情報処理学会論文誌 Vol.56 No.2 516–524 (Feb. 2015)

更新履歴に基づいたメモリページ転送順序スケジューリングによる仮想マシンライブマイグレーションの高速化

中居 新太郎[1,†1] 川島 龍太[1,a)] 齋藤 彰一[1,b)] 松尾 啓志[1,c)]

受付日 2014年5月14日, 採録日 2014年11月10日

概要：仮想マシンライブマイグレーションの一般的な実現方法である pre-copy 型ライブマイグレーションでは，仮想マシンを一時的に停止させる前に大部分のメモリページを転送することで停止期間を短縮化する．しかし，メモリページの転送は仮想マシンが動作した状態で行われるため，転送済みのページが更新され，メモリページの再送が頻発する場合がある．これまで，更新をともなう可能性の高いメモリページを推定し，ページの転送順序を変更する手法などが提案されているが，仮想マシンの一時停止期間が長くなるという新たな問題が発生する．そこで本研究では，各メモリページにおける更新履歴を基に，段階分けされたページ転送優先度を定義し，総ページ転送量を増やすことなく仮想マシンの一時停止期間を短縮する手法を提案する．さらに，拡張ページテーブル（EPT）に着目したメモリアクセスプロファイリングにより，仮想マシン動作に対する性能オーバヘッドをほぼ無視できる手法も提案する．提案手法を QEMU/KVM 上に実装し評価を行った結果，ライブマイグレーションにおける仮想マシンの中断時間が削減できることを確認した．

キーワード：サーバ仮想化，ライブマイグレーション，クラウドコンピューティング，QEMU，KVM，EPT

Improving the Performance of Virtual Machine Live Migration by Ordering Memory Page Transfer Based on Update History

Shintaro Nakai[1,†1] Ryota Kawashima[1,a)] Shoichi Saito[1,b)] Hiroshi Matsuo[1,c)]

Received: May 14, 2014, Accepted: November 10, 2014

Abstract: Currently, live migration of virtual machines is based on a pre-copy approach that transfers most memory pages to the destination host in order to shorten the paused period. But, retransmission of pages can occur for updates because virtual machines are running during the memory transfer time and this results in increase of total migration period. Many studies have focused on this problem so far, such as frequently-updated pages are stopped to transfer during the running state. However, such approach prolongs the paused period and user experience gets worse. In this paper, we propose an improved approach that defines staged page transfer ordering based on page update history, and our method can reduce the paused period without increasing the total amount of page transfer. In addition, we further propose a memory access profiling method using Extended Page Table (EPT) that minimizes performance overhead of running virtual machines. The experimental results showed the proposed method reduced the retransmissions and stopped period.

Keywords: server virtualization, live migration, cloud computing, QEMU, KVM, EPT

[1] 名古屋工業大学
Nagoya Institute of Technology, Nagoya, Aichi 466–8555, Japan
[†1] 現在，株式会社インテック
Presently with INTEC Inc.
[a)] kawa1983@nitech.ac.jp
[b)] shoichi@nitech.ac.jp
[c)] matsuo@nitech.ac.jp

1. はじめに

現在，Amazon EC2 [1] などの Infrastructure-as-a-Service（IaaS）型クラウドサービスを提供するデータセンタでは，計算機資源の利用状況がつねに変化するため，仮想マシンの初期配置時には予測できない負荷の集中や偏りが発生す

る場合がある．そのため，ライブマイグレーションによって，実行中の仮想マシンを別の物理マシン上に移動させることで，各物理マシンに掛かる負荷を動的に均衡化させる手法が広く用いられている [2], [3], [4], [5]．

仮想マシンのライブマイグレーション方式には，主に pre-copy 型 [6] と post-copy 型 [7] の 2 種類が存在する．pre-copy 型のライブマイグレーションは，仮想マシンを一時停止させる前に，必要なメモリページの大部分を転送する手法で，停止期間を短縮することができるが，仮想マシンが動作した状態でページ転送を行うため，転送済みページの更新および再転送が頻発する場合がある．post-copy 型の場合は，ライブマイグレーション中にはページ転送を行わず，仮想マシンの動作再開後に，必要に応じてページ転送を行う．pre-copy 型の課題であったメモリページの再転送を防ぐことが可能であるが，マイグレーション後の仮想マシンの動作性能が不安定になるため，現状ではあまり使用されていない．そこで本研究では，KVM [8] や VMware [9] などの仮想マシンモニタにおいて標準的に利用されている，pre-copy 型のライブマイグレーションを対象とする．

これまで，pre-copy 型ライブマイグレーションにおける問題を解決するため，様々な手法が提案されてきた．Hu ら [10] は，更新される可能性の高いメモリページ（high dirty pages）をあらかじめ推定しておき，pre-copy イテレーションの最終ラウンドでこれらのページを転送することで再転送を抑制する手法を提案している．しかし，ページの使用状況によっては仮想マシン停止期間（stop-and-copy）中のページ転送量が増大し，停止期間が長くなる場合がある．

そこで本稿では，各メモリページにおける更新履歴を基に，段階分けされたページ転送順序を新たに定義し，ページの転送量を増やすことなく仮想マシンの一時停止期間を短縮する手法を提案する．本提案手法では，ライブマイグレーション開始時に，過去のメモリアクセスプロファイリングの結果から更新される可能性の高いメモリページを推定し，更新される可能性のより高いメモリページほど，マイグレーションの後工程で転送することによってページの再転送を抑制し，かつ仮想マシンの停止期間を短縮化する．筆者らは，仮想マシンによるメモリアクセスを拡張ページテーブル（EPT）を利用して低オーバヘッドでプロファイリングする仕組みと，ライブマイグレーション時におけるメモリページの転送順序を変更する機能を仮想マシンモニタの 1 つである QEMU/KVM に対して実装した．提案手法を評価した結果，QEMU/KVM 本来のライブマイグレーション実装と比較して，ライブマイグレーション時における仮想マシンの一時停止期間と総ページ転送量を削減できた．

以下，2 章では，pre-copy 型ライブマイグレーションの概要と関連研究について，3 章では提案手法のアルゴリズムについて，4 章では QEMU/KVM 上での提案手法の実装について説明する．5 章で性能評価を行った後，6 章でまとめと今後の課題を示す．

2. 研究背景

2.1 pre-copy 型ライブマイグレーション

KVM や VMware などの代表的な仮想マシンモニタでは，ライブマイグレーションのアルゴリズムとして，pre-copy 型を使用している．ここで，一般的な pre-copy 型ライブマイグレーションの処理手順を以下に示す．

(1) pre-copy
仮想マシンを動作させた状態でメモリページを先頭から順に転送する．転送済みのページが更新された場合は，再転送を行う．転送すべき残りメモリページ数が十分少なくなるまでこの動作を繰り返す．本稿では，この一連の動作を pre-copy イテレーションと呼ぶ．pre-copy イテレーションが一定回数行われると，残りメモリページ数にかかわらず，強制的に (2) の stop-and-copy 処理へ移る．

(2) stop-and-copy
仮想マシンの動作を一時的に停止し，転送すべき残りメモリページと仮想 CPU などの各種デバイスに関する情報を転送する．

(3) restart
転送先の物理マシン上で仮想マシンの動作を再開する．

2.2 pre-copy 型ライブマイグレーションの問題点

既存の pre-copy 型ライブマイグレーションでは，仮想マシンを停止させる前に，転送すべきメモリページの大部分を転送しておくことで，仮想マシン停止期間中に転送するメモリページ数を削減している．しかし，仮想マシンが動作した状態でメモリページの転送を行うため，転送済みのメモリページが更新される可能性がある．その結果，同一のメモリページが何度も転送されることになり，転送量の増大によるライブマイグレーション実行時間の長期化を招くという問題がある．

2.3 関連研究

pre-copy 型ライブマイグレーションにおけるメモリページの再送問題に着目し，全体のデータ転送量を削減する研究として，文献 [10], [11] があげられる．前者の手法は pre-copy における最終イテレーションでのみ，更新される可能性の高いメモリページを転送する．そのため，pre-copy における転送量を削減可能だが，stop-and-copy における転送量が増加するという問題点がある．一方，後者の手法は各 pre-copy イテレーションの終了時に更新されたメモリページを検出し，次回以降のイテレーションでなるべく転送しないように，メモリページの転送順序を変更する．

またメモリページを再送する場合は，差分データのみを転送する．この手法では，各 pre-copy イテレーションでプロファイリングと差分符号化処理を行うため，CPU リソース消費が大きくなる恐れがある．

また，仮想マシンが物理マシン間を往来する状況を想定した研究として文献 [12] がある．この手法では，仮想マシンのメモリ内容を転送元の物理マシンが保持し，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際にその内容を再利用する．しかし，物理マシンのメモリの消費量が増加するという問題点がある．

文献 [13] では，メモリページ間の重複を取り除く手法として，MDD（Migration with Data Deduplication）を提案している．MDD はフィンガープリント技術を用いて重複を検出し，転送済みページとの差分を圧縮化することで転送量を削減している．しかし，メモリページが頻繁に更新される状況下では，計算オーバヘッドの増大が問題となる．

3. 提案手法

本研究では，ライブマイグレーション開始時に，更新される可能性の高いメモリページを過去の更新履歴を基に推定し，可能性の低い順にページ転送を行う手法（メモリページの転送順序スケジューリング手法）を提案する．また，仮想マシンのゲスト物理アドレスとホストマシンの物理アドレス変換に使用する拡張ページテーブルに着目した，低コストでメモリアクセス情報を取得できるメモリアクセスプロファイリング方法も提案する．拡張ページテーブルには Intel の Extended Page Tables（EPT）[14] と AMD の Rapid Virtual Indexing（RVI）[15] が存在するが，本稿では拡張ページテーブルを EPT と総称する．

3.1 転送順序の変更

一般的な pre-copy 型のライブマイグレーションでは，メモリページのアクセスパターンを考慮せず，先頭のメモリページから順に転送する．そのため，メモリを頻繁に更新するアプリケーションが動作する仮想マシンを移動する場合，メモリページの再転送が頻発する可能性が高くなる．そこでライブマイグレーション実行時に各メモリページの更新の有無を予測し，更新される可能性が低いメモリページを pre-copy イテレーションの前半部分で転送し，更新の可能性が高いメモリページを pre-copy イテレーションの後半部分で転送することで，メモリページ転送後の更新発生を抑制する．

3.2 段階分けされた転送優先度の定義

メモリページの転送順序を変更する方法として，各ページにおける過去の更新履歴から更新可能性を推定し，可能性の低い順にページを転送する方法が考えられる．しかし，この方法では全メモリページの更新予測の結果をソートする必要があるため，オーバヘッドが大きくなってしまう．

そこで本研究では，過去の更新履歴を基に，段階分けされた転送優先度を新たに定義した．まずはじめに，各メモリページの更新履歴を参照し，（相対的に）頻繁に更新されているページを更新可能性の高いページと推定する．同様に，更新頻度が少ないページは更新可能性が低いページと判断する．後述するように，各ページの更新頻度はタイムスタンプで表されている．そして，段階分けされた転送優先度を定義するためにタイムスタンプの最大値と最小値を抽出し，両者の区間を等分割する．次に，各区間に対して転送順序における優先度を対応付ける．優先度は，更新される可能性が相対的に低い区間ほど高く設定される（先に転送される）．そして，各メモリページに対し，自身の更新頻度（タイムスタンプ）に対応する区間の優先度が与えられる．転送順序の変更はこの優先度に従って行うものとする．以上の手法により，転送順序変更のオーバヘッドを抑えることが可能となる．

3.3 EPT に着目したメモリアクセスプロファイリング

加えて筆者らは，メモリアクセスプロファイリングのオーバヘッドを抑えるために，EPT に着目したプロファイリング手法を提案する．EPT はゲスト物理アドレス空間と実際の物理アドレス空間の対応関係（EPTE: EPT Entry）を TLB（Translation Lookaside Buffer）に登録することで，ゲスト仮想アドレス空間からホスト物理アドレス空間への高速な変換を低コストで実現する機能である．

本研究で使用する QEMU/KVM は，EPT および TLB を用いたハードウェアによるアドレス変換と，ソフトウェアによるアドレス変換を行っている．後者の変換は，仮想マシンによるメモリアクセスの際に TLB ミスが発生した場合に実行される．仮想マシンは，TLB ミス発生時に通常のページウォークを行い，ゲスト仮想アドレスからゲスト物理アドレスに変換する．次に，仮想マシンから QEMU/KVM 側に処理が遷移し，処理の遷移が発生した理由と TLB ミスの原因となったゲスト物理アドレスが KVM に通知される．KVM はゲスト物理アドレスから実際のホスト物理アドレスへの変換を行い，その対応関係を示す EPTE を TLB に格納する．このとき，KVM に対してゲスト物理アドレスとともにメモリページの更新情報が通知される．したがって，TLB ミスが発生して KVM が EPTE を TLB に格納する際に，該当ページのメタデータとしてタイムスタンプを記録しておくことで，メモリアクセスによるメモリページの更新情報を低コストで記録することが可能になる．

4. 設計と実装

4.1 設計

本章では，QEMU/KVM を対象とした提案手法の実現

方法について説明する．提案手法を以下の 3 つの機能により実現した．

1. メモリアクセスプロファイリング

通常動作（ライブマイグレーションを行っていない場合の仮想マシンの動作）時において，仮想マシンのメモリ更新にともなう TLB ミスが発生した場合，更新履歴としてタイムスタンプを記録する．

2. 転送順序における優先度の設定

1. の結果を基に，仮想マシンモニタが仮想マシンの各メモリページに転送順序の優先度を与える．

3. メモリページ転送順序の変更

2. で設定した優先度を基にメモリページの転送順序を変更する．

また，本研究で行う実装は，QEMU/KVM の独自機能に依存しないため，他の仮想マシンモニタ上に移植する際の技術的障壁は少ないと考えられる．

4.2 EPT に着目したメモリアクセスプロファイリング

各メモリページの更新頻度を求めるために，プロファイリングアルゴリズムを検討する必要がある．複雑なアルゴリズムだと，更新頻度を求める際の演算オーバヘッドが大きくなるため，本実装ではタイムスタンプを用いた MRU（Most Recently Used）方式を採用した．

MRU 方式を用いるためには，各メモリページの更新時刻を示すタイムスタンプが必要になるが，タイムスタンプとして実時間を使用すると，システムクロック取得のオーバヘッドが大きくなる．そこで本実装では，タイムスタンプとして論理クロック値を使用することとし，論理クロック値を格納する配列 `time_stamp` を仮想マシンモニタ内に追加する．配列 `time_stamp` の各要素は各メモリページに対応している．本実装におけるメモリアクセスプロファイリングの実際の動作を以下に記述する．まず，仮想マシンモニタは EPTE を TLB に格納する際，仮想マシンから通知された更新情報を確認する．メモリアクセスによる更新を検知した場合，その時点の論理クロックの値をタイムスタンプとして `time_stamp` に格納し，論理クロックの値をインクリメントする．そしてライブマイグレーション開始時に，`time_stamp` に格納された値を基に，各メモリページの更新可能性を推定する．

4.3 転送順序における優先度の取得

メモリページの転送順序を示す優先度は，**図 1** に示したタイムスタンプの値と優先度の対応関係によって決められる．まず，タイムスタンプ値が 0 のメモリページは更新対象外のページと推定できるため，転送の優先度が最も高くなるように設定する（図 1 の例では 5）．次に更新頻度の最大値と最小値，すなわちダーティなメモリページにおけるタイムスタンプの最新の値と最古の値を取得する．この区間を等間隔で分割し，最小値を含む区間から順に高い優先度を与える．各メモリページには，該当する区間の優先度を転送順序における優先度として設定する．

図 1 タイムスタンプと優先度の対応関係
Fig. 1 The relationship between timestamps and priorities.

ページの更新予測が完全であるような理想的な状況下では，優先度を細かく設定するほど転送順序変更の効果が得られるが，実際には過去の更新履歴どおりにページの更新が発生するわけではなく，さらに順序変更のオーバヘッドも考えられる．逆に優先度の段数が小さ過ぎる場合は，きわめて頻繁に更新が発生するページがあまり更新が行われないページと同じ優先度を持つ可能性がある．そこで本実装では，筆者らの経験もふまえて，オーバヘッドが小さくかつ順序変更の効果が得られる段数として 6 を選択した．

TLB への EPTE の登録は KVM モジュールが行っているため，各メモリページの更新情報はカーネル空間に存在する．一方で，メモリページの転送順序を管理する仮想マシンモニタはユーザプロセスであるため，優先度を用いて転送順序を変更するためには，カーネル空間に存在する優先度情報をユーザ空間にコピーしなければならない．そこで本実装では，Linux が標準でサポートしている `ioctl` システムコール内に優先度情報を取得する機能を追加した．

仮想マシンモニタが転送順序の優先度を要求してから取得するまでの動作を**図 2** に示す．まず仮想マシンモニタは，ライブマイグレーションを開始する直前に，`ioctl` システムコールを用いて転送順序の優先度情報を KVM モジュールに対して要求する．次に，KVM モジュールは，各メモリページに対してメモリアクセスプロファイリングを行って得られたタイムスタンプの値から，転送順序の優先度を決定する．最後に，KVM モジュールは `copy_to_user` システムコールを用いて，仮想マシンモニタに優先度情報を通知する．

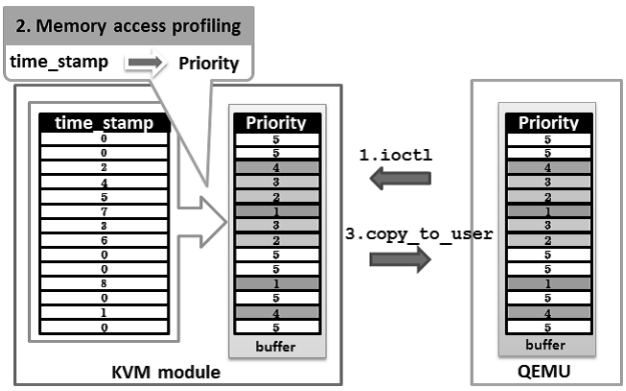


図 2 メモリページの転送順序における優先度の取得
Fig. 2 Acquisition of priority in memory page transfer.

4.4 メモリページ転送順序の変更

メモリページの転送順序の変更は仮想マシンモニタが行う．まず，メモリページの転送に必要な情報を格納する配列 `transferred_order` を用意し，`ioctl` システムコールで得た優先度情報を先頭のメモリページから順に確認し，メモリページの転送に必要な情報を `transferred_order` に転送優先度順に格納する．そして，ライブマイグレーション実行時に `transferred_order` の先頭要素から順に内容を確認し，その内容に従ってメモリページを転送する．

5. 評価

本章では，QEMU/KVM 上に実装した提案手法（proposal）によって，実際のライブマイグレーション時における仮想マシンの停止期間がどの程度削減できるかを，オリジナルの QEMU/KVM が使用する手法（original）と比較して評価を行った．また参考として，文献 [10] で提案された手法（TSB）も QEMU/KVM 上に実装して評価を行った．なお，文献ではメモリアクセスプロファイリングの詳細が述べられていなかったため，本実験では proposal と TSB は同じメモリアクセスプロファイリング方法を用いている．また，TSB では現在の論理クロックから 50,000 クロック前の値までをタイムスタンプとして保持するメモリページを更新される可能性の高いメモリページとして扱っており[*1]，一方で proposal では 6 段階の優先度を設けてある．

5.1 評価環境

本評価実験では，ライブマイグレーション実行時における転送元となる物理マシン（source）および転送先となる物理マシン（target）を 1 台ずつ使用した（**表 1**，**表 2** 参照）．また本評価環境では，共有ストレージ上に仮想マシンのイメージファイルを配置し，仮想マシンの起動および仮想マシン上でのファイルの利用やページスワップなどを行う際には，ネットワーク越しにこのイメージファイルを利用する．したがって，ライブマイグレーションを行う際には，仮想マシンのメモリ内容と仮想 CPU などの状態情報のみが転送される．

なお，仮想マシンには単一の CPU と 1 GB のメモリ空間を与え，ゲスト OS として CentOS 5.8 server を動作させた．

表 1 転送元マシンのスペック
Table 1 Machine spec. of the source machine.

OS	Ubuntu 11.04 Server（Linux 3.5.4）
VMM	QEMU/KVM 0.14.0
CPU	Intel(R) Core^{TM}i5 CPU 660 @ 3.33 GHz
Memory	8 GB
Network	100BASE-TX

表 2 転送先マシンのスペック
Table 2 Machine spec. of the target machine.

OS	Ubuntu 11.04 Server（Linux 3.5.4）
VMM	QEMU/KVM 0.14.0
CPU	Intel(R) Core^{TM}2 Quad CPU Q6600 @ 2.40 GHz
Memory	4 GB
Network	100BASE-TX

[*1] この閾値はヒューリスティックスな方法で決定しており，後述する評価実験において最も優れた性能を示した値を採用している．

5.2 オーバヘッド評価

まずはじめに，上記の評価環境を用いて，本提案手法におけるライブマイグレーション実行時の演算オーバヘッドを評価した．本評価では，ユーザアプリケーションが動作していない状態の仮想マシンをライブマイグレーションした際の，メモリページ転送順序における優先度の取得と転送順序変更の実行時間を測定した．その結果，優先度の取得の平均計測時間は 0.278 秒，転送順序変更の平均計測時間は 0.004 秒であった．これらの合計は約 0.3 秒であり，pre-copy 期間の平均実行時間（約 17 秒）の約 0.6%と非常に小さいものだった．したがって，本提案手法による演算オーバヘッドはライブマイグレーションの実行時間に大きな影響を与えることはないといえる．

5.3 ベンチマークによる性能評価

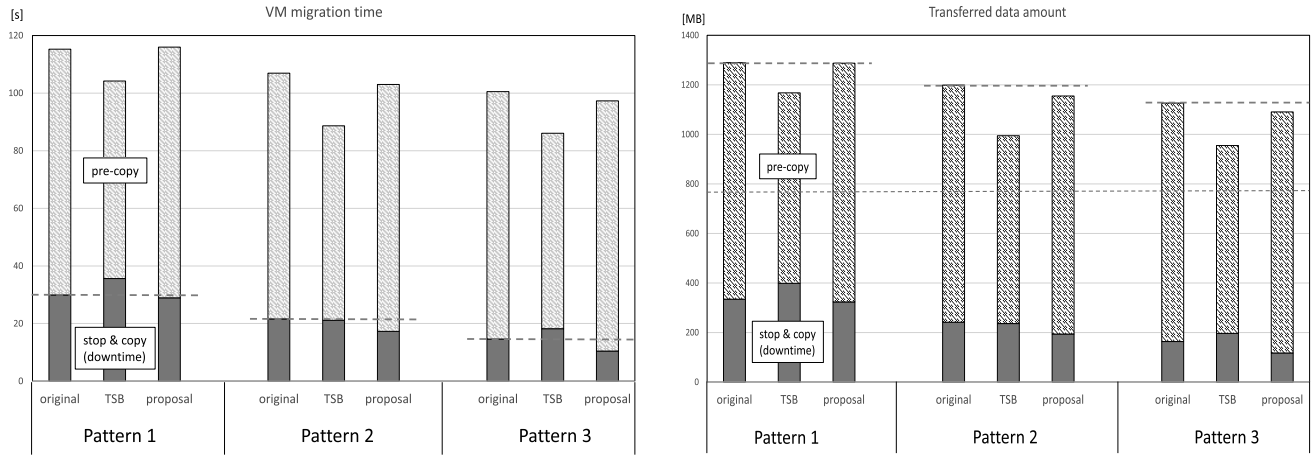
本実験では，メモリを更新し続けるプログラムを仮想マシン上で動作させた状態でライブマイグレーションを行い，pre-copy 期間および stop-and-copy 期間の長さを評価した．加えて，提案手法による転送順序の変更によってページの再転送が抑制されることを確認するため，ライブマイグレーション中におけるデータ転送量の評価も行った．

仮想マシン上で動作させるプログラムとして，本実験のために筆者らが作成したベンチマークアプリである memory_dirty と，計算機クラスタのベンチマークとして幅広く使用されている NAS Parallel Benchmarks（NPB）[16] の 2 種類を使用した．

5.3.1 memory_dirty を用いた評価実験

本実験のために筆者らが作成したベンチマークアプリである memory_dirty の動作概要について説明する．

memory_dirty は，起動直後に 768 MB のメモリ領域を確保した後で，事前処理として全メモリページに対して書き込みを行い，全ページがライブマイグレーション時の転送対象となるようにする．次に，一般的なアプリケーションによるメモリアクセスの局所性を想定して，確保したメモリ領域をさらに S1，S2，S3，S4（128 MB），S5（256 MB）の 5 つの領域に分割する．このうち，S5 を除く S1，S2，S3，S4 は事前処理以降も更新され続ける領域であり，また **表 3** で示すように，S1，S2，S3，S4 におけるメモリページの更新頻度に偏りを持たせた．そして，S1 から S4 の各



(a) 実行時間 [s]　　(b) データ転送量 [MB]

図 3 memory_dirty を用いた評価結果
Fig. 3 Evaluation results using memory_dirty.

表 3 更新頻度の偏り
Table 3 Memory update patterns.

	S1	S2	S3	S4
pattern1	0.50	0.30	0.15	0.05
pattern2	0.70	0.25	0.04	0.01
pattern3	0.90	0.05	0.03	0.02

領域に対して，S1 から順にランダムにページを選択して書き込みを行うという処理を繰り返し，書き込みが既定回数に達したら次の領域の処理に遷移する．S4 の処理が終わった時点で，各領域におけるページの更新頻度（回数）は表 3 で示すような偏りを持っている．その後，また S1 から順に上記の処理を繰り返す．

表 3 の各パターンについて，pattern1 は各領域における更新頻度の差が小さく，各メモリページが更新される可能性の高さは僅差である．pattern2 は pattern1 に比べて更新頻度の偏りが大きく，特に S1 と S2 に更新が集中する．pattern3 は S1 の更新頻度が他の領域に比べ極端に大きく，更新は S1 のメモリページに集中し，他の領域は更新される可能性が低い．したがって，pattern1，pattern2，pattern3 の順に更新頻度の偏りが小さいといえる．

既存の QEMU では，先頭のメモリページから順に転送を行うため，提案手法における転送順序変更の効果は，先頭に近いメモリ領域の更新頻度に依存すると考えられる．

5.3.2 NAS Parallel Benchmarks を用いた評価実験

NPB は 5 つの Parallel Kernel Benchmarks と 3 つの Parallel CFD（Computational Fluid Dynamics）Application Benchmarks から構成されており，並列コンピュータにおける実行性能の評価に広く利用されている．本実験では Parallel CFD Application Benchmarks を利用した．

Parallel CFD Application Benchmarks は，流体力学のシミュレーションを行うアプリケーションの特徴を擬似的に再現した以下の 3 種類のベンチマークプログラムを提供している．

(1) BT（Block Tri-diagonal Solver）
非優位対角な 5 × 5 のブロックサイズを持つブロック 3 重対角方程式（3 重対角行列を含む方程式）を解く．

(2) SP（Scalar Perita-diagonal Solver）
非優位対角なスカラ 5 重対角方程式（5 重対角行列を含む方程式）を解く．

(3) LU（Lower-Upper Gauss-Seidel Solver）
5×5 のブロックを持つ上下三角行列システムを SSOR（Symmetric Successive Over-Relaxation）法で解く．

各ベンチマークプログラムが生成するグリッドのサイズとして，NPB で定義されている C クラスを利用しており，生成されるグリッドのサイズはすべてのベンチマークプログラムにおいて 162 × 162 × 162 である．

加えて，複雑なアクセスパターン下でのライブマイグレーション性能を評価するため，LU，SP，BT を並列動作させた場合の性能も測定した．このとき，各ベンチマークプログラムが生成するグリッドサイズは 100 × 100 × 100 とした．なお，本実験では，ライブマイグレーションの対象となる仮想マシンに割り当てる仮想 CPU 数は 4 とした．

5.3.3 評価結果（memory_dirty）

仮想マシン上で memory_dirty を実行し，ライブマイグレーションを行った場合の pre-copy 期間と stop-and-copy 期間の実行時間およびデータ転送量を**図 3** に示す．なお，stop-and-copy 期間の実行時間は，ライブマイグレーション実行期間における仮想マシンの停止時間に相当する．

図 3 (a) の結果を見ると，更新頻度の偏りが顕著である場合（pattern2，pattern3）において，提案手法における stop-and-copy 期間の短縮率が大きくなることが確認できる．一方で pattern1 では，proposal と original の停止期間にほとんど差が見られない結果となった．これは，各領域

の更新頻度の差が小さく，仮想マシン上のメモリページの大部分が更新対象となるため，提案手法による転送順序スケジューリングの効果が小さかったことが原因である．

続いてデータ転送量 (b) の結果を見ると，pattern1 における総ページ転送量は original と同程度であるが，pattern2 および pattern3 では転送量を削減できていることが分かる．

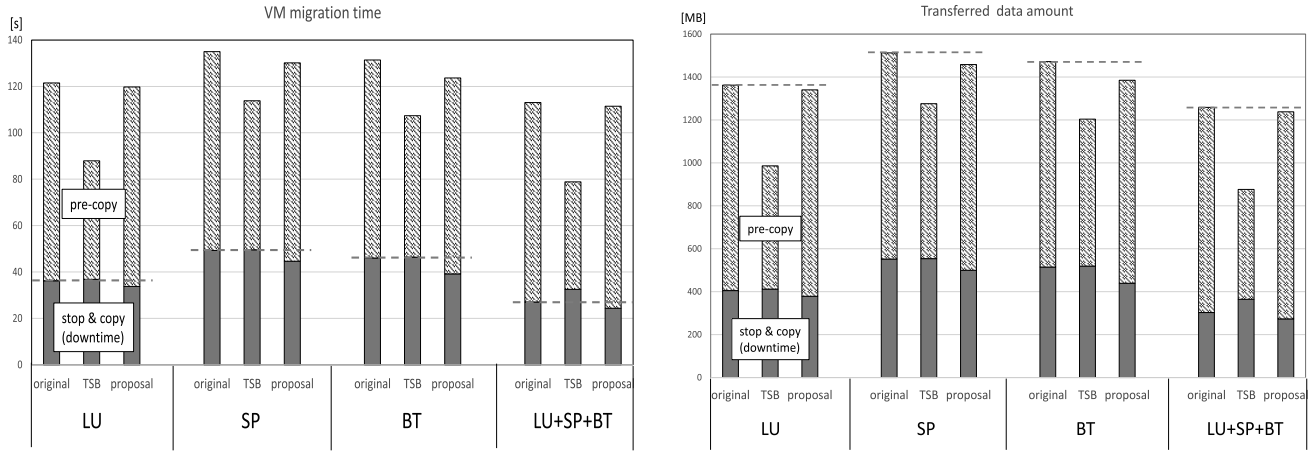
次に，TSB と比較した場合，pre-copy 期間は長期化したものの，stop-and-copy 期間における転送時間は大きく削減された．これは，TSB が更新される可能性の高いメモリページの転送を pre-copy 期間の終了時まで延期することで，pre-copy 期間における再送回数が制限され，stop-and-copy 期間のメモリページの転送量が増大してしまうのに対し，proposal では pre-copy 期間において更新される可能性の高いメモリページの転送を制限せず，更新される可能性の低いページを優先して転送しているからである．提案手法を用いたライブマイグレーションでは，TSB と比較して stop-and-copy 期間（仮想マシンの停止期間）が平均で 17.2%，最大で 28.3%削減されており，ライブマイグレーション実行時における仮想マシン利用者の体感品質を改善できることが分かった．また，評価結果から，各領域における更新頻度の偏りが大きいほど，proposal が有効に働くことも分かった．これは，複雑なメモリアクセスパターンになるほど，各領域における更新頻度の差が大きくなるため，優先度に基づく転送順序変更が有効に働くことが原因だと考えられる．

5.3.4 評価結果（NAS Parallel Benchmarks）

次に，仮想マシン上で NAS Parallel Benchmarks の提供する 3 つのベンチマークプログラムを利用した際の評価結果を**図 4** に示す．ここで，SP，LU，BT はベンチマークプログラムである SP，LU，BT を単一で実行した場合の実験結果を示し，SP+LU+BT は SP，LU，BT を並列で実行した場合の評価結果を表している．

実験結果から，各プログラムにおいて proposal は stop-and-copy 期間における実行時間の削減，および original に対する総ページ転送量の削減を確認できる．これは，LU，SP，BT の共通点である多重対角行列がプログラム中に表れることが理由としてあげられる．多重対角行列は，行列における主対角線とその上下 $(n-1)/2$ の対角線にのみ非零の成分を持つ行列であり，多重対角行列を用いるプログラムでは，各メモリページの間で更新頻度に大きな差が生まれる．proposal では優先度に基づいたメモリページ転送順序の変更を行うため，非零成分を含むページの転送が後工程に回ったと考えられる．実際，LU を用いた実験では 6.6%，SP を用いた実験では 9.4%，BT を用いた実験では 14.9%停止時間を削減できた．したがって，実際に利用されるプログラムが動作している仮想マシンのライブマイグレーションにおいて，提案手法は有効に働くことが確認できる．また，LU，SP，BT を並列に実行した実験では 9.8%の削減を実現できた．このことから，異なるアクセスパターンを持つプログラムが並列動作するという複雑な条件下において，優先度に基づく転送順序の変更が有効に働くことが分かった．一方，LU，SP，BT を並列に実行した実験において，TSB の stop-and-copy 期間における実行時間およびデータ転送量が original と比較して増加していた．これは，pre-copy 終了時まで更新される可能性の高いメモリページを転送しなかったことが原因で，stop-and-copy 期間における転送量が増加したためである．NPB が提供するベンチマークプログラムを用いた実験より，proposal は，各メモリページの更新頻度に偏りがあるプログラムの実行に対して，各プログラムのアクセスパターンの差異を吸収し，その効果を発揮することを確認した．

以上のことから，更新可能性を考慮したメモリページの転送順序スケジューリングはメモリページの更新頻度に大きな差がある場合，有効に働くことが確認できた．また，



(a) 実行時間 [s]　　(b) データ転送量 [MB]

図 4 NPB を用いた評価結果
Fig. 4 Evaluation results with NPB.

複雑なアクセスパターンに基づき更新が行われる場合においても，優先度に基づいた転送順序の変更により，転送順序スケジューリングの効果を維持していることも確認できた．

したがって，本実験により，EPT に着目したメモリアクセスプロファイリングと転送順序の変更は，有効であることが分かった．

6. まとめ

ライブマイグレーションにおけるメモリページの再送回数を削減するため，仮想マシンによるメモリアクセスプロファイリングを使用したメモリページ転送順序スケジューリング方式と，拡張ページテーブル（EPT）を利用した低オーバヘッドなプロファイリング手法を提案した．

本研究では，仮想マシンモニタの 1 つである QEMU/KVM 上に提案手法を実装し，実際のライブマイグレーションにおける性能評価を行った．特定のパターンでメモリ更新を行うプログラムと NAS Parallel Benchmarks が提供するプログラムを仮想マシン上で動作させ，比較対象にはオリジナルの QEMU/KVM が利用する方式および既存研究である TSB を用いた．実験の結果，仮想マシン上のメモリが複雑なアクセスパターンに基づいて更新される状況において，stop-and-copy 期間におけるデータ転送時間および総ページ転送量の削減を確認できた．

今後の課題として，理想的な転送優先度の段数と実環境（ページサイズ，ページ数，アプリケーションの種類など）との関係性を明らかにする必要がある．そして，実際に運用されているサービスを想定した評価実験を行い，実運用段階での本提案手法の有効性を確認したい．また，本提案手法の効果を保証しつつ pre-copy 期間におけるページ転送量を削減することで，ライブマイグレーション実行時における計算機資源（ネットワークや CPU）の消費量の増大を防ぐ必要もある．

謝辞 本研究の一部は，科研費基盤研究（C）24500113 による．

参考文献

[1] Amazon Web Services, Inc.: AWS – Amazon Elastic Compute Cloud (EC2) (online), available from ⟨https://aws.amazon.com/ec2/⟩ (accessed 2014-05-02).

[2] Yang, K., Gu, J., Zhao, T. and Sun, G.: An Optimized Control Strategy for Load Balancing Based on Live Migration of Virtual Machine, *Proc. 2011 6th Annual ChinaGrid Conference* (*ChinaGrid*), pp.141–146 (2011).

[3] Hirofuchi, T., Nakada, H., Itoh, S. and Sekiguchi, S.: Reactive Consolidation of Virtual Machines Enabled by Postcopy Live Migration, *Proc. 5th Intl. Workshop on Virtualization Technologies in Distributed Computing* (*VTDC'11*), pp.11–18 (2011).

[4] Zhang, W., Zhu, M., Mei, Y., Xiao, L., Ruan, L., Gao, Y. and Sun, Y.: LVMCI: Efficient and Effective VM Live Migration Selection Scheme in Virtualized Data Centers, *Proc. IEEE 18th Intl. Conf. on Parallel and Distributed Systems* (*ICPADS 2012*), pp.368–375 (2012).

[5] Xu, F., Liu, F., Liu, L., Jin, H., Li, B. and Li, B.: iAware: Making Live Migration of Virtual Machines Interference-Aware in the Cloud, *IEEE Trans. Computers* (2013).

[6] Clark, C., Fraser, K., Hand, S., Hansen, J.G., Jul, E., Limpach, C., Pratt, I. and Warfield, A.: Live Migration of Virtual Machines, *Proc. 2nd ACM/USENIX Conf. on Symp. on Networked Systems Design & Implementation* (*NSDI'05*), pp.273–286 (2005).

[7] Hirofuchi, T., Nakada, H., Itoh, S. and Sekiguchi, S.: Enabling Instantaneous Relocation of Virtual Machines with a Lightweight VMM Extension, *Proc. 10th IEEE/ACM Intl. Conf. on Cluster, Cloud and Grid Computing* (*CCGrid*), pp.73–83 (2010).

[8] Kivity, A., Kamay, Y., Laor, D., Lublin, U. and Liguori, A.: KVM: The Linux Virtual Machine Monitor, *Proc. 2007 Linux Symposium*, pp.225–230 (2007).

[9] VMware, Inc.: VMware vSphere, (online), available from ⟨http://www.vmware.com/products/vsphere⟩ (accessed 2014-05-02).

[10] Hu, B., Lei, Z., Lei, Y., Xu, D. and Li, J.: A Time-Series Based Precopy Approach for Live Migration of Virtual Machines, *Proc. IEEE 17th Intl. Conf. on Parallel and Distributed Systems* (*ICPADS*), pp.947–952 (2011).

[11] Svard, P., Tordsson, J., Hudzia, B. and Elmroth, E.: High Performance Live Migration Through Dynamic Page Transfer Reordering and Compression, *Proc. 2011 IEEE 3rd Intl. Conf. on Cloud Computing Technology and Science* (*CloudCom*), pp.542–548 (2011).

[12] Akiyama, S., Hirofuchi, T., Takano, R. and Honiden, S.: MiyakoDori: A Memory Reusing Mechanism for Dynamic VM Consolidation, *Proc. 2012 IEEE 5th Intl. Conf. on Cloud Computing* (*CLOUD*), pp.606–613 (2012).

[13] Zhang, X., Huo, Z., Ma, J. and Meng, D.: Exploiting Data Deduplication to Accelerate Live Virtual Machine Migration, *Proc. IEEE Intl. Conf. on Cluster Computing* (*CLUSTER*), pp.88–96 (2010).

[14] Neiger, G., Santoni, A., Leung, F., Rodgers, D. and Uhlig, R.: Intel Virtualization Technology: Hardware Support for Efficient Processor Virtualization, *Intel Technology Journal*, Vol.10, No.3, pp.167–178 (2006).

[15] Advanced Micro Devices, Inc.: AMD-V Nested Paging, White paper (2008).

[16] Bailey, D.H., Barszcz, E., Barton, J.T., Browning, D.S., Carter, R.L., Fatoohi, R.A., Frederickson, P.O., Lasinski, T.A., Simon, H.D., Venkatakrishnan, V. and Weeratunga, S.K.: The NAS parallel benchmarks, *The International Journal of Supercomputer Applications*, Vol.5, No.3, pp.66–73 (1991).

中居 新太郎

2012 年名古屋工業大学工学部情報工学科卒業．2014 年同大学大学院創成シミュレーション工学専攻博士前期課程修了．同年株式会社インテック入社，現在に至る．

川島 龍太 （正会員）

2005 年岩手県立大学ソフトウェア情報学部ソフトウェア情報学科退学．2007 年同大学大学院博士前期課程修了．2010 年総合研究大学院大学複合科学研究科情報学専攻博士課程修了．同年株式会社 ACCESS 入社．2013 年名古屋工業大学大学院助教，現在に至る．SDN，システムソフトウェア等の研究に従事．博士（情報学）．電子情報通信学会，IEEE 各会員．

齋藤 彰一 （正会員）

1993 年立命館大学理工学部情報工学科卒業．1995 年同大学大学院博士前期課程修了．1998 年同大学院博士後期課程中退．同年和歌山大学システム工学部情報通信システム学科助手．2003 年同講師，2005 年同助教授．2006 年名古屋工業大学大学院助教授，2007 年同准教授，現在に至る．オペレーティングシステム，インターネット，セキュリティ等の研究に従事．博士（工学），ACM，IEEE-CS 各会員．

松尾 啓志 （正会員）

1983 年名古屋工業大学工学部情報工学科卒業．1985 年同大学大学院修士課程修了．同年松下電器産業（株）入社．1989 年同大学院博士課程修了．同年名古屋工業大学電気情報工学科助手．講師，助教授を経て，2003 年同大学大学院教授，現在に至る．分散システム，分散協調処理に関する研究に従事．工学博士．電子情報通信学会，人工知能学会，IEEE 各会員．