO-intensive VMs have large amount of page cache that prolong live migration and has negative impact on the energy consumption of a data center. Page cache teleportation utilizes the storage area network in a data center to efficiently transfer this page cache in the background of normal memory transfer via the general purpose network.

3. Third, this thesis gives methodologies and results of integrated simulations of live migration and aggressilve VM relocation. Although there are many studies on dynamic VM consolidation (relocation) and live migration optimization, the former ignore or take little of live migration overhead and the latter evaluate their techniques only with a single migration. Therefore, this thesis is the first to conduct integrated simulations of them with regard to energy consumption. The simulation results show that the two mechanisms proposed in this thesis have positive impact on extra energy overhead and short sleep time, and they can save several percent of the overall energy of the simulated data center. Several percent is a significant improvement because the overall energy consumption of a data center is now extremely huge as mentioned in Chapter 1.

6.2 FUTURE PROSPECT

In this Section, future prospects of this thesis (Section 6.2.1) and the whole filed of live migration research are discussed (Section 6.2.2 and Section 6.2.3). The discussion for the research filed focuses on how live migration should evolve more to be used in real data centers and real society, which are based on the best of author's knowledge and belief from the experience.

6.2.1 FUTURE DIRECTION OF THIS WORK

A big goal that is directly connected from this work is to give methodology to evaluate energy aspect of live migration in a more general way than done in this thesis. This thesis analyzes real implementation of each live migration mechanism to evaluate the energy aspect. However, existing studies have proposed different types of migration mechanisms that use different types of resources, and also new studies continue appearing to make live migraiton more efficient. Data center operators cannot select the most suitable migration mechanism on their environment and workloads without evaluating

these mechanisms' energy aspects.

One possible direction to this goal is to build a more abstract model of live migration acceleration. Normal pre-copy live migration (without any acceleration) can be simply modeled by corporating memory transfer throughput and memory update speed. Based on this modeling, it might be possible to generally model acceleration methods of live migration as a function between extra resource usage and speed up against the pre-copy method. If this is possible, function between extra resource usage and extra energy consumption is easily given by existing studies on computer architectures, thus a function between speed up and extra energy consumption is given.

Another possible direction to this goal is to provide a simulation framework to achieve easy implementation and evaluation of a new live migration acceleration method. Current simulations in this thesis are implemented ad-hoc based on the deep knowledge of actual implementation of each acceleration method. However, data center operators do not have such knowledge nor time to do. It has never been done yet to emperically discuss on what are essential aspects to model various migration mechanisms easily but still detailed enough for evaluatin energy.

6.2.2 SECURE LIVE MIGRATION

Security is also a very imporant issue in cloud data centers as well as energy consumption. An unique characteristic of cloud security is that even the underlying PMs cannot be fully trusted because cloud providers can be malicious [101]. In contrast, for the traditional cluster or grids the owner and users of a cluster/grid are the same, or the ownes is a trustable organization such as an university or a well-known big company.

Many studies have been done to protect users data/programs/privacy from malicious cloud providers. Address space layout randomization (ASLR) [102] prevents introspection-based data leaking, Szefer *et al.* [103] introduce a new microprocessor that provents hypevisors from reading VM's memory freely, and Li *et al.* [104] enable VMs to run natively on the underlying PM with the ability to run multiple VMs on a single PM.

Live migration techniques must care more about this characteristic of cloud security. Current live migration techniques assume that all memory pages of the migrated VM are freely readable from the PM although this assumption does not hold for security-aware clouds/VMs.

6.2.3 HETEROGENEOUS LIVE MIGRATION

Heterogeneousity has been proven to play an important role for improve efficiency of computer systems. It is also becoming common in cloud computing and many researches have been done such as GPU-capable VMs [105, 106] to get better performance for HPC-like tasks, or hybrid of public and private clouds (or cloud orchestration) to get better security while maintaining scalability and low cost.

Live migration and dynamic VM consolidation are also catching this trend, but they are still on the way of evolving. As introduced in Chapter 2 (Related Work) heterogeneous ISA process migration [18] and heterogeneous hypervisor migration [57] have been approached, and also offloading tasks from high power x86 machines from low power ARM machines is proved to be promising [20]. There is one more step toward live migration a VM transparently from underlying ISA and hypervisor to achieve more energy efficienty clouds.

REFERENCES

[1] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori. KVM: the Linux Virtual Machine Monitor. In *Proceedings of the 2007 Linux Symposium*, pages 225 – 230, 2007.

[2] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. In *Proceedings of ACM Symposium on Operating Systems Principles (SOSP)*, pages 164–177, 2003.

[3] Stephen Soltesz, Herbert Pötzl, Marc E. Fiuczynski, Andy Bavier, and Larry Peterson. Container-based operating system virtualization: A scalable, high-performance alternative to hypervisors. *SIGOPS Operating Systems Reveviw*, 41(3):275–287, March 2007.

[4] Open VZ Linux Containers. `http://openvz.org/`.

[5] FreeBSD Handbook. `https://www.freebsd.org/doc/handbook/index.html`.

[6] AWS — Amazon Elastic Compute Cloud (EC2) - Scalable Cloud Hosting. `http://aws.amazon.com/ec2/`.

[7] Google App Engine - Google Developers. `https://developers.google.com/appengine/`.

[8] Salesforce1 Platform: Trusted Application Development Platform - Salfesforce.com. `http://www.salesforce.com/platform/`.

[9] Office. `http://office.microsoft.com/`.

[10] Cloud Applications — Cloud Solutions — SAP. `http://www.sap.com/pc/tech/cloud/software/cloud-applications/`.

[11] Salesforce, Google, Amazon Cloud Winners, Says Piper; Microsoft Straddles the Line. `http://blogs.barrons.com/techtraderdaily/2013/10/17/salesforce-google-amazon-cloud-winners-says-piper-microsoft-straddles-the-line/` *(accessed on Oct 23, 2014)*.

[12] Customer Success. Powered by the AWS Cloud. `http://aws.amazon.com/solutions/case-studies/` *(accessed on Oct 23, 2014)*.

[13] Jonathan G Koomey. Worldwide electricity used in data centers. *Environmental Research Letters*, 3(3):1–8, 2008.

[14] EPA Report on Server and Data Center Energy Efficiency. `https://www.energystar.gov/index.cfm?c=prod_development.server_efficiency_study`.

[15] Hong Xu and Baochun Li. Reducing electricity demand charge for data centers with partial execution. In *Proceedings of the International Conference on Future Energy Systems (e-Energy)*, pages 51–61, 2014.

[16] Akshat Verma, Gargi Dasgupta, Tapan Kumar Nayak, Pradipta De, and Ravi Kothari. Server workload analysis for power minimization using consolidation. In *Proceedings of the USENIX Annual Technical Conference (USENIX ATC)*, pages 1–14, 2009.

[17] Ming-Han Tsai, J. Chou, and Jye Chen. Prevent vm migration in virtualized clusters via deadline driven placement policy. In *Proceedings of 2013 IEEE 5th International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 599–606, 2013.

[18] M. DeVuyst, A. Venkat, and D. M. Tullsen. Execution migration in a heterogeneous-isa chip multiprocessor. In *Proceedings of Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 261–272, 2012.

[19] Hiroshi Endo, Hiroyoshi Kodama, Hiroyuki Fukuda, Toshio Sugimoto, Takashi Horie, and Masao Kondo. Cooperative control architecture of fan-less servers and fresh-air cooling in container servers for low power operation. In *Proceedings of Workshop on Power-Aware Computing and Systems (HotPower)*, pages 4:1–4:5, 2013.

[20] F. Durr. Improving the efficiency of cloud infrastructures with elastic tandem machines. In *Proceedings of IEEE Sixth International Conference on Cloud Computing (IEEE CLOUD)*, pages 91–98, 2013.

[21] Fereydoun Farrahi Moghaddam, Mohamed Cheriet, and Kim Khoa Nguyen. Low Carbon Virtual Private Clouds. In *Proceedings of the 2011 IEEE 4th International Conference on Cloud Computing (IEEE CLOUD)*, pages 259–266, 2011.

[22] Rahul Singh, David Irwin, Prashant Shenoy, and K.K. Ramakrishnan. Yank: Enabling green data centers to pull the plug. In *Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 143–155, 2013.

[23] Shekhar Srikantaiah, Aman Kansal, and Feng Zhao. Energy aware consolidation for cloud computing. In *Proceedings of Conference on Power Aware Computing and*

Systems (HotPower), pages 1–5, 2008.

[24] S. Lee, R. Panigrahy, and V. Prabhakaran, V. Ramasubramanian. Validating heuristics for virtual machines consolidation. Technical Report MSR-TR-2011-9, Microsoft Research, 2011.

[25] Zhibo Cao and Shoubin Dong. An energy-aware heuristic framework for virtual machine consolidation in cloud computing. *The Journal of Supercomputing*, 69(1):429–451, July 2014.

[26] I Goiri, F. Julia, R. Nou, J.L. Berral, J. Guitart, and J. Torres. Energy-aware scheduling in virtualized datacenters. In *Proceedings of IEEE International Conference on Cluster Computing (CLUSTER)*, pages 58–67, 2010.

[27] Min Yeol Lim, F. Rawson, T. Bletsch, and Vincent W. Freeh. Padd: Power aware domain distribution. In *Proceedings of the IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 239–247, 2009.

[28] Umesh Deshpande, Xiaoshuang Wang, and Kartik Gopalan. Live gang migration of virtual machines. In *Proceedings of the 20th International Symposium on High Performance Distributed Computing (HPDC)*, pages 135–146, 2011.

[29] Jui-Hao Chiang, Han-Lin Li, and Tzi-cker Chiueh. Introspection-based memory de-duplication and migration. *ACM SIGPLAN Notice*, 48(7):51–62, March 2013.

[30] Jihun Kim, Dongju Chae, Jangwoo Kim, and Jong Kim. Guide-copy: Fast and silent migration of virtual machine for datacenters. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC)*, pages 66:1–66:12, 2013.

[31] Petter Svärd, Benoit Hudzia, Johan Tordsson, and Erik Elmroth. Evaluation of delta compression techniques for efficient live migration of large virtual machines. In *Processdings of the 7th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE)*, pages 111–120, 2011.

[32] Xiang Zhang, Zhigang Huo, Jie Ma, and Dan Meng. Exploiting data deduplication to accelerate live virtual machine migration. In *Processdings of the 2010 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 88–96, 2010.

[33] Hai Jin, Li Deng, Song Wu, Xuanhua Shi, and Xiaodong Pan. Live virtual machine migration with adaptive, memory compression. In *Proceedings of the 2009 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 1–10, 2009.

[34] Yuyang Du, Hongliang Yu, Guangyu Shi, Jian Chen, and Weimin Zheng. Microwiper: Efficient memory propagation in live migration of virtual machines.

In *Proceedings of the 2010 39th International Conference on Parallel Processing (ICPP)*, pages 141–149, 2010.

[35] Andrey Mirkin, Alexey Kuznetsov, and Kir Kolyshkin. Containers checkpointing and live migration. In *Proceedings of the Linux Symposium, Volume 2*, pages 85–90, 2008.

[36] Michael R. Hines, Umesh Deshpande, and Kartik Gopalan. Post-copy live migration of virtual machines. *ACM SIGOPS Operating Systems Review*, 43(3):14–26, July 2009.

[37] Takahiro Hirofuchi, Hidemoto Nakada, Satoshi Itoh, and Satoshi Sekiguchi. Enabling instantaneous relocation of virtual machines with a lightweight vmm extension. In *Proceedings of the IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid)*, pages 73–83, 2010.

[38] Haikun Liu, Cheng-Zhong Xu, Hai Jin, Jiayu Gong, and Xiaofei Liao. Performance and energy modeling for live migration of virtual machines. In *Proceedings of 20th International Symposium on High Performance Distributed Computing (HPDC)*, pages 171–182, 2011.

[39] Balazs Gerofi and Yutaka Ishikawa. Workload adaptive checkpoint scheduling of virtual machine replication. In *Proceedings of the 2011 IEEE 17th Pacific Rim International Symposium on Dependable Computing (PRDC)*, pages 204–213, 2011.

[40] Khaled Z. Ibrahim, Steven Hofmeyr, Costin Iancu, and Eric Roman. Optimized pre-copy live migration for memory intensive applications. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, pages 40:1–40:11, 2011.

[41] Kejiang Ye, Xiaohong Jiang, Ran Ma, and Fengxi Yan. Vc-migration: Live migration of virtual clusters in the cloud. In *Proceedings of the 2012 ACM/IEEE 13th International Conference on Grid Computing (GRID)*, pages 209–218, 2012.

[42] Edwin Zhai, Gregory D Cummings, and Yaozu Dong. Live migration with pass-through device for linux vm. In *Processdings of the 2008 Linux Symposium*, volume 2, pages 261–268, 2008.

[43] Ryousei Takano, Hidemoto Nakada, Takahiro Hirofuchi, Yoshio Tanaka, and Tomohiro Kudoh. Ninja migration: An interconnect-transparent migration for heterogeneous data centers. In *Proceedings of 2013 IEEE International Symposium on Parallel & Distributed Processing (IPDPS), Workshops and Phd Forum*, pages 992–1000, 2013.

[44] Christoffer Dall and Jason Nieh. Kvm/arm: The design and implementation of the linux arm hypervisor. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 333–348, 2014.

[45] Michael Nelson, Beng-Hong Lim, and Greg Hutchins. Fast transparent migration for virtual machines. In *Proceedings of the Annual Conference on USENIX Annual Technical Conference (USENIX ATC)*, pages 391–394, 2005.

[46] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. Live migration of virtual machines. In *Proceedings of the 2nd USENIX Symposium on Networked Systems Design & Implementation (NSDI)*, pages 273–286, 2005.

[47] VMWare vSphere, Server Virtualization, Cloud Infrastructure. `http://www.vmware.com/products/datacenter-virtualization/vsphere/`.

[48] Oracle VM VirtualBox. `https://www.virtualbox.org/`.

[49] Constantine P. Sapuntzakis, Ramesh Chandra, Ben Pfaff, Jim Chow, Monica S. Lam, and Mendel Rosenblum. Optimizing the migration of virtual computers. *SIGOPS Operating Systems Review*, 36(SI):377–390, December 2002.

[50] Robert P. Goldberg. Survey of virtual machine research. *Computer*, 7(9):34–45, September 1974.

[51] Dejan S. Milojíčić, Fred Douglis, Yves Paindaveine, Richard Wheeler, and Songnian Zhou. Process migration. *ACM Computing Survey*, 32(3):241–299, September 2000.

[52] Peter M. Chen and Brian D. Noble. When virtual is better than real. In *Proceedings of the Eighth Workshop on Hot Topics in Operating Systems (HotOS)*, pages 133–137, 2001.

[53] Roy Bryant, Alexey Tumanov, Olga Irzak, Adin Scannell, Kaustubh Joshi, Matti Hiltunen, Andres Lagar-Cavilla, and Eyal de Lara. Kaleidoscope: Cloud micro-elasticity via vm state coloring. In *Proceedings of the Sixth Conference on Computer Systems (EuroSys)*, pages 273–286, 2011.

[54] PCI-SIG SR-IOV Primer An Introduction to SR-IOV Technology. `http://www.intel.com/content/dam/doc/application-note/pci-sig-sr-iov-primer-sr-iov-technology-paper.pdf`.

[55] John R. Lange and Peter Dinda. Symcall: Symbiotic virtualization through vmm-to-guest upcalls. In *Proceedings of the 7th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE)*, pages 193–204, 2011.

[56] Haikun Liu, Hai Jin, Xiaofei Liao, Chen Yu, and Cheng-Zhong Xu. Live virtual machine migration via asynchronous replication and state synchronization. *IEEE Transactions on Parallel and Distributed Systems*, 22(12):1986–1999, Dec 2011.

[57] Pengcheng Liu, Ziye Yang, Xiang Song, Yixun Zhou, Haibo Chen, and Binyu Zang. Heterogeneous live migration of virtual machines. In *Proceedings of International Workshop on Virtualization Technology (IWVT)*, pages 1–9, 2008.

[58] Kazushi Takahashi and Koichi Sasada. Winkvm : An alternative example of porting hypervisor (in japanese). *IPSJ Transactions on Advanced Computing Systems (ACS)*, 5(1):27–40, 2012.

[59] D. Aikema, A Mirtchovski, C. Kiddle, and R. Simmonds. Green cloud vm migration: Power use analysis. In *Proceedings of International Green Computing Conference (IGCC)*, pages 1–6, 2012.

[60] Mohammad M. Hossain, Jen-Cheng Huang, and Hsien-Hsin S. Lee. Migration energy-aware workload consolidation in enterprise clouds. In *Proceedings of the IEEE 5th International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 405–410, 2012.

[61] Damien Borgetto, Michael Maurer, Georges Da-Costa, Jean-Marc Pierson, and Ivona Brandic. Energy-efficient and sla-aware management of iaas clouds. In *Proceedings of the International Conference on Future Energy Systems (e-Energy)*, pages 25:1–25:10, 2012.

[62] Daniel Shelepov, Juan Carlos Saez Alcaide, Stacey Jeffery, Alexandra Fedorova, Nestor Perez, Zhi Feng Huang, Sergey Blagodurov, and Viren Kumar. Hass: a scheduler for heterogeneous multicore systems. *SIGOPS Operating Systems Review*, 43:66–75, April 2009.

[63] Vahid Kazempour, Ali Kamali, and Alexandra Fedorova. Aash: an asymmetry-aware scheduler for hypervisors. In *Proceedings of the International Conference on Virtual Execution Environments (VEE)*, pages 85–96, 2010.

[64] Theora.org. `http://www.theora.org/`.

[65] Viodeo LAN - VLC: Official site. `https://www.videolan.org/`.

[66] tpcc-mysql: Code: percona-tools. `https://code.launchpad.net/~percona-dev/perconatools/tpcc-mysql`.

[67] NAS PARALLEL BENCHMARKS. `http://www.nas.nasa.gov/Resources/Software/npb.html`.

[68] Hitesh Ballani, Keon Jang, Thomas Karagiannis, Changhoon Kim, Dinan Gu-

nawardena, and Greg O'Shea. Chatty tenants and the cloud network sharing problem. In *Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 171–184, 2013.

[69] Ankita Garg. Looking inside memory - tooling for tracing memory reference patterns. In *Proceedings of the 2010 Linux Symposium*, pages 63–74, 2010.

[70] Kazushi Takahashi, Kouichi Sasada, and Takahiro Hirofuchi. A fast virtual machine storage migration technique using data deduplication. In *Proceedings of the International Conference on Cloud Computing, GRIDs, and Virtualization (CLOUD-COMP)*, pages 57–64, 2012.

[71] Kazushi Takahashi. *A technique of virtual machine migration with high applicability*. PhD thesis, The University of Tokyo, 2013.

[72] Akane Koto, Hiroshi Yamada, Kei Ohmura, and Kenji Kono. Towards unobtrusive vm live migration for cloud computing platforms. In *Proceedings of the Asia-Pacific Workshop on Systems (APSys)*, 2012.

[73] Michael R. Hines and Kartik Gopalan. Post-copy based live virtual machine migration using adaptive pre-paging and dynamic self-ballooning. In *Proceedings of the International Conference on Virtual Execution Environments (VEE)*, pages 51–60, 2009.

[74] Changyeon Jo, Erik Gustafsson, Jeongseok Son, and Bernhard Egger. Efficient live migration of virtual machines using shared storage. In *Proceedings of International Conference on Virtual Execution Environments (VEE)*, pages 41–50, 2013.

[75] Cisco Systems, Inc. Data center: Load balancing data center. Technical report, 2004.

[76] S. Akiyama, T. Hirofuchi, R. Takano, and S. Honiden. Fast live migration with small io performance penalty by exploiting san in parallel. In *Proceedings of IEEE 7th International Conference on Cloud Computing (IEEE CLOUD)*, pages 40–47, 2014.

[77] J. Katcher. Postmark: A new file system benchmark. Technical Report TR3022, 1997.

[78] Vasily Tarasov, Saumitra Bhanage, Erez Zadok, and Margo Seltzer. Benchmarking file system benchmarking: It *is* rocket science. In *Proceedings of the 13th Workshop on Hot Topics in Operating Systems (HotOS)*, pages 1–5, 2011.

[79] TPC-C. `http://www.tpc.org/tpcc/`.

[80] William Voorsluys, James Broberg, Srikumar Venugopal, and Rajkumar Buyya. Cost of virtual machine live migration in clouds: A performance evaluation. In *Cloud Computing*, volume 5931 of *Lecture Notes in Computer Science*, pages 254–265. Springer Berlin Heidelberg, 2009.

[81] Xuxian Jiang, Xinyuan Wang, and Dongyan Xu. Stealthy malware detection through vmm-based "out-of-the-box" semantic view reconstruction. In *Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS)*, pages 128–138, 2007.

[82] Samer Al-Kiswany, Dinesh Subhraveti, Prasenjit Sarkar, and Matei Ripeanu. Vmflock: Virtual machine co-migration for the cloud. In *Proceedings of the International Symposium on High Performance Distributed Computing (HPDC)*, pages 159–170, 2011.

[83] M. Tsugawa, R. Figueiredo, J. Fortes, T. Hirofuchi, H. Nakada, and R. Takano. On the use of virtualization technologies to support uninterrupted it services. In *Proceedings of the IEEE ICC 2012 Workshop on Re-think ICT infrastructure designs and operations*, pages 7892–7896, 2012.

[84] DRBD. `http://www.drbd.org/`.

[85] Thomas Knauth and Christof Fetzer. dsync: Efficient block-wise synchronization of multi-gigabyte binary data. In *Proceedings of the 27th Large Installation System Administration Conference (LISA)*, pages 45–58, 2013.

[86] Changyeon Jo and Bernhard Egger. Optimizing live migration for virtual desktop clouds. In *Proceedings of the IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 1–8, 2013.

[87] S. Akiyama, T. Hirofuchi, R. Takano, and S. Honiden. Miyakodori: A memory reusing mechanism for dynamic vm consolidation. In *Proceedings of the IEEE International Conference on Cloud Computing (IEEE CLOUD)*, pages 606–613, 2012.

[88] SimGrid. `http://simgrid.gforge.inria.fr/`.

[89] SimGrid Git Repository. `git://scm.gforge.inria.fr/simgrid/simgrid.git`.

[90] T. Hirofuchi, A Lebre, and L. Pouilloux. Adding a live migration model into simgrid: One more step toward the simulation of infrastructure-as-a-service concerns. In *Proceedings of IEEE 5th International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 96–103, 2013.

[91] Anja Strunk. A lightweight model for estimating energy cost of live migration of virtual machines. In *Proceedings of the IEEE International Conference on Cloud Computing (IEEE CLOUD)*, pages 510–517, 2013.

[92] Luiz Andre Barroso, Jimmy Clidaras, and Urs Holzle. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Morgan & Claypool Publisher, 2nd edition edition, 2013.

[93] Jordi Arjona Aroca, Angelos Chatzipapas, Antonio Fernández Anta, and Vincenzo Mancuso. A measurement-based analysis of the energy consumption of data center servers. In *Proceedings of the International Conference on Future Energy Systems (e-Energy)*, pages 63–74, 2014.

[94] Mayank Mishra and Anirudha Sahoo. On theory of vm placement: Anomalies in existing methodologies and their mitigation using a novel vector based approach. In *Proceedings of the 2011 IEEE International Conference on Cloud Computing (IEEE CLOUD)*, pages 275 –282, 2011.

[95] Nguyen Trung Hieu, M. Di Francesco, and A.Y. Jaaski. A virtual machine placement algorithm for balanced resource utilization in cloud data centers. In *Proceedings of the 2014 IEEE 7th International Conference on Cloud Computing (IEEE CLOUD)*, pages 474–481, 2014.

[96] Kejiang Ye, Xiaohong Jiang, Dawei Huang, Jianhai Chen, and Bei Wang. Live migration of multiple virtual machines with resource reservation in cloud computing environments. In *Proceedings of the 2011 IEEE Conference on Cloud Computing (IEEE CLOUD)*, pages 267–274, 2011.

[97] Amazon EC2 Service Level Agreement. `http://aws.amazon.com/ec2/sla/`.

[98] Sudhanva Gurumurthi, Anand Sivasubramaniam, Mahmut Kandemir, and Hubertus Franke. Drpm: dynamic speed control for power management in server class disks. In *Proceedings of 30th Annual International Symposium on Computer Architecture (ISCA)*, pages 169–179, June 2003.

[99] Qingbo Zhu, Francis M. David, Christo F. Devaraj, Zhenmin Li, Yuanyuan Zhou, and Pei Cao. Reducing energy consumption of disk storage using power-aware cache management. In *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA)*, pages 118–118, Feb 2004.

[100] Renyu Yang, IS. Moreno, Jie Xu, and Tianyu Wo. An analysis of performance interference effects on energy-efficiency of virtualized cloud environments. In *Proceedings of the IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 112–119, 2013.

[101] Sören Bleikertz, Anil Kurmus, Zoltán A. Nagy, and Matthias Schunter. Secure cloud maintenance: Protecting workloads against insider attacks. In *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security (ASIACCS)*, pages 83–84, 2012.

[102] Hovav Shacham, Matthew Page, Ben Pfaff, Eu-Jin Goh, Nagendra Modadugu, and

Dan Boneh. On the effectiveness of address-space randomization. In *Proceedings of the 11th ACM Conference on Computer and Communications Security (CCS)*, pages 298–307, 2004.

[103] Jakub Szefer and Ruby B. Lee. Architectural support for hypervisor-secure virtualization. In *Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 437–450, 2012.

[104] Chunxiao Li, A. Raghunathan, and N.K. Jha. Secure virtual machine execution under an untrusted management os. In *Proceedings of the 2010 IEEE 3rd International Conference on Cloud Computing (IEEE CLOUD)*, pages 172–179, 2010.

[105] Giulio Giunta, Raffaele Montella, Giuseppe Agrillo, and Giuseppe Coviello. A gpgpu transparent virtualization component for high performance computing clouds. In *Proceedings of the 16th International Euro-Par Conference on Parallel Processing (EuroPar)*, pages 379–391, 2010.

[106] Yusuke Suzuki, Shinpei Kato, Hiroshi Yamada, and Kenji Kono. Gpuvm: Why not virtualizing gpus at the hypervisor? In *Proceedings of 2014 USENIX Annual Technical Conference (USENIX ATC)*, pages 109–120, 2014.