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　74–85 (Mar. 2012)

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

穐山 空道[1,a)]　広渕 崇宏[2]　高野 了成[2]　本位田 真一[1,3]

受付日 2011年7月19日, 採録日 2011年10月24日

概要：IaaS 型クラウドコンピューティングの効率を上げるため仮想マシンのライブマイグレーションを用いる研究が多く行われている．たとえば仮想マシンを負荷に応じて集約・分散することで仮想マシンの性能保証を行いつつクラウド全体の消費電力を下げる，似たメモリ内容を持つ仮想マシンを集めることでメモリ利用効率を向上させる，ユーザデスクトップ，計算ノード，データベースの間を仮想マシンが行き来することでアプリケーションの実行時間を短縮する等が行われている．これらの研究では仮想マシンがホスト間を何度も移動することにより積極的な最適化が可能である．従来のライブマイグレーションでは仮想マシン移動にかかる時間が長い，転送量が大きくネットワーク負荷が高いという問題がある．これに対し既存研究では 1 回限りの移動についてオーバヘッドを削減する試みが行われているが，繰り返し何度も移動する場合については研究されていない．そこで本稿では仮想マシンが 1 度実行されたことのあるホストに戻る場合に，過去の実行でのメモリを再利用する手法を提案する．メモリを再利用し変更された部分のみ転送することにより上記の問題点を解決する．本稿ではメモリの再利用を行って仮想マシンがライブマイグレーションするシステム，都鳥を開発した．マイクロベンチマークおよび apache を用いたアプリケーションベンチマークの結果，仮想マシンが 1 度実行されたことのあるホストに戻る場合に都鳥はライブマイグレーションにおける移動時間とメモリ転送量を削減できることが示された．

キーワード：仮想化，ライブマイグレーション，クラウドコンピューティング

MIYAKODORI: Optimization for Sequence of Live Migrations by Reusing VM Memory

Soramichi Akiyama[1,a)]　Takahiro Hirofuchi[2]　Ryousei Takano[2]　Shinichi Honiden[1,3]

Received: July 19, 2011, Accepted: October 24, 2011

Abstract: Many studies use live migration technique to increase efficiency of IaaS Cloud. For example, gathering and distributing Virtual Machines (VMs) in response to their load can guarantee the performance of VMs while suppressing energy consumption, memory usage can be more efficient by gathering VMs whose memory contents are similar with each other, and certain types of applications are executed faster by VM migration between user's desktop, compute nodes, and databases. In these studies, aggresive optimization are executed by making VMs migrate between hosts again and again. However, live migration have drawbacks that it takes a long time to migrate a VM, and that network load is large due to the large amount of transferred data. Existing studies about live migration tackle these problems by focusing on only 'a' migration, not migrations. In this paper, we propose Memory Reusing. With Memory Reusing, we can reuse the memory of a VM when it returns back to a host on which the VM have once executed. We developed a system called MIYAKODORI, where VMs can migrate with Memory Reusing. Micro benchmarks and application benchmarks showed that MIYAKODORI can reduce the amount of data to transfer and the time to take, when a VM migrates back.

Keywords: virtualization, live migration, cloud computing

1. 序論

IaaS 型クラウドコンピューティング（以下単にクラウドと呼ぶ）が様々な企業で利用されている [1]．それにともない仮想マシンの配置を動的に変更し最適化するシステムの

[1] 東京大学大学院情報理工学系研究科
Graduate School of Information Science and Technology, The University of Tokyo, Bunkyo, Tokyo 113–8656, Japan
[2] 独立行政法人産業技術総合研究所
National Institute of Advanced Industrial Science and Technology, Tsukuba, Ibaraki 305–8561, Japan
[3] 国立情報学研究所
National Institute of Informatics, Chiyoda, Tokyo 100–8430, Japan

[a)] akiyama@nii.ac.jp

研究が行われている．具体的に，我々は仮想マシンを負荷に応じて動的に配置変更し仮想マシンの性能保証を行いつつ IaaS クラウドシステムの消費電力を抑制する機構を開発している [2]．アイドル状態の仮想マシンが多数存在する際には少数の物理サーバ上に集約し，残りのサーバの電源を落とす．一方，仮想マシンの負荷が高まった際には複数のサーバ上に分散して配置し，仮想マシンの性能を保証する．そのほかにも似たメモリ内容を持つ仮想マシンを集約してメモリ利用効率を上げる研究 [3] や，仮想マシンがユーザデスクトップ，計算ノード，データベースを行き来することでアプリケーションの実行時間を短縮する研究 [4] 等がある．これらの研究において仮想マシンの移動にはライブマイグレーションを用いる．したがって，ライブマイグレーションは仮想マシンの動的な再配置を用いた最適化システムにおいて重要な技術である．

しかし仮想マシンのライブマイグレーションにはオーバヘッドが存在する．既存研究では pre-copy 型 [5] と post-copy 型 [6], [7] の 2 種類のライブマイグレーションが提案されている．両者に共通の問題点は，仮想マシンの全メモリを転送するため転送量が非常に多くなるという点である．また pre-copy 型に特有の問題点としてライブマイグレーションの完了に長い時間がかかるという点，post-copy 型に特有の問題点にはライブマイグレーション後に仮想マシンの性能が下がるという点がある．仮想マシンを動的に再配置するシステムではライブマイグレーションが何度も発生するため，これらの問題点を解決する必要がある．

ライブマイグレーションのオーバヘッド削減に関する研究 [6], [8], [9] は行われているが，すべて 1 回限りのライブマイグレーションに対する改善である．これらの研究はどんな場合のライブマイグレーションにでも適用できるという点では優れるが，一方で一般的な条件しか利用できないという欠点がある．本稿では仮想マシンが何度も移動するという条件を用い連続するライブマイグレーションに対して研究を行うことで，1 回限りのライブマイグレーションを考える場合からより進んだ最適化を目指す．本稿のように仮想マシンが何度も移動するという場合を考察した研究は我々の知る限り存在しない．

本稿では，ある仮想マシンが 1 度実行されたことのある物理マシンに再び戻る場合を考え，前回の実行から変更されていないメモリを再利用することで既存のライブマイグレーションの問題点を解決する手法を提案する．各メモリページごとに**世代**を管理してマイグレーション時に比較することにより，前回の実行から変更されていないメモリを再利用する．我々はメモリ再利用を行ってライブマイグレーションを行うシステム，都鳥[*1]を開発した．都鳥を用いて提案手法を評価した結果，従来のライブマイグレーションと比較してメモリ転送量，移動時間ともに削減できることが示された．

[*1] 都鳥は古典でユリカモメ（渡り鳥の一種）を指す．

2. ライブマイグレーション

既存のライブマイグレーション手法として，pre-copy 型と post-copy 型が提案されている．以下それぞれについて具体的に説明する．なおこれらの名称は文献 [6] に従っている．

2.1 pre-copy 型

pre-copy 型のライブマイグレーションは仮想マシンのライブマイグレーションとして文献 [5] で初めて提案されたものである．pre-copy 型は以下の手順で行われる．

(1) メモリを先頭から順にコピー．
(2) (1) のコピー中に書き換わったメモリを再度コピー．
(3) コピーすべき残りメモリが十分少なくなるまで (2) を繰り返す．
(4) 仮想 CPU を停止し，残りメモリをコピー．
(5) 移動先で仮想 CPU を再開．

本方式は仮想 CPU を移動する前にメモリをコピーする．すなわち (1)，(2) の間は計算の実行は移動元で行われている．現在 KVM や Xen 等実用化された仮想マシンモニタでサポートされているライブマイグレーションはすべて pre-copy 型である．

2.2 post-copy 型

post-copy 型のライブマイグレーションは文献 [6] において KVM で，文献 [7] において Xen で提案，実装された．post-copy 型は以下の手順で行われる．

(1) 仮想 CPU をコピー．
(2) 移動先で仮想 CPU を再開．
(3) メモリアクセスによって発生したページフォールトをとらえ，要求されたメモリページとその周辺のページをコピー．
(4) (3) と並列でメモリを先頭からコピー．

すなわち先に計算の実行を移動先に移し，必要なメモリをオンデマンドにコピーする方式である．pre-copy 型と異なりメモリをコピーする間の計算の実行は移動先で行われる．

2.3 従来のライブマイグレーションの問題点

本節では従来のライブマイグレーションを仮想マシンの動的な再配置という観点から述べる．pre-copy 型と post-copy 型に共通する問題点は，転送量が大きく仮想マシンが何度も，あるいは何台もライブマイグレーションする場合にネットワーク負荷が高くなることである．既存手法ではすべてのメモリを転送するため，たとえば仮想マシンの使用メモリが 1 GB あれば 1 GB ものデータを転送する必要

がある．仮想マシンを繰り返し動的に再配置するシステムにおいてはこの問題を解決しネットワーク負荷を軽減することは重要である．

また pre-copy 型に特有の問題点として，ライブマイグレーションが命令されてから実際に計算の実行が移動するまでに長い時間がかかるという点がある．これは 2.1 節で述べたメモリの繰返しコピーが原因である．pre-copy 型ではメモリをコピーする間は計算の実行は移動元で行われるため，メモリの繰返しコピーが何度も起きると長い時間仮想 CPU を移動することができない．したがって，仮想マシンを繰り返し動的に再配置するシステムでは，仮想マシンのメモリ更新速度によっては最適化が完了するまでに非常に長い時間を要する可能性がある．よって pre-copy 型における移動時間を短縮することは仮想マシンの動的な再配置システムの性能向上にとって不可欠である．

post-copy 型に特有の問題点には移動後に仮想マシンの性能が低下するという点がある [6]．これは移動直後にはメモリがコピーされておらずすべてのメモリアクセスがページフォールトを起こすためである．すべてのメモリのコピーが完了すれば性能低下はなくなるため，post-copy 型においてもメモリ転送量を削減することでこの問題を解決できる．

3. 提案手法：メモリ再利用

本稿ではある仮想マシンが 1 度実行されたことのある物理マシンに再び戻る場合に，前回の実行から変更されていないメモリを再利用する手法を提案する．以下本手法をメモリ再利用と呼ぶ．2 章で説明したように既存のライブマイグレーションではすべてのメモリをコピーする．また既存のライブマイグレーションに対する改善 [8], [9] では 1 回限りのライブマイグレーションを考えた最適化を行っている．それに対し本稿では仮想マシンが何度もホスト間を移動する場合を考えているという点が新しい着眼点である．

具体的には，仮想マシン V があるホスト A から別のホスト B へライブマイグレーションし，その後逆に B から A へライブマイグレーションすることを考える．このとき $A \to B$ の移動で V のメモリ内容を A に保存しておき，$B \to A$ の移動で保存されたメモリを再利用する．メモリ再利用を用いたライブマイグレーションのアルゴリズムは以下のようになる．

(1) 仮想マシン V は各メモリページごとカウンタ値を持つ．これをそのページの**世代**，全メモリページの世代の集合を**世代テーブル**と呼ぶ．世代は以下のように設定される．
  (a) V の起動時には全ページの世代は 0.
  (b) あるメモリページが更新されたとき，当該ホスト上での最初の更新に限り世代を 1 増やす．
(2) V がホスト A から別のホストへ移動するとき，V の全メモリページと世代テーブルを A のメモリ空間内に保持する．
(3) V が再び A に移動する際，メモリコピーを以下のように行う．
  (a) 当該メモリページについて A に保存されたページの世代と V の持つページの世代が一致すれば A のメモリから V のメモリへ直接コピーし再利用する．
  (b) そうでなければ通常のライブマイグレーションと同様にネットワーク越しにコピーする．

以上を模式図で表すと**図 1** のようになる．図の A，B はホストを，V はホスト上で実行される仮想マシンを表す．また各四角形はそのマシンのメモリを表す．1 段目で V が $A \to B$ とライブマイグレーションする際に，V の全メモリを A に保持する．2 段目は V が $B \to A$ とライブマイグレーションしている最中を表す．B での実行で更新されなかった部分（仮想マシン V のメモリのうち外側の四角）は再利用を行い，更新された部分（仮想マシン V のメモリのうち内側の小さな四角）はネットワーク越しに転送する．なお本手法は A からの移動先とその後の A への移動元が別のホストでも適用可能である．

本手法は 2 章で説明した pre-copy 型ライブマイグレーションの手順 (1) で行うメモリのコピーを簡略化する．従来手法では最初に仮想マシンのすべてのメモリをコピーす

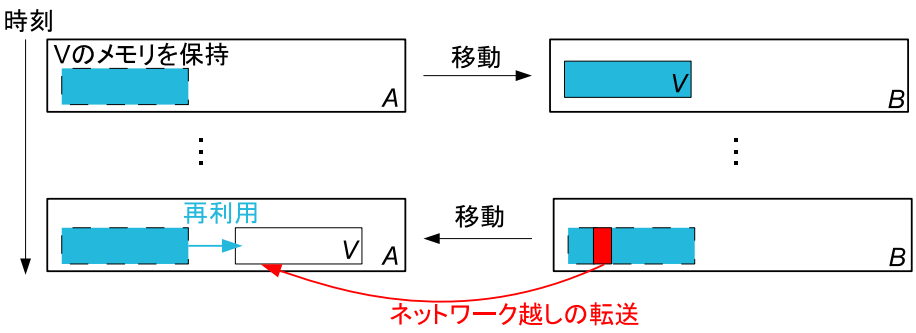

図 1　本手法の模式図
Fig. 1　Overview of proposed method.

る．それに対し本手法では再利用可能なページについてはコピーする必要がないため，移動先での前回の実行から更新されたメモリページのみをコピーする．コピー中に更新されたページを再度コピーする処理や，仮想 CPU を移動する処理は従来手法と同様である．また post-copy 型ライブマイグレーションでは従来は全メモリページがページフォールトを起こすのに対し，本手法では再利用不可能なページのみがページフォールトを起こす．すなわちいずれの手法においても再利用可能なページが事前にキャッシュされている状態となり，ライブマイグレーションにかかる通信量や移動時間を削減する．

4. 提案システム：都鳥

4.1 概要

提案手法を評価するため実際にシステムを構築した．本システムは仮想マシンがライブマイグレーションを行う際にメモリ再利用を行って転送量と移動時間を短縮する．仮想マシンと世代テーブルを結び付けて管理し，ライブマイグレーションの際に移動元と移動先の世代テーブルを自動的に比較しコピーすべきメモリページのみをコピーする．以後本システムを都鳥と呼ぶ．

4.2 実装の要件

提案手法を実装する際に満たすべき要件を述べる．

(1) 既存の仮想マシンモニタやハードウェアエミュレータに大きな変更を加えない．
(2) pre-copy 型と post-copy 型のいずれにも適用できる．

仮想マシンモニタはすでに多くのソフトウェアが実用化されており，既存の仮想マシンモニタに大きな変更を加える設計は提案手法を実用化する妨げとなる．したがって，提案手法を実用的なものにするために要件 (1) が必要である．また 2.3 節で述べたように既存のマイグレーションにはそれぞれ異なる問題点がある．そのため状況によって 2 つの手法を使い分けることが重要だと我々は考える．たとえば長い移動時間によって全体最適性が下がってでも個々の仮想マシンの性能を落としたくない場合には pre-copy 型が，逆に高い全体最適性を得たい場合には post-copy 型が適する．よって提案手法の実装では要件 (2) が必要である．

4.3 設計と実装

4.3.1 概要

都鳥の実装には仮想マシンモニタに KVM，ハードウェアエミュレータに QEMU を用い，これらに加えて世代テーブルを管理するためのプログラムを開発した．本プログラムを世代管理サーバと呼ぶ．世代管理サーバは QEMU からの接続を受け付けるサーバとして動作する．世代管理サーバを独立したプログラムとしたのは要件 (1) のためである．また pre-copy 型ライブマイグレーションには QEMU に標準で実装されているライブマイグレーションを，post-copy 型のライブマイグレーションには文献 [6] による実装を用いた．

世代テーブルは各仮想マシン，ホストごとに保持され，(仮想マシン，ホスト) の組で指定される．たとえば仮想マシン VM_0 がホスト B からホスト A にメモリを再利用してマイグレーションする際の手順は以下のようになる．

(1) 仮想マシン VM_0 の実行を一時停止する．
(2) ホスト B に仮想マシン VM_0 のメモリを保存する．
(3) ホスト B での実行で更新されたメモリページを表すビットマップ（dirty bitmap）を世代管理サーバに転送する．
(4) 世代管理サーバは dirty bitmap から (VM_0, B) を更新する．
(5) 世代管理サーバが (VM_0, A) と (VM_0, B) を比較する．
(6) 世代管理サーバが再利用可能なページを通知する．
(7) 仮想マシン VM_0 の実行を再開する．
(8) メモリ再利用をともなうライブマイグレーションを行う．

この様子を表したものが**図 2** の右図である．図中の数字は上記 (1) から (8) に対応する．ホスト A からホスト B への初めてのマイグレーションのようにメモリの再利用ができない場合は (5)，(6) は省略される．この様子は図 2 の左図に表されている．

なお仮想マシン，ホストの表現には QEMU の持つ uuid およびホストの IP アドレスを使用した．

4.3.2 詳細

コピーすべきページを QEMU に通知する方法は具体的なマイグレーション手法に pre-copy 型と post-copy 型のいずれを使うかで異なる．pre-copy 型では移動元から移動先に対してページを送信する．したがって，移動元に送信しなくてもよいページ，すなわち 2 つの世代テーブルで世代が同じであるページ番号を通知する．QEMU は pre-copy 型での繰返しコピーに対応するため全メモリページに関して dirty flag というフラグを持っており，フラグが dirty なページを転送する実装になっている．このフラグを変更して再利用不可能なページのみ dirty にすることでメモリ再利用を実現した．一方 post-copy 型では移動先がオンデマンドにページを取得する．したがって，移動先に取得すべきページ，すわなち 2 つの世代テーブルで世代が異なるページ番号を通知する．文献 [6] による実装では仮想マシンのページフォールトを vmem driver と呼ばれるドライバおよび vmem process と呼ばれるユーザ空間プロセスで処理し post-copy 型を実現している．本稿では vmem process を改変し再利用可能なページをネットワーク越しではなくホストのメモリ空間内からコピーすることを実現した．

世代には 4 バイトの符号なし整数を用いた．3 章で述べた本手法の手順から，世代はマイグレーション 1 回に最大

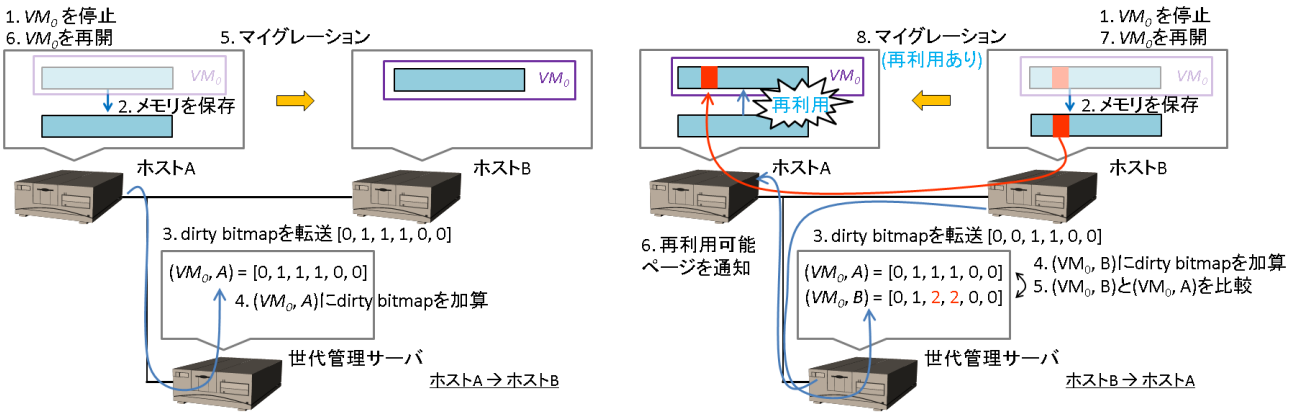

図 2　都鳥の動作．左図がホスト A からホスト B への 1 回目のマイグレーションを表す．これは再利用するメモリがまだないため再利用できない場合である．右図がホスト B からホスト A に戻るマイグレーションを表す．こちらは再利用が有効にはたらく場合である．右図の詳細な動作は以下のようになる．1. メモリと世代を保存するため仮想マシンを一時停止する．2. 仮想マシンのメモリをホスト B に保存する．3. ホスト B での実行で書き込みがあったページを表すビットマップ（dirty bitmap）を世代管理サーバに転送する．4. 世代管理サーバはビットマップから世代テーブル (VM_0, ホスト B) を更新する．5. 世代管理サーバは世代テーブル (VM_0, ホスト A) と (VM_0, ホスト B) を比較する．6. 比較の結果世代が同じだったページは再利用可能．当該ページを表すビットマップを転送する．7. 仮想マシンを再開する．8. ライブマイグレーションを行う．ただし再利用可能なページは移動先にすでに存在するためコピーしなくてよい

Fig. 2　How MIYAKODORI works: Left side illustrates first migration from host A to host B, where the memory is not reused because there is no cache yet. Right side illustrates when the VM migrates back from host B to host A, where the memory is reused. Detailed description of right side is as follows: 1. Stop the VM to save the memory and the generation table. 2. Save memory of the VM into host B. 3. A dirty bitmap including pages that have been updated during the execution on host B is transferred to generation management server. 4. Generation management server updated generation table (VM_0, host$B$) using the dirty bitmap. 5. Generation management server compares generation tables (VM_0, host$A$) and (VM_0, host$B$). 6. Pages that have the same generation are reusable. A bitmap including these pages are transferred from generation management server. 7. The VM is resumed. 8. Execute migration, where reusable pages are not copied via network.

1 しか増えないためこれで実用上十分な大きさである．また仮想マシンのメモリが 1 GB あると仮定すると，x86 および x64 のページサイズは 4 KB であるから世代テーブルの大きさは 1 GB ÷ 4 KB/page × 4 B/page = 1 MB となる．これは仮想マシンのメモリサイズに比べて十分小さく，実用上問題ない．

本設計において既存のシステムへの変更は，再利用によりコピーしなくてよいページ番号を既存のシステムに伝える機能の追加である．世代テーブルの管理等これ以外の機能はすべて別プロセスのサーバで行う．ページ番号の通知についても，pre-copy 型の場合については前段落で述べたように既存のシステムが持つ表の値を変更するだけで可能である．また post-copy 型についてはコピー処理が仮想マシンモニタおよびハードウェアエミュレータの外で実装されているため，コピー処理に変更を加えても仮想マシンモニタ等を変更する必要がない．したがって，本設計での既存のシステムへの変更点は十分小さく，要件 (1) を満たす．また本手法はメモリのうちどこをコピーするか，あるいはしないかを定める手法でありメモリのコピー方式とは独立である．したがって，pre-copy 型にも post-copy 型にも適用でき要件 (2) を満たす．

5. 評価実験

メモリ再利用の有用性を確認するため，都鳥を用いて評価実験を行った．

5.1 評価項目

評価はシステムの動作確認，マイクロベンチマーク，アプリケーションベンチマークにより行った．以下それぞれの詳細を述べる．

動作確認　都鳥の動作確認を行った．2台のホストと1台のサーバを用い実際に仮想マシンがメモリを再利用しながらホスト間を行き来できることを確認した．

マイクロベンチマーク　都鳥の基本的な性能評価を行った．指定された大きさのメモリ領域を確保し，各メモリページの先頭に値を書く動作を領域の全ページにわたって行った．メモリ転送量および移動時間がどの程度削減できるかを，様々なメモリの書き込み速度に対して確認した．

アプリケーションベンチマーク　実際のアプリケーションを用いた評価を行った．本稿ではapacheとhttperf[10]を用い，静的なwebページの読み込みタスク中にマイグレーションを行ってどの程度再利用できるかを評価した．

5.2　評価環境

評価では動作確認およびマイクロベンチマークには実際のクラスタを用い，アプリケーションベンチマークにはシミュレーション環境を用いた．シミュレーション環境を用いた理由はクラスタに用いたマシンではKVM 2.6.32のバグが存在したためである．実際にはページの書き換えが起こっていないのにページが書き換わったと判定される場合が確認された．ただし動作確認およびマイクロベンチマークについてはシミュレーション環境での結果および理論値と一致したためクラスタでの結果を掲載した．また本バグはKVM 2.6.38では修正されていることが確認できた．

クラスタでは2台のホストA，Bおよび世代管理サーバを配置するホスト1台の合計3台を用いた．サーバのハードウェアスペックおよびソフトウェアのバージョンは**表 1**のとおりである．シミュレーション環境では1台のマシンに移動元，移動先，都鳥のすべてを配置した．ハードウェアスペックおよびソフトウェアのバージョンは**表 2**のとおりである．

いずれの環境でもゲストOSにはUbuntu 10.10 serverを用い，メモリは1 GBを割り当てた．

5.3　評価結果

5.3.1　システムの動作確認

はじめに，実装したシステムの動作確認を行った．ライブマイグレーションにはpre-copy型を用い，都鳥を使う場合と使わない場合でマイグレーション完了までのメモリ転送量を比較した．なおpost-copy型を用いた場合についても同様の実験を行い正しく動作することを確認したが，紙面の都合上post-copy型に関する結果は省略しアプリケーションベンチマークで取り上げる．実験の手順は以下のとおりである．

(1) ログインプロンプトが出た状態でホストAからホストBにライブマイグレーションを行う．
(2) 仮想マシンでログインを行う．
(3) ホストBからホストAにライブマイグレーションを行う．

$A \rightarrow B$のライブマイグレーションでは都鳥のあるなしにかかわらず同じ結果になり，$B \rightarrow A$でのライブマイグレーションではメモリが再利用できるため都鳥を用いるとメモリ転送量が削減されるはずである．

実験結果を**表 3**に示す．数値は3回の実行の平均をとった．都鳥なしの$A \rightarrow B$，$B \rightarrow A$および都鳥ありの$A \rightarrow B$がほぼ同じ値を示した．これはメモリ再利用をしない，あるいは初めてのマイグレーションなので再利用できない場合であり，全使用メモリが転送されているからである．なお仮想マシンの割当てメモリ量よりも実際に転送されたメモリ量が少ないのはQEMUに同じバイトが連続するページはその1バイトのみを代表して送る機能があるためである．未使用のメモリページは全バイト0であるためこの機能によって送信が省かれている．

一方都鳥ありの$B \rightarrow A$ではメモリ転送量を約87%削減できた．これはメモリ再利用によって変更されていないメモリページを転送していないからである．また変更のあるページに関しては従来どおり転送しているため，ライブマイグレーション後には正しくログイン後の状態でAに戻る

表 1　クラスタのハードウェアおよびソフトウェア仕様
Table 1　Hardware and sofware specifications of the cluster.

CPU	Intel Xeon X5460 (4 cores)
Memory	8 GB
Disk	250 GB HDD
Network	1 GB Ethernet NIC
OS	Debian GNU/Linux 6.0
kernel	Linux 2.6.32
QEMU	0.13.0

表 2　シミュレーション環境のハードウェアおよびソフトウェア仕様
Table 2　Hardware and sofware specifications of the simulation environment.

CPU	AMD Phenom II X4 955 (4 cores)
Memory	8 GB
Disk	2 TB HDD
Network	(使用しない)
OS	Debian GNU/Linux 6.0
kernel	Linux 2.6.32
QEMU	0.13.0

表 3　都鳥あり/なしでのメモリ転送量
Table 3　Amount of transferred data with/without MIYAKODORI.

マイグレーション	都鳥なし	都鳥あり
$A \rightarrow B$（再利用不可）	137 MB	141 MB
$B \rightarrow A$（再利用可）	138 MB	18 MB

ことが確認された．

以上から提案手法のメモリ再利用および提案システムの都鳥は正しく動作していることが確認できた．

5.3.2　マイクロベンチマーク

次に再利用がどの程度可能なのかを調べるため，マイクロベンチマークによる評価を行った．ベンチマークには指定された大きさのメモリを確保し各メモリページの先頭に値を書き込んでいくプログラムを用いた．提案手法ではメモリの再利用可能判定はメモリページに書き込みがあったかどうかによって判断しているため，メモリページの先頭に値が書き込まれると再利用できない．そこで本プログラムで書き込み速度を変えてどの程度再利用ができるかを調べた．

具体的にはベンチマークプログラムには 500 MB を指定し，以下の手順で実験を行った．なお post-copy 型は移動中のメモリ更新がないマイグレーションであり，pre-copy 型でメモリ書き込みが遅い場合と同様の傾向を示す．そこで本項では post-copy 型のマイクロベンチマークの結果は省略し pre-copy 型の結果のみ示す．

(1) A 上で確保した領域にランダムデータを書き込む．
(2) $A \rightarrow B$ とライブマイグレーションを行う．
(3) B 上で確保した領域のページを指定した速さで書き換える．
(4) 10 秒後に $B \rightarrow A$ とライブマイグレーションを行う．

手順 (4) でのメモリ転送量および移動時間を都鳥ありとなしで比較した．またメモリの書き込み速度は 200 ページ/秒，1,000 ページ/秒，5,000 ページ/秒の 3 種類を行った．

実験結果を**図 3** に示す．図 3 の左図はメモリ書き込み速度とライブマイグレーションで転送されたメモリの関係である．200 ページ/秒と 1,000 ページ/秒では差が少ないが 5,000 ページ/秒で大きく転送量が増えているのは，5,000 ページ/秒ではメモリ書き込みが速すぎて繰返しコピーが発生しているからである．なお書き込み速度を 10,000 ページ/秒にすると書き込み速度がコピー速度を超えてしまいマイグレーションを行うことができなかった．都鳥なしと都鳥ありを比較すると，後者の方がメモリ転送量が減っており，特にメモリ書き込み速度が遅い場合で削減が顕著である．これは書き込み速度が遅いとメモリのほとんどが再利用できるからである．一方繰返しコピーが発生するような状況では再利用可能なメモリ量が少ないためメモリ転送量の削減は 5 割程度にとどまっている．

図 3 の右図はメモリ書き込み速度とライブマイグレーションにかかった時間の関係である．ライブマイグレーションにかかる時間はほぼすべてメモリコピーにかかる時間であるため，メモリ転送量のグラフと同じ傾向を示した．

以上のマイクロベンチマークにより本稿の提案はライブマイグレーションにおけるメモリ転送量および移動時間を削減できることが確認された．

5.3.3　アプリケーションベンチマーク

最後に実際のアプリケーションによるベンチマークを行った．ベンチマークには apache および httperf [10] を用い，静的な web ページの読み込みをタスクとした．httperf は web サーバの性能を測るベンチマークツールであり，1 秒間に生成するコネクション数やアクセスするファイルを指定したワークロードの作成が可能である．

具体的には 256 KB のファイルを 1,024 個用意し apache の管理ディレクトリ下に設置した．それらのファイルに対し httperf を用いて以下の条件でアクセスした．

(1) 100 コネクション/秒．
(2) 1,024 個のファイルを 1 番目から順に取得．
(3) 10,240 コネクションで終了．

アクセス開始後 30 秒で移動元から移動先にライブマイグレーションし，さらに 30 秒で移動元に戻るライブマイグレーションを行った．100 コネクション/秒で 10,240 コネクションに到達するには約 100 秒かかるため，ベンチマークの終了時点は元のホストに戻ってくる時点よりも後である．なお apache および httperf はともに移動する仮想マシ

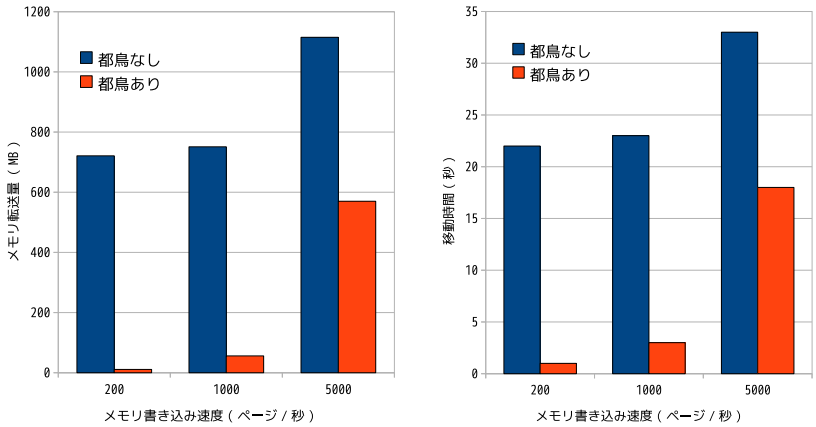

図 3　メモリ書き込み速度とメモリ転送量/移動時間の関係

Fig. 3　Amount of transferred data and total migration time with memory update frequency.

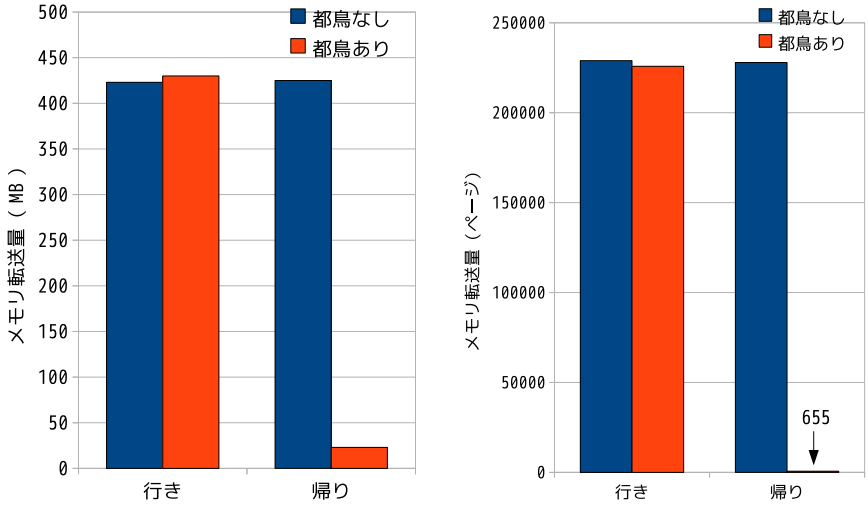

図 4　pre-copy 型/post-copy 型でのアプリケーションベンチマーク結果

Fig. 4　Results of application benchmarks with pre-copy and post-copy live migration.

ン上で動作させた．

図 4 の左図が pre-copy 型での結果，右図が post-copy 型での結果である．数値は 3 回の実行の平均をとった．post-copy 型でのメモリ転送量とは，ページフォールトの処理とは独立に行われるメモリの先頭からのコピーで転送された量である．図中の行きとは開始後 30 秒でのライブマイグレーションを，帰りとはさらに 30 秒後でのライブマイグレーションを表す．いずれの図も都鳥なしの行き，帰りおよび都鳥ありの行きがほぼ同じ値を示している．これは動作確認で述べたのと同様に 1 回目のライブマイグレーションではメモリが移動先にないためメモリ再利用ができないためである．

一方都鳥ありの帰りでは pre-copy 型，post-copy 型ともに転送量が大きく減っている．これは web ページの読み込み等のファイルアクセスでは 1 度アクセスされたファイルがメモリ上にキャッシュされるため，2 度目以降のアクセスではメモリを読み込むだけでよいからである．本ベンチマークでは 1,024 個のファイルに対し毎秒 100 コネクションで順番にアクセスするため，約 10 秒の時点でファイルがすべてキャッシュされる．実際に書き換わったメモリページの数を調べたところ約 10 秒間増え続けその後ほとんど増加しないことが確認された．最初の 10 秒間でのメモリ書き込み速度は 7,000 ページ/秒程度，それ以降のメモリ書き込み速度は 300 ページ/秒程度であった．したがって，帰りではほとんどすべてのメモリが再利用できるためメモリ転送量が大きく削減できている．なお post-copy 型の場合に pre-copy 型よりも削減率が大きいのは，post-copy 型の実装には同一バイトが連続するページを送信しない機能がないため再利用なしでは未使用のメモリも転送されているからである．未使用のページは更新されないため再利用すると転送されない．また pre-copy 型においてはマイクロベンチマークで示したとおり移動時間とメモリ転送量は同様の傾向を示す．実際に本実験においてもメモリと同様の削減が移動時間についても見られた．

以上でメモリ再利用および都鳥が実アプリケーションにおいてもライブマイグレーションにおける転送量と移動時間を削減できることが示された．

6. 議論

本章では都鳥の適用範囲と保存されたメモリの管理，および都鳥によるオーバヘッドについて論じる．

6.1 都鳥の適用可能なシステム

都鳥は仮想マシンが 1 度実行されたことのあるホストに何度も戻るようなクラスタに適用可能である．これは本研究の前提として序論で述べたものである．たとえば我々の開発している仮想マシンパッキングシステム [2] が該当する．本システムは多くのメモリを持ちアイドル状態の仮想マシンを集約する倉庫サーバと，アクティブな仮想マシンを 1 台のみ実行する占有サーバを持つ．ある仮想マシンの負荷が高まれば倉庫サーバから占有サーバへ，逆に負荷が下がれば占有サーバから倉庫サーバへマイグレーションする．すなわちクラスタ全体の負荷の少ない状態では多くの仮想マシンが倉庫サーバへ集約され，逆に負荷の高い状態では多くの占有サーバへ分散される．これにより低負荷時に仮想マシンを実行していない占有サーバをサスペンド可能にし消費エネルギーを削減する．本システムで実環境をエミュレーションしたワークロードを実行したところ，1 時間あたり 22 回から 30 回のマイグレーションが行われた [2]．

都鳥は 1 度実行した仮想マシンのメモリを保存することにより高速化を行うため，ホストやクラスタ全体のメモリ使用率が非常に高いシステムでは再利用が有効にはたらかない．ただし，

(1) 都鳥によるメモリ消費を削減する工夫は可能である．
(2) メモリに余裕がないシステムに適用した場合でも，従来のライブマイグレーションと同等の性能を示す．

(1) については次節で詳しく述べる．(2) の場合には再利用のために保存したメモリをすべて解放すればよい．この場合再利用による高速化は得られず都鳥によるオーバヘッドのみ残るが，6.3 節に示すようにオーバヘッドは十分小さいといえる．

また節電のためホストの電源を落としてしまう場合には，完全に電源を落とすのではなくサスペンドにする等の対策で適用が可能である．

6.2 保存されたメモリの管理

都鳥によりライブマイグレーションにおける移動時間が短縮されるためには，再利用されるメモリがホスト間のネットワークよりも読み出しの速い記憶装置に保存されることが必要である．現在の実装ではある仮想マシンのメモリは RAM ディスク上に全ページを含むファイルとして保持される．すなわち 1 GB のメモリを持つ仮想マシンを 1 度実行したホストは，当該仮想マシンのメモリを保持するために 1 GB のメモリを消費する．主記憶の容量は限られているため，保存されたメモリから不要と判断されたページを解放することが望ましい．

保存されたメモリは任意のページごとに解放が可能である．これは以下の理由による．

(1) 世代の管理とマイグレーションにおけるメモリのコピーがページごとに行われる．
(2) Linux では RAM ディスク上のファイルに割り当てられたメモリは *madvise*(2) を用いてページごとの解放が可能である．

(2) の実装は QEMU に少量の変更を加えるだけで可能である．また *madvice* を用いたメモリページの解放処理のコストも大きくないと考えられる．これはガーベジコレクションとは異なりページ間の依存関係をたどる等の処理が必要ないからである．すなわち保存されたメモリページの解放は実装上・実用上ともに小さなコストで実現できる．

ページを解放するには以下の手順が必要である．ただし通信の説明はページ解放を判断するモジュールを世代管理サーバと同一のノードに設置した場合のものである．

(1) どのメモリページを解放するか判断する．たとえば最近使用されていないメモリページは不要と判断する．
(2) 世代管理サーバが，解放すべきと判断されたページの世代に無効を表す特別な値（たとえば世代の最大値）を設定する．
(3) 世代管理サーバが当該ページを保持しているホストに対して当該ページを解放するように通知する．
(4) 通知を受けたホストが当該ページに割り当てられたメモリを解放する．

(3) で発生する通信のコストはメモリ再利用における再利用可能ページの通知と同程度のコストで済み，十分小さい．詳しい値については 6.3.3 項を参照されたい．(1) で行うどのメモリページを解放するかの判断についてはホストやクラスタのメモリ使用率の計測や予測が必要となるため，より進んだ枠組みとして今後取り組む予定である．

6.3 都鳥によるオーバヘッド

6.3.1 ページ書き換え判定による影響

メモリページが書き換わったかどうかの判定には QEMU の持つ dirty page tracking という機能を用いた．本機能は x86 システムにおいてメモリページへの書き込みがあった際にページテーブルに書き込まれる dirty フラグをユーザプロセスである QEMU からも見えるようにしたものである．本機能は通常では pre-copy 型でのライブマイグレーションにおいて繰返しコピー時に書き換わったメモリページを追跡するために使われるが，都鳥ではライブマイグレーション時以外もつねに有効にしてメモリページへの書き込みを監視している．

本機能をつねに有効にすることによるオーバヘッドを NAS Parallel Benchmarks [11] の memory-bound ベンチマークである CG で見積もった．本機能が有効な仮想マシンと無効な仮想マシンで CG の実行時間を比較したところ，差が確認されなかった．また実行中の CPU 使用率についても差は見られなかった．よって dirty page tracking を有効にすることによるアプリケーション実行時間への影響は十分小さいといえる．なお CG でのメモリ書き込み速度は 2,000 ページ/秒程度であった．

ただし真にメモリ書き込みのみを行うようなマイクロベンチマークでは差が見られた．各ページの先頭に 1 バイトずつ書き込みを行うプログラムで書き込み速度を調べたところ，dity page traking を有効にすると 1 回目の書き込みに限り約 20%の性能低下が確認された．しかし各ページに 1 度しか書き込まないという動作は一般のアプリケーションでは少ないと考えられるため，この性能低下は問題ではない．

6.3.2 仮想マシン停止による影響

4.3.1 項に示したように，都鳥ではメモリの保存と世代テーブルの更新のために仮想マシンを一瞬停止する．停止中は仮想マシン上のアプリケーションも停止するためアプリケーションの実行に影響する．停止中に実行される処理は以下である．

(1) 仮想マシンのメモリを実行中のホストへ保存する．
(2) dirty bitmap を世代管理サーバへ転送する．

(1) について仮想マシンのメモリを RAM ディスク上のファイルとして実装しているため，当該ファイルにつねに最新のメモリ状態が存在する．すなわち実際には保存にかかるコストは 0 である．(2) については世代テーブ

ルの比較は単純にどのページの世代が異なるかを比較するだけでよいためページ数 n に対して $O(n)$ で済む．仮想マシンのメモリサイズが 1 GB の場合，x86 システムの標準ではページサイズが 4 KB であるためページ総数は 1 GB ÷ 4 KB/page = 250 Kpageである．したがって，$n = 250{,}000$ 程度となりこの比較にかかる時間は無視できるほど短い．また dirty bitmap を転送するコストについては 6.3.3 項で述べるが，これも無視できるほど小さい．以上の議論により仮想マシン停止による影響は実用上十分短いといえる．

6.3.3　ネットワークへの影響

世代の同期および再利用可能ページの通知によるネットワーク転送量は十分に小さい．前者は当該実行で更新されたページ，後者は再利用可能なページを表すビットマップを転送する．仮想マシンのメモリサイズが 1 GB のときページ総数は 6.3.2 項に示したように 250 Kpage となる．すなわちビットマップのサイズは 250 Kbits÷8 bits/byte ≈ 30 KB となり，十分小さいといえる．

6.3.4　仮想マシンのメモリを保持することによる影響

仮想マシンを 1 度実行したホストでは再利用のために当該仮想マシンのメモリを保持しておく必要がある．現在の実装では，ある仮想マシンのメモリを保持しているホストは，当該仮想マシンに割り当てられたメモリサイズ分のメモリを消費する．しかし 6.2 節で述べたように今後は不要と判断したメモリを解放可能にする予定である．

また CPU 時間については仮想マシンのメモリを保持することによるコストは 0 である．これは 4.3.1 項に述べたように世代テーブルに関する処理はライブマイグレーション開始時に移動元ホストと世代管理サーバの間で実行されるからである．すなわち仮想マシンのメモリを保持しているだけでマイグレーションを実行中ではないホストでは世代を更新する等の処理は行われない．

7.　関連研究

ライブマイグレーションの方式を工夫しオーバヘッドを減らす研究はいくつか存在する．文献 [9] では pre-copy 型の繰返しコピーの段階において前回コピーとの差分をとることで転送量を削減している．一般にメモリページが書き換わるとき，ページ内の全ビットが書き換わることはあまりない．この特性から，あるページを繰返しコピーする際には前回コピーしたページ内容とビットごとの xor をとると多くの場合に 0 が長く続く列が得られる．これを run-length 符号化で圧縮することによって転送量を削減し，繰返しコピーにかかる時間を減らす．run-length は簡単な符号化方式のため負荷が小さく，また同じビットが長く続く場合には十分な圧縮効率を得ることができる．この手法は 1 回限りのライブマイグレーションに対する最適化であり本研究とは観点が異なる．ただし差分のみ転送する手法として本研究と組み合わせて適用することは可能である．

文献 [8] では本稿と同じく仮想マシンのメモリページを一様に扱うことが問題であるとしている．本文献は文献 [7] を発展させたものである．文献 [7] では仮想マシンのコピーにおいて post-copy 型のメモリコピー方式をとることにより瞬時に仮想マシンをコピーすることを実現している．メモリをオンデマンドにコピーする際に要求されたページから連続したページをまとめてプリフェッチするが，ホストの物理アドレス上で連続していても仮想マシンの物理アドレス上で連続しているとは限らない．そこで本文献では仮想マシンのページテーブルにある実行可能ビット，ユーザビットとゲスト OS の持つ情報を見ることにより，各メモリページをユーザコード/カーネルコード/ユーザデータ/カーネルデータ/ファイルキャッシュ/空きにクラスタリングする．各クラスタ内で連続するページをプリフェッチすることにより，プリフェッチの精度を上げて post-copy 型で発生するページフォルトによる性能低下を抑えている．

このように単一のライブマイグレーションにおいてオーバヘッドを減らす研究は存在する．本研究は単一のマイグレーションではなく連続するマイグレーションを考えている点でこれらの研究とは異なる．

本研究はクラスタにおいて仮想マシンを効率良くマイグレーションする手法であり，マイグレーションをいつ・どこへ行うかについては対象としていない．最適化のために仮想マシンをいつ・どこへマイグレーションするかを対象とした研究には序論で述べたもの [2], [3], [4] のほかに，たとえば文献 [12] があげられる．本文献ではクラスタ内の負荷を監視・予測し，負荷の高いホストから低いホストへ仮想マシンをマイグレーションすることでアプリケーションの性能低下を防ぐ．負荷の監視・予測には Black-box と Gray-box の 2 通りの戦略が使われる．Black-box 戦略はディスク使用率やスワップの使用率等仮想マシンの外側から取得できる情報のみを用いる．Gray-box 戦略は仮想マシンの OS 内にプログラムを設置し Black-box 戦略に加えてメモリの使用率やアプリケーションのログが利用できる．Black-box 戦略は不特定多数の利用するクラウド等に適し，Gray-box 戦略はクラスタ管理者と利用者が同じであるシステムに適する．本文献では移動する仮想マシンを選ぶ際にマイグレーションのオーバヘッドを削減する工夫をしている．負荷が高いほど大きくなる値 *volume* を定義し，ある仮想マシンの *volume* を当該仮想マシンの割当てメモリ量で割る．この商が最も大きい，すなわち *volume* あたりの割当てメモリ量が最も小さい仮想マシンをマイグレーションする．これは単位 *volume* を分散するために転送するメモリ転送量が最も小さくなるように仮想マシンを選ぶことに相当する．本文献に都鳥を適用する場合，移動先のサーバを選ぶアルゴリズムを変更することが考えられる．本文献では最も *volume* の低いサーバを選んでいるが，

たとえば上記計算で仮想マシンの割当てメモリ量から再利用可能なメモリ量を引くことで再利用可能なサーバに戻りやすくできると考えられる．

8. 結論

本稿ではクラウドの効率化にライブマイグレーションが用いられることを述べ，従来のライブマイグレーションでは移動時間やメモリ転送量が問題となることを指摘した．

それに対し本稿では仮想マシンが1度実行されたことのある物理マシンにライブマイグレーションで戻ってくる際にメモリの再利用を行うことでライブマイグレーションのオーバヘッドを削減するシステム，都鳥を開発した．マイクロベンチマークおよびアプリケーションベンチマークの結果，都鳥によってライブマイグレーションでのメモリ転送量や移動時間が大きく削減できることが示された．

今後の課題は以下である．

動的なワークロードでのアプリケーションベンチマーク　本稿ではアプリケーションベンチマークとして静的な web ページの読み込みをタスクとした．5 章で説明したとおり同一ファイルの 2 度目以降の読み込みはキャッシュからの読み込みになるためメモリの書き換えが起こらない．そのため本評価は提案手法にとって理想的な場合であるといえる．したがって，ベンチマーク実行中にメモリが書き換わるような場合についても評価が必要である．

再利用の粒度　本稿の提案手法では各メモリページごとに再利用を行う．そのためメモリページが 1 bit でも変更されると再利用できない．たとえば wikipedia や twitter 等で利用されているキャッシュシステムである memcached [13] では，古いデータを捨てるために最終アクセス時刻を保持している．このような場合にはデータ自体は更新されていなくても同一ページに格納されたアクセス時刻が更新されるだけで当該ページは再利用できなくなる．実際，我々の実験では memcached に格納されたデータを読み込むだけで多くのメモリページが更新されることが観察された．データの読み込みに対しても最終アクセス時刻を更新するという要求は多くあると考えられるため，このような場合に対応することは重要である．簡単には再利用の粒度を小さくすることが考えられるが，粒度を $\frac{1}{n}$ にすると世代テーブルのサイズは n 倍になるためこれに対処する工夫も必要である．

実際の最適化手法との組合せ　序論で紹介した仮想マシンの動的な再配置を行うシステムと本稿での提案を組み合わせた評価が必要である．本稿では実際の再配置・最適化手法とは独立に提案手法の有用性を評価した．今後は実際の運用において本稿で対象としたようなライブマイグレーションの連続がどの程度発生するか等の評価が必要である．

謝辞　本研究は科研費（23700048）および CREST（情報システムの超低消費電力化を目指した技術革新と統合化技術）の支援を一部受けた．

参考文献

[1] amazon web services 導入事例，入手先 ⟨http://aws.amazon.com/jp/solutions/case-studies/⟩.
[2] Hirofuchi, T., Nakada, H., Itoh, S. and Sekiguchi, S.: Reactive consolidation of virtual machines enabled by postcopy live migration, *Virtualized Technologies in Distributed Computing*, pp.11–18 (2011).
[3] Wood, T., Tarasuk-Levin, G., Shenoy, P., Desnoyers, P., Cecchet, E. and Corner, M.D.: Memory buddies: Exploiting page sharing for smart colocation in virtualized data centers, *Virtual Execution Environment*, pp.31–40 (2009).
[4] Lagar-Cavilla, H.A., Tolia, N., Lara, E., Satyanarayanan, M. and O'Hallaron, D.: Interactive Resource-Intensive Applications Made Easy, *Middleware*, pp.143–163 (2007).
[5] Clark, C., Fraser, K., Hand, S., Hansen, G.J., Jul, E., Limpach, C., Prat, I. and Warfiel, A.: Live Migration of Virtual Machines, *Networked Systems Design and Implementation*, pp.273–286 (2005).
[6] Hirofuchi, T., Nakada, H., Itoh, S. and Sekiguchi, S.: Enabling Instantaneous Relocation of Virtual Machines with a Lightweight VMM Extension, *Cluster, Cloud and Grid Computing*, pp.73–83 (2010).
[7] Lagar-Cavilla, H.A., Whitney, J.A., Scannell, A., Patchin, P., Rumble, S.M., Lara, E., Brudno, M. and Satyanarayanan, M.: SnowFlock: Rapid Virtual Machine Cloning for Cloud Computing, *EuroSys*, pp.1–12 (2009).
[8] Bryant, R., Tumanov, A., Irzak, O., Scannell, A., Joshi, K., Hiltunen, M., Lagar-Cavilla, H.A. and Lara, E.: Kaleidoscope: Cloud Micro-Elasticity via VM State Coloring, *EuroSys*, pp.273–286 (2011).
[9] Svärd, P., Hudzia, B., Tordsson, J. and Elmroth, E.: Evaluation of delta compression techniques for efficient live migration of large virtual machines, *Virtual Execution Environment*, pp.111–120 (2011).
[10] Mosberger, D. and Jin, T.: httperf: A Tool for Measuring Web Server Performance, *Performance Evaluation Review*, Vol.26, No.3, pp.31–37 (1998).
[11] NAS PARALLEL BENCHMARKS, available from ⟨http://www.nas.nasa.gov/Resources/Software/npb.html⟩.
[12] Wood, T., Shenoy, P., Venkataramani, A. and Yousif, M.: Black-box and Gray-box Strategies for Virtual Machine Migration, *Networked Systems Design and Implementation*, pp.229–242 (2007).
[13] memcached, available from ⟨http://memcached.org/⟩.

穐山 空道　（学生会員）

2010 年京都大学工学部情報学科卒業．同年より東京大学大学院情報理工学系研究科修士課程，現在に至る．仮想化技術，システムソフトウェア等に興味を持つ．

広渕 崇宏　（正会員）

2001 年京都大学理学部地球物理学教室卒業．2002 年同大学大学院理学研究科地球惑星科学専攻修士課程退学．2007 年奈良先端科学技術大学院大学情報科学研究科博士課程修了．同年独立行政法人産業技術総合研究所入所．現在，情報技術研究部門研究員．グリッドコンピューティング，クラウドコンピューティング，Green IT の研究に従事．システムソフトウェア，ネットワーク，仮想化技術等に興味を持つ．博士（工学）．

高野 了成　（正会員）

2005 年東京農工大学大学院電子情報工学専攻博士後期課程単位取得退学．2003 年（株）アックス入社．2008 年より独立行政法人産業技術総合研究所情報技術研究部門研究員．博士（工学）．オペレーティングシステム，分散並列環境における高性能計算に関する研究開発に従事．ACM，IEEE，USENIX 各会員．

本位田 真一　（フェロー）

1978 年早稲田大学大学院理工学研究科修士課程修了．（株）東芝を経て 2000 年より国立情報学研究所教授，2004 年より同研究所アーキテクチャ科学研究系研究主幹を併任，現在に至る．2008 年より同研究所先端ソフトウェア工学・国際研究センター長を併任，現在に至る．2001 年より東京大学大学院情報理工学系研究科教授を兼任，現在に至る．現在，早稲田大学客員教授，英国 UCL 客員教授を兼任．2005 年度パリ第 6 大学招聘教授．工学博士（早稲田大学）．1986 年度情報処理学会論文賞受賞．日本ソフトウェア科学会理事，情報処理学会理事を歴任．ACM 日本支部会計幹事，日本ソフトウェア科学会編集委員長，FIT2009 プログラム委員長，日本学術会議連携会員．