情報処理学会論文誌　コンピューティングシステム　Vol. 3　No. 3　248–262 (Sep. 2010)

既存VMMへの適用が容易でゲスト透過なポストコピー型仮想マシン再配置機構

広渕崇宏[†1]　中田秀基[†1]
伊藤智[†1]　関口智嗣[†1]

ポストコピー型の仮想マシン再配置機構は仮想マシンの実行ホストを素早く切り替えられるため，データセンタの運用効率を向上させるうえで有用な技術であると考えられる．しかしながら，今日一般的に利用できるまでには至っていない．先行研究におけるポストコピー型再配置機構は，既存の仮想マシンモニタ（VMM）への変更が大きく，ゲストOSの改変も必要になる点に問題がある．そこで我々は，既存VMMのへ拡張が単純でゲストOSの改変も不要な，新たなポストコピー型ライブマイグレーション機構を提案する．メモリアクセスのトラップ処理とメモリページのコピー処理をVMMの外部で実装することで，VMMへの変更量を抑えながらポストコピー型再配置を実現する．再配置性能を検証するため，SPECweb2005を用いて評価実験を行った．負荷の高いウェブサーバを実行するVMであっても，1秒以内に実行ホストを切り替えることができた．実行ホスト切替え後の性能低下は限定的であった．プレコピー型再配置に比べて，VMのすべての状態を移動する時間も短縮できた．

A Guest-transparent, Post-copy-based VM Migration Mechanism with a Lightweight Extension to KVM

Takahiro Hirofuchi,[†1] Hidemoto Nakada,[†1]
Satoshi Itoh[†1] and Satoshi Sekiguchi[†1]

Post-copy-based VM migration is considered a promising technology for next-generation datacenters; memory pages are transferred after a VM restarts at a destination host, thereby minimizing the time of switching the execution host. Post-copy-based migration mechanisms, however, have not yet been available in industry. Prototype implementations in prior work need major modifications to existing virtual machine monitors (VMMs), and also require special software support in guest operating systems. In this paper, we propose a simple and plain implementation of post-copy-based migration, which is implemented as a lightweight extension to KVM. It supports any guest operating systems without their modifications. The RAM of a migrated VM is mapped to a special character device, which transparently transfers memory pages on demand. Experiments were conducted by using the SPECweb2005 benchmark. A running VM with heavily-loaded web servers was successfully relocated to a destination within one second. Temporal performance degradation after relocation was alleviated by pre-caching memory pages. In addition, for memory intensive workloads, our migration mechanism moved all the states of a VM faster than a pre-copy-based migration mechanism.

1. はじめに

計算機センタやデータセンタの資源運用の効率性を高めるために，仮想計算機（VM）とそのライブマイグレーション技術が注目されている．仮想計算機技術によって計算機資源を抽象化して論理的に分割・共有できる．さらに仮想マシンモニタ（VMM）が備えるライブマイグレーション機能によって，VMをいっさい停止することなく異なる物理ノード上に再配置可能になる．

我々が進めている省エネデータセンタプロジェクトでは，ライブマイグレーションによって計算機クラスタ上のVM配置を積極的に変更することで，データセンタの稼動エネルギーコストを削減することを目指している．今日，Amazon EC2[1)]等の仮想ホスティング・サービスプロバイダにおいては，VMに対して割り当てた性能値（性能の目安となる値）を基に，1つの物理ノードに対して配置するVM数を固定的に決めている．たとえ，VMのCPU使用率が小さくて，物理ノードの処理能力に余裕が生じていても，余剰計算資源を有効利用できない．しかし，我々のプロジェクトでは，VMのCPU使用率をモニタしながら，動的に配置を最適化する．VMのCPU使用率が小さければ，1つの物理ノードに対して，性能保証値から導かれる数よりも多数のVMを配置（*overcommit*）する．また，VMのCPU使用率が大きくなれば，性能保証を満たすべくVMを多数の物理ノードへ分散させる．VMの処理性能を保証しつつもハードウェア資源の稼動効率を向上させることで，一歩進んだ省エネルギー・効率化が可能になると考えている．

我々はすでに遺伝的アルゴリズムを用いた動的なVM配置決定システム[2)]を試作してい

†1 産業技術総合研究所
National Institute of Advanced Industrial Science and Technology (AIST)

る．しかし，既存の VMM[3),4)] で一般に利用できるプレコピー型とよばれるライブマイグレーション[5),6)] では，VM をホストしている物理ノードの変更に数十秒単位の時間がかかり，VM の CPU 負荷の増加に追随して，配置状態を速やかに変更することが困難であった．性能保証値を満たせない期間が長時間にわたってしまい，実際的な商用ホスティングサービスへの応用が難しかった．

先行研究[7),8)] においては，実行ホストの切替え時間を短縮するため，ポストコピー型と呼ばれるライブマイグレーション手法が提案されている．ライブマイグレーション開始時には，CPU レジスタやデバイスの状態のみを移動先ホストにコピーし，すぐさま VM の実行ホストを切り替える．その後，移動元ホストからオンデマンドにメモリページを取得している．しかし，いずれの手法も既存 VMM やゲスト OS カーネルへの変更量が大きく，将来的にオープンソースコミュニティで開発される VMM に統合することは容易ではない可能性がある．また，VM 内部において特殊なドライバやプログラムを動かす必要があり，データセンタのように顧客の VM をホストする環境では望ましくない．

そこで，本研究では，既存 VMM への拡張が単純で，ゲスト OS への変更も不要な，新たなポストコピー型ライブマイグレーション機構を提案する．ポストコピー型再配置を実装するうえで必要になる，VM のメモリアクセスのトラップ処理と移動元ホストからのページコピー処理の両方を KVM の外部で実装することで，KVM への変更量を抑えながら機構全体を単純に実装できる．また，KVM 本来の完全仮想化機能を損なうことなく，既存のゲスト OS をそのままポストコピー型再配置においても利用できる．近い将来，オープンソースコミュニティで開発されている VMM へ開発成果を統合することを期待している．

2 章で，現在利用可能なプレコピー型マイグレーション技術では不十分であることを述べる．3 章で先行研究におけるポストコピー型マイグレーション機構を概観する．4 章で我々の提案機構について説明し，5 章で我々の実装を確認する．6 章で評価実験を示し，7 章で議論を追加し，8 章でまとめる．

2. VM 配置の最適化とプレコピー型ライブマイグレーションの問題点

我々の研究プロジェクトでは，顧客の VM をホスティングする仮想化データセンタを対象として，計算機クラスタにおける効率的な VM 配置手法を開発している．VM の性能値だけで VM の配置を決めるのではなく，実際の CPU 使用率によって VM 配置を動的に決定・変更する．VM の CPU 使用率が小さいときには，顧客に示した性能値にかかわらず，1 つの物理ノード上に多数の VM を配置して効率的な運用が可能になる．一方，VM の CPU 使用率が増大し，物理ノードの資源を超えたときには，顧客に示した性能値を満たすべく，すぐさま新たな配置状態に変更し過負荷状態を解消する．そこで，VM を実行しているホストを迅速に変更できる仕組みが必要となる．また，その仕組みは顧客に対して透過的である必要がある．

しかし，現在一般に利用できるライブマイグレーションの実装[5),6),9),10)] は，いずれも VM のすべての状態を移動先に転送してから実行ホストを切り替える，プレコピー型と呼ばれる機構である．それゆえに，迅速に実行ホストを変更することができない．ライブマイグレーションの開始から実際に実行ホストを切り替えるまでの時間は，メモリサイズやネットワーク帯域，またメモリの更新頻度に依存するものの，たとえば，Amazon EC2 の標準的な VM サイズ（メモリ 1.7 GB）であれば，Gigabit Ethernet の環境で最低 13.6 秒（1.7 GB/1 Gbps）を要する計算になる．メモリイメージのコピー中は VM は移動元で動作中であり，変更されたメモリページを再帰的に転送するため，実際にはさらに時間がかかってしまう．メモリページの更新頻度が高い VM の場合，ページの転送速度がページの更新速度に追いつかず，有限時間内でマイグレーションが完了できない場合もある（6.3 節参照）．VMM の実装によっては，強制的にページ更新頻度を抑制する[5)] ため，性能低下が顕著になる場合もある．また，更新ページの再送によって，移動が完了するまで長時間にわたって大量のデータ転送が発生してしまう．実行ホストの切替えに時間がかかり，その所要時間の見積りが難しい点は，VM 配置の動的な最適化機構にとって望ましくないと考える．

3. 関 連 研 究

これまで，1 つの物理ノードに対して，できるだけ多くの VM を配置するため，メモリ使用量の抑制技術が開発されてきた．VM に割り当てるメモリ量を実際の使用量に応じて増減する機構や同一物理ノード上に存在する複数の VM 間で内容が同じメモリページを共有する機構がある[11)–14)]．我々は，これらの技術がすでに実用化されつつあることを考慮すれば，今後 VM の集約率を高めるうえでメモリ資源量よりも CPU 資源量や I/O 帯域幅がボトルネックとなる可能性があると認識している．そこで，それらの資源の overcommit を可能とするために，VM の実行ホストを迅速に切り替える仕組みが必要であると考えている．

先行研究においては，実行ホストの切替え時間を短縮するため，ポストコピー型と呼ばれるライブマイグレーション機構が提案されている．ライブマイグレーション開始時には，CPU レジスタやデバイスの状態のみを移動先ホストにコピーし，すぐさま VM の実行ホストを切り替える．そして，移動元ホストからオンデマンドにメモリページを取得している．

いずれの先行研究の手法も実行ホストの切替え時間の短縮に成功している．また，更新ページを再帰的に送る必要がないため，データ転送量も削減できている．

Snowflock[7] は，動作中の VM を他の物理ホスト上に複製する機構を設けている．プロセスの `fork()` 処理に類似した VM 複製を制御する API（VM Fork）を利用することで，複数台のノードを用いた分散処理プログラムを容易に記述できる．VM を動的に複製する機構は，Snowflock が VM Fork 後に複製元 VM の実行を継続する点を除けば，ポストコピー型のライブマイグレーションの仕組みに近い．複製の際には，複製元の VM の動作を停止して，CPU レジスタやページテーブル等を複製先ホストに転送する．そして，すぐさま複製元 VM の実行を再開する．このとき複製元では複製時点のメモリイメージを上書きしないように，メモリの変更は新たに割り当てたページ上で行う．複製先ホストで VM の実行を再開し，メモリアクセスがあるたび複製元のホストからメモリページを取得する．Xen の準仮想化モードに対して実装されており，ゲスト OS のメモリ割当ての振舞いを改変することで，複製元ホストからのメモリ取得を減らしている．しかし，その反面ゲスト OS に対する改変が必要になっている．

文献 8) では，ゲスト OS に専用ドライバを組み込んで特殊なスワップデバイスを作ることで，Xen の準仮想化モードに対してポストコピー型のライブマイグレーションを実装している．このスワップデバイスは，ディスクではなくゲスト OS メモリの予約領域をスワップアウト先とする．Xen の準仮想化モードにおけるメモリ抽象化を使って[⋆1]，スワップアウト・イン時にメモリコピーが発生しないようにしている．マイグレーションする際には，ゲスト OS はスワップアウト可能なメモリページをすべてスワップデバイスに書き出す．次に VM の実行を停止し，スワップアウトできなかったメモリ領域や CPU 状態を移動先ホストへコピーする．移動先ホストで VM が実行を再開すると，大半のメモリページがスワップアウトされた状態になっている．VM が新たにメモリページにアクセスするたびにスワップインが発生し，専用ドライバが移動元ホストから徐々にメモリページを取得していく．ゲスト OS のスワップ機能を使用することでハイパーバイザに対する変更量を若干抑えている．しかし，ゲスト OS に対して特殊なドライバやプログラムを組み込む必要がある．

いずれの手法もあらかじめゲスト OS 内部を改変しなければならない．Linux 以外へのゲスト OS に対応するためには新たな開発作業が必要になり，OS がバージョンアップするたびに対応作業を要する可能性がある．Amazon EC2 等 VM 内部の自由なカスタマイズを許す IaaS サービスプロバイダにおいては，特定のゲスト OS やその内部の機構に依存したマイグレーションは好ましくない．

また，VM の複製機構である Snowflock については実装が公開されているものの，ポストコピー型マイグレーションとして利用するためにはさらなる改変を要する．現状，ポストコピー型マイグレーションは期待される有用性に反して一般的に利用できる状況ではなく，早期に実環境で利用可能になる新たな実装が求められている．

4. 提 案 機 構

そこで，我々は，既存 VMM への拡張が単純で，ゲスト OS の改変も不要な，ポストコピー型ライブマイグレーション機構を新たに提案する．VMM として KVM（Kernel-based Virtual Machine）を利用し，VM のメモリ領域を特殊なデバイスファイルにマップすることで，ポストコピー型マイグレーションを KVM のコードをほとんど改変することなく実現する．以下，本章では，最初に要件を整理して実装方法を検討した後，提案機構の概要を述べる．

4.1 要　　件

仮想ホスティングデータセンタを対象とした動的な VM 配置最適化に対して，ポストコピー型ライブマイグレーションは有用であると考える．しかし，既存機構では不十分であり，新たに以下の要件を満たす仕組みが必要となる．

- 既存 VMM に対する変更が軽微であり，提案機構全体ができる限り単純であること．先行研究のポストコピー型機構は VMM やゲスト OS への変更量が多い．しかも仮想化の要となるメモリ管理部分のコードを大きく改変している．我々はできるだけ早くポストコピー型マイグレーションをオープンソースコミュニティで開発される VMM に統合し，近い将来に実証的な環境で広く運用したいと考えている．そのため，VMM に対する変更ができる限り小さく，ポストコピー型マイグレーションが単純な機構で実現できることが求められる．
- VM 内部の仕組みに依存しない仕組みであること．先行研究のポストコピー型機構はゲスト VM 内部の対応が必要になる．VMM を運用するデータセンタ側と，VM 内部にサービスを構築する顧客の側の，両方の管理ドメインにまたがった仕組みとなってしまい，現実的ではない．

⋆1 ゲスト OS から見た物理アドレス（Pseudo Physical Address）と実際の物理アドレス（Machine Frame Number）とのマッピングを変更する．

4.2　ポストコピー型マイグレーションの実装方法の検討

我々はオープンソースコミュニティで最も活発に開発が進められている VMM の 1 つである KVM を対象に，ポストコピー型マイグレーションの実装方法を検討した．KVM は，そのドライバを組み込んだ Linux カーネルをハイパーバイザとする．Intel VT [15] あるいは AMD-V [16] という CPU の仮想化機能を使用することで，物理ハードウェア向けの OS を改変することなく小さなオーバヘッドで VM 上で起動できる．ハードウェアエミュレータである QEMU [17] に対する拡張として実装されており，VM はホスト OS からは通常のプロセスとして見える．プレコピー型のマイグレーションに対応している．

ポストコピー型マイグレーションは，VM の実行ホストが移動先ホストに切り替わった後に，メモリページを転送する．移動先ホストにおいて，メモリアクセスが発生したページの内容を移動元からオンデマンドに取得する処理が必要である．つまり，

- 移動先ホストで，あるページに対するはじめてのメモリアクセスをトラップする処理，
- トラップしたメモリアクセスの対象ページについて，移動元ホストから内容をコピーする処理，

という 2 つの機能を実装しなければならない．

メモリアクセスのトラップ処理について着目すれば，実装方法は大きく分けると，KVM の中でメモリアクセスをトラップする手法と，次節以降で述べる KVM の外でトラップする手法が考えられる．

当初，我々は前者の手法を検討した．メモリアクセスのトラップ処理のみ KVM で実装し，メモリページの取得処理は KVM 外部において実装する．しかし，KVM 拡張を施した QEMU において，ゲスト OS からのメモリアクセス処理は，カーネル内に存在する KVM ドライバで動作する．一方，エミュレーションするデバイスからのメモリアクセス処理（DMA 等）は，大半のデバイスにおいてはユーザ空間において動作する．このため，カーネル空間およびユーザ空間それぞれでメモリアクセスをトラップする必要が生じ，拡張が複雑化してしまう．さらに，メモリアクセスをトラップした際には，コピー済みのページが記録してあるビットマップを参照する必要がある．カーネル空間およびユーザ空間双方から，適切な排他制御を行いながら 1 つのビットマップを参照・更新することは，複雑化の要因となり加えて速度低下も懸念された[⋆1]．

⋆1 1 つの VM に仮想 CPU を複数割り当てた場合に，カーネル空間とユーザ空間でのメモリアクセスが同時並行的に発生する．ビットマップの参照・更新のたびに何らかのシステムコールを介した排他制御が必要になる．

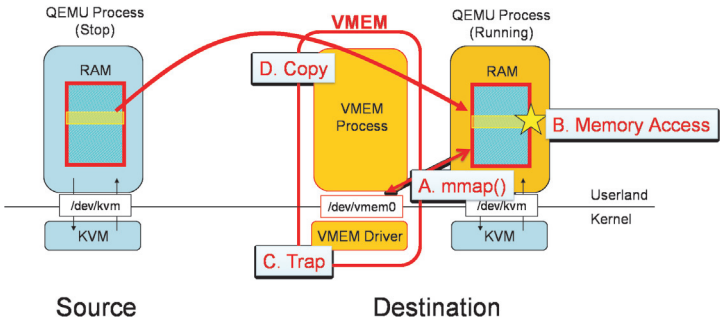

図 1　提案機構の概要
Fig. 1　Overview of proposed mechanism.

実際，我々の当初の実装では，カーネル空間でのシャドウページテーブルの作成処理中と，ユーザ空間でのデバイスエミュレーション処理中に，コピー対象のメモリページアクセスをトラップしていた．しかし，KVM に対する改変が複雑かつ大きなものになってしまったため，次に述べる提案機構を実装するに至った．

4.3　提 案 機 構

提案機構では，メモリアクセスをトラップする機構とメモリページを取得する機構両方を KVM の外部で実装することで，KVM への変更点を単純なものにとどめている．提案機構の動作を簡略化した**図 1** が示すように，移動先ホストでは，KVM の外部に実装された，VMEM と呼ぶ機構でメモリアクセスをトラップしページ内容を取得している．KVM は，VM のメモリ領域の割当て方法を変更して，VMEM が提供するデバイスファイルをマッピングすることで，VM のメモリ（RAM）を確保する（図 1 A）．VM があるメモリページにはじめてアクセスすると（B），VMEM ドライバにてメモリアクセスをトラップし（C），VMEM プロセスが移動元からページ内容を取得する（D）．

VMEM は，（現状の実装は KVM のみを想定しているものの）基本的には，メモリアクセスをトラップしアクセス対象ページの内容を用意するための汎用的な機構であると見なせる．以下，我々が実装した VMEM について述べた後，KVM からの利用方法について述べる．

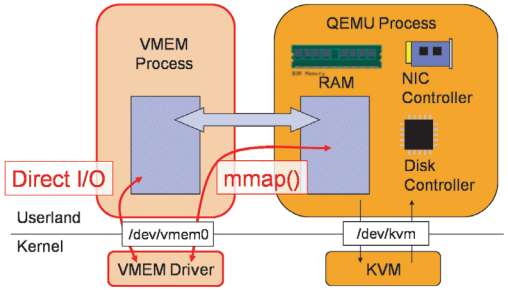

図 2　VMEM デバイスの概要
Fig. 2　Overview of VMEM device.

4.3.1　VMEM

我々が開発した VMEM（**図 2**）は，Direct I/O[*1] および `mmap()` によって，VMEM プロセスと `/dev/vmem0` を対象に `mmap()` を呼び出したプロセス（QEMU プロセス）の間で，メモリ領域を共有する仕組みである．しかも，`mmap()` を呼び出したプロセスが，あるページに最初にアクセスした瞬間をトラップできる．

まず，ホスト OS に対して VMEM ドライバ（`vmem.ko`）をロードし，専用キャラクタデバイス（`/dev/vmem0` 等）を作成する．そして，`/dev/vmem0` に対してメモリを割り当てるために，補助プログラムである VMEM プロセスを起動する．VMEM プロセスは，`mmap()` 用のメモリをユーザ空間で確保し，その領域を VMEM ドライバに通知する．ユーザ空間の VMEM プロセスとカーネル空間の VMEM ドライバは，Linux カーネルの Direct I/O 機能により，確保したメモリページを共有する．

そして，あるプロセス（QEMU プロセス）が `/dev/vmem0` を引数として `mmap()` を呼び出すと，VMEM ドライバは VMEM プロセスが確保したメモリ領域を提供する．結果，`mmap()` を呼び出したプロセスと VMEM プロセスが，同一のメモリ領域を共有することになる．VMEM プロセスは，`mmap()` を呼び出したプロセスによるメモリ領域の更新内容を観察できるほか，その内容を書き換えることもできる．

`mmap()` を行ったプロセスがメモリ領域中のあるページに初めてアクセスした際には，ページフォルトが発生してプロセスの実行が中断され，カーネルによって VMEM ドライバのページフォルトハンドラが呼ばれる．VMEM ドライバは，ページフォルトの発生を対象ページ番号とともに VMEM プロセスに通知する．VMEM プロセスは対象ページの内容を準備して，その完了を VMEM ドライバに通知し，`mmap()` を行ったプロセスの実行を再開する．

4.3.2　KVM による VMEM の利用

KVM は，VM のメモリ領域を QEMU プロセス中のユーザランドメモリ空間に確保する．そこで，移動先ホストで KVM が VM のメモリ領域を割り当てるコードにおいて，`MAP_ANONYMOUS` を引数とした `mmap()` の呼び出しを，`/dev/vmem0` をマッピングする `mmap()` の呼び出しに変更する．また，VM を最初に起動する移動元ホストでは，VMEM ドライバを用いないで単にメモリファイルシステム上で確保したファイル（`/dev/shm/kvm/mem0`）を `mmap()` の対象とするよう変更する．このファイルを対象として `read()` 等を実行すれば VM のメモリ内容を読み込める．実行ホスト切替え後には，このファイルから切替え直前のメモリページ内容を取得する．

VM を再配置した後に，さらに続けて別のホストへ再配置することや元のホストに戻すことも可能である．この場合，移動元ホストにおけるマッピング対象も `/dev/vmem0` 等となる．

4.3.3　VM 移動処理

提案手法による VM の移動処理を**図 3** に示す．実行ホスト切替え後には，VM のメモリアクセスに応じて 2 段階目から 5 段階目の処理が繰り返される．

(0)　移動元ホストで VM を停止して，ゲスト OS の実行を中断する．QEMU プロセスのメモリ空間内には，停止時点のメモリ内容，CPU レジスタやデバイスの状態が存在する．
(1)　CPU レジスタとデバイスの状態を取り出し，移動先ホストに転送し QEMU プロセス中にロードする．そして移動先ホストで VM の実行を再開する．
(2)　VM がメモリにアクセスする．
(3)　もしそのメモリページが移動先ホストにおいて最初にアクセスされたものであれば，VMEM ドライバのページフォルトハンドラがよばれる．VM の実行を一時停止し (4)，(5) の処理を行う．
(4)　VMEM ドライバのページフォルトハンドラが VMEM プロセスにメモリページ取得を要請する．
(5)　移動元ホストからメモリページ内容を取得し VM メモリ領域に書き込む．VM の実行を再開する．

このオンデマンドのメモリ取得処理および後述するバックグラウンドでのメモリ取得処理

*1 ユーザ空間とカーネル空間との間でメモリページを共有する機構．共有に際してメモリコピーが発生しない．

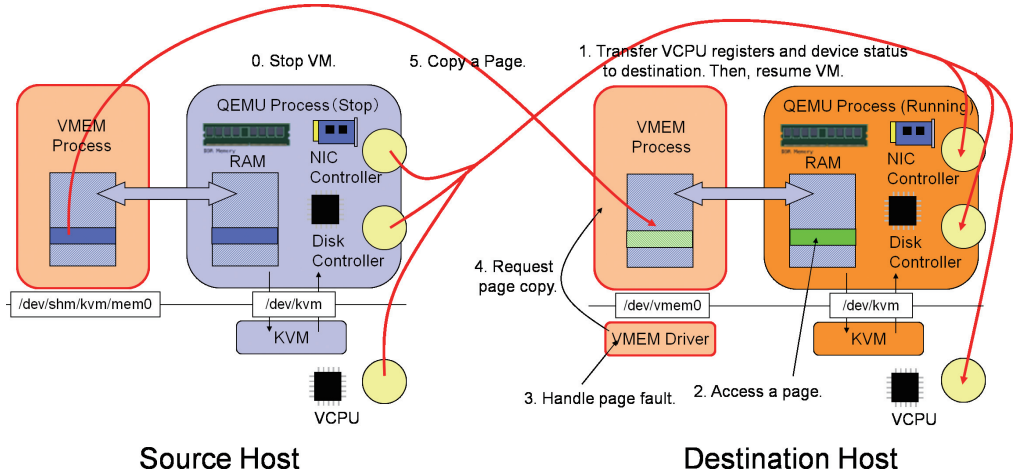

図 3　再配置動作の概要

Fig. 3　VMM relocation steps in our Post-copy Live Migration.

によって，最終的にすべてのメモリページが転送されると，移動元ホストへの依存性がなくなる．以後，移動先ホストのみで動作できる．移動先ホストでは，移動元ホストへの接続を終了する．また，移動元ホストでは，停止中の VM を破棄し，確保したメモリ領域を開放する．

また，メモリの遠隔取得による性能低下を抑制するため，以下の工夫を行っている．まず，移動元からのメモリ取得の際には，要求があった 1 ページ（4 KB）のみ転送するのではなく，そのページに続く複数のページを一度にコピーする．直後にアクセスされる可能性が高いページを，あらかじめ移動先ホストにコピーしておくことで，ネットワーク越しのページ転送回数を削減する．現在の実装では，連続する 128 ページの範囲において未転送のページを一度にコピーしている．

さらに，オンデマンドのメモリ取得と並行してバックグラウンドでのメモリ取得も動作させる．残りのメモリページをすべてコピーしメモリ領域全体の再配置を完了する．VM がアクセスする可能性が高いメモリ領域をできるだけ早い段階でコピーできれば，ネットワーク越しのページ転送回数を削減して性能低下を抑制できる．そこで，バックグラウンドのメモリ取得においては，移動後にページフォルトが頻出する領域から先にコピーしている．現在の実装では，4 MB のメモリ領域ごとにページフォルト回数をカウントし，アクセスが多い領域からメモリ内容をコピーしている．

5. 提案機構の実装と要件の確認

4.1 節で提案機構に対する要件として，既存 VMM に対する変更が軽微で機構全体が単純であり，さらにゲスト OS の改変が不要であることを述べた．本章では実装が要件を満たすことを確認する．

5.1 KVM に対する変更点

KVM-88 を対象に提案機構のプロトタイプを実装した．KVM においてはユーザ空間のコードのみに対して変更を加えた．すでに KVM に存在するプレコピー型マイグレーションと我々が今回実装したポストコピー型マイグレーションは，コマンドラインオプションによって選択的に利用できる．KVM に対する変更点は以下の 3 カ所である．

- VM のメモリ領域を割り当てる処理（`qemu_ram_alloc():exec.c`）において，ポストコピー型マイグレーション有効時には，KVM に本来備わっている `qemu_vmalloc()` の代わりに我々が作成した `qemu_vmalloc_postcopy()`（67 行）を呼ぶように変更した．`qemu_vmalloc()` は，`MAP_ANONYMOUS` を引数とした `mmap()` の呼び出しにより，VM が利用するメモリ領域を割り当てている．一方，`qemu_vmalloc_postcopy()` では，KVM

のコマンドラインオプション（-ram_file）で指定されたファイル名を，マッピング先として mmap() を呼び出して，メモリ領域を割り当てる．
4.3.2 項で述べたように，移動先ホストでは/dev/vmem0 等がマッピング先となる．また，仮想マシンを最初に起動するホストでは，メモリファイルシステム上のファイルがマッピング先となる．後者の場合は，マッピング対象のファイルを指定されたファイル名で新規に作成し，内部を 0x00 で初期化する．

- マイグレーションにおける移動元側の処理において，ポストコピー型マイグレーション有効時には，KVM に本来備わっている ram_save_live():vl.c の代わりに，我々が作成した ram_save_live_postcopy()（35 行）を呼ぶように変更した．ram_save_live() では，すべてのメモリページを移動先にコピーする．ダーティーページトラッキングによって，コピー中に変更されたページも再帰的にコピーする．一方，ram_save_live() を簡略化した ram_save_live_postcopy() では，VM から RAM として見える領域はコピーせず，フレームバッファや BIOS 領域のみ移動先にコピーする．コピー中はすでに VM を停止しているため，ダーティーページトラッキングも利用せず，再帰的なコピーも行わない．
  なお，マイグレーションにおける移動先側の処理においては，変更は不要である．移動元から移動先に送信されるページデータには，VM から見たページアドレスが付随しているため，移動先側の処理（ram_load()）は共通化できた．
- 新たに追加したコマンドラインオプション（-ram-file および-postcopy）を処理するために，vl.c および qemu-options.hx [*1] を約 10 行変更した．

以上のように，提案機構によって，KVM に対する変更点を軽微なものにとどめながら，ポストコピー型マイグレーションを実装できた．変更量も百数十行程度と少なくできた．

5.2 VMEM の実装

VMEM の実装においては，カーネル空間に存在するデバイスドライバをできる限り単純化することに努めた．4.3.1 項で述べたように，VMEM ドライバは，ページフォルトの発生をユーザ空間に通知し，対象ページの取得の完了を待つことのみ行う．ページの内容を移動元ホストから取得し，その内容を VM のメモリ領域に書き込む処理は，ユーザ空間の VMEM プロセスで実装した．

VMEM ドライバはローダブルモジュールとして実装した．ソースコードは約 500 行程度である．ページフォルトハンドラの記述（80 行）と，VMEM プロセスとの通信に用いる ioctl() の記述（90 行）が中心である．ページフォルト発生時には，

- カーネルによって，VMEM ドライバのページフォルトハンドラが呼び出される．ハンドラは，ページフォルトを起こしたプロセス（QEMU プロセス）の実行をブロックする，
- あらかじめ，VMEM プロセスは VMEM_IOC_GET_PENDINGS フラグを引数とした ioctl() を呼び出して，ページフォルトの発生を待っている．ページフォルトが発生すると，VMEM ドライバによって ioctl() のブロックが解除されて，対象ページのアドレスが通知される，
- VMEM プロセスは，ページ内容を VM のメモリ領域に書き込む処理が完了すると，VMEM_IOC_PUT_PENDINGS フラグを引数とした ioctl() を呼び出して，ページフォルト処理の完了を VMEM ドライバに通知する，
- VMEM ドライバは，QEMU プロセスのブロックを解除する，

という処理が行われる．その他，キャラクタデバイスドライバに要求されるコールバック関数を実装した．

VMEM プロセスは，ページ内容の取得に単純なストレージ I/O プロトコルである NBD（Network Block Device）プロトコルを用いている．我々が開発している NBD クライアント/サーバプログラム[18),19)] の大半を流用することができた．バックグラウンドコピーにおいて転送速度を任意の値に設定できる．

5.3 要件の確認

以上のように，メモリアクセスをトラップしメモリページを取得する処理を KVM から VMEM という外部のコンポーネントに分離することで，KVM 本体への変更を軽微なものにとどめられた．また，VMEM 自体も，カーネル空間で動作するドライバを小さくすることで，簡潔な実装となった．ポストコピー型マイグレーションを実現するシステム全体が，極力単純な形で実現できたと考えている．

また，提案機構は，完全仮想化を実現する KVM の特徴を失うことなく，ポストコピー型マイグレーションを実現している．ゲスト OS に対する改変が不要であるという点を達成できた．

6. 評 価 実 験

提案手法の性能を検証すべく，評価実験を行った．ウェブサーバのベンチマークである SPECweb2005 [20)] を動作させながら，ライブマイグレーションを行った．実験環境を**図 4**

⋆1 Makefile 中でシェルスクリプトで処理されて qemu-options.c と qemu-options.h が生成される．

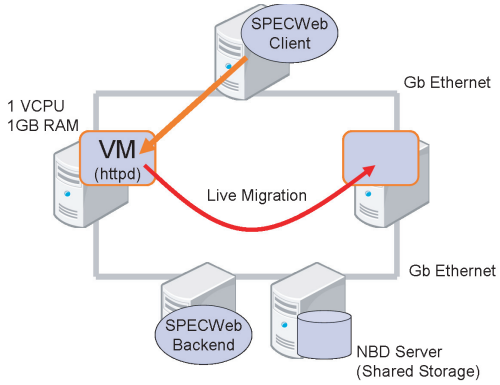

図 4　実験環境
Fig. 4　Experiment environment.

に示す．移動元・移動先の物理ホスト（Intel Core 2 Duo E6305，4 GB RAM）間はそれぞれ Gigabit Ethernet の LAN に接続されている．データセンタのプライベートネットワークを想定したネットワークには，移動元・移動先双方のホストからアクセスできる共有ストレージを用意し，マイグレーション前後に仮想ディスクアクセスを継続できるようにしている．また，SPECweb においてデータベースの振舞いをエミュレーションするバックエンドシミュレータノードも設置している．VM（1 CPU Core，1 GB RAM）上では Apache ウェブサーバを動作させ，パブリックネットワークを想定したネットワークから SPECweb クライアントで接続する．マイグレーションに関する通信はプライベートネットワークで行う．

6.1　提案機構（バックグラウンドコピー無効）

SPECweb クライアントからの同時接続数を 200 と設定して，インターネットバンキングのウェブサイトをシミュレーションするベンチマーク（SPECweb Banking）を走らせた．同時接続数 200 とは，用いた VM が安定的に対応可能な接続数の最大値である．ベンチマーク中 VM の CPU 負荷はほぼ 100%を推移する．開始後 150 秒付近でライブマイグレーションを行い，実行ホストを切り替えている．ポストコピー型の実行結果を**図 5**，**図 6**，**図 7**，**図 8** に示す．この実験ではオンデマンド取得のみ動作させバックグラウンドコピーは無効にしている．

実行ホストの切替え自体は 1 秒程度で完了し，すぐさま移動先ホストで VM が再開した．

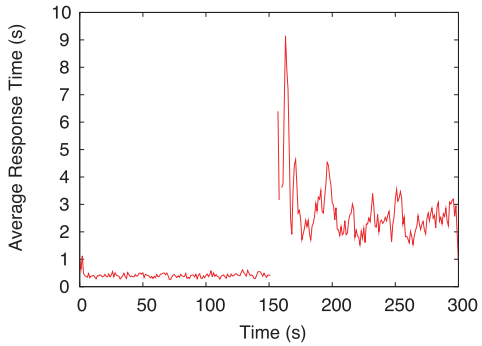

図 5　リクエスト応答時間の時間遷移（提案機構，バックグラウンドコピー無効，同時接続数 200）
Fig. 5　Response time (Post-Copy, Background copy disabled, 200 concurrent connections).

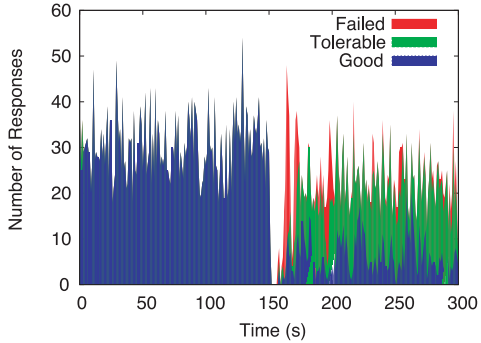

図 6　リクエスト応答品質（QoS）の時間遷移（提案機構，バックグラウンドコピー無効，同時接続数 200）．150 秒付近以前はほぼ Good（青）が占める．150 秒経過後はグラフ下部より Good，Tolerable（緑），Failed（赤）の順で描かれている
Fig. 6　Response QoS (Post-Copy, Background copy disabled, 200 concurrent connections). Good (blue) is dominant before 150s. After 150s, from the bottom of the graph, Good, Tolerable (green), and Failed (red) are painted.

しかし，図 5 が示すように，切替え直後にはリクエスト応答時間が増加（悪化）している．このとき，図 7 が示すように，移動元からのページ取得数が急激に増えている．その後 10 秒程度経過すると，ページ取得数の増加は緩やかになり始める．その後，図 5 の描画範囲では明確でないものの，リクエスト応答時間が徐々に改善されてくる．

図 6 が示すリクエスト応答品質（QoS）の時間遷移とは，インターネットバンキングを利

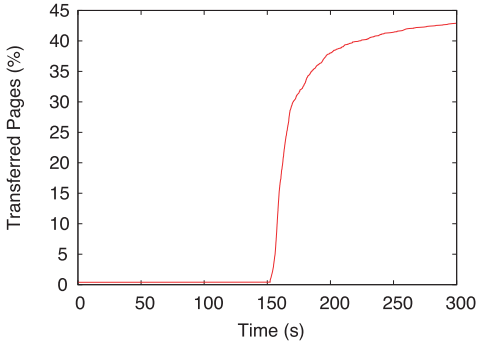

図 7　移動元からのページ取得数の時間遷移（提案機構，バックグラウンドコピー無効，同時接続数 200）
Fig. 7　Number of page retrievals (Post-Copy, Background copy disabled, 200 concurrent connections).

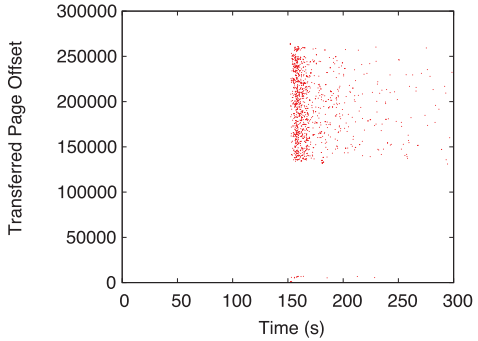

図 8　取得ページオフセットの時間遷移（提案機構，バックグラウンドコピー無効，同時接続数 200）
Fig. 8　Page offsets (Post-Copy, Background copy disabled, 200 concurrent connections).

用しているユーザが，どの程度ストレスを感じるかを示している．グラフ中の Failed 状態とは，ページの応答時間が遅いために，ユーザがウェブサイトを離れることを示す[⋆1]．実行ホスト切替え直後 10 秒程度は Failed 状態が大半を占めるものの，その後ページ取得数が緩やかになるにつれて状態は改善されていく．

図 8 は，あるページの内容がいつ取得されたのかを，横軸に取得時刻，縦軸に取得ページのオフセットをとって，1 つのページ取得の発生を 1 つの点として表している．どの時刻に VM のメモリ領域中のどの範囲のページが取得されたのか，という点を把握することを意図している．傾向として，150 秒以降にはページオフセットが 130000 から 260000 までの範囲が取得対象の大半となっている．必ずしもすべてのメモリ領域が実行ホスト切替え後即座に必要になるわけではない．そこで，バックグラウンドコピーによって，必要なメモリページをページフォルトよりも前にコピーできれば，実行ホスト切替え後の性能低下を抑制できるはずである．

⋆1 応答時間が 4 秒を超えた場合に Failed 状態，2 秒を超えた場合に Tolerable 状態となる．

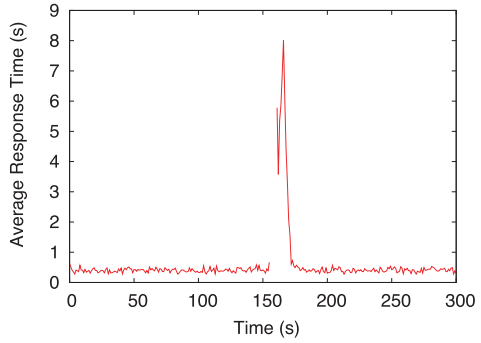

図 9　リクエスト応答時間の時間遷移（提案機構，バックグラウンドコピー有効，同時接続数 200）
Fig. 9　Response time (Post-Copy, Background copy enabled, 200 concurrent connections).

6.2　提案機構（バックグラウンドコピー有効）

次に，提案機構のバックグラウンドコピー機能を有効にして，再度実験を行った．実行ホストを切り替えて 5 秒経過した後から，バックグラウンドコピーを 800 Mbps の転送速度を設定して実行した．10 秒程ですべてのページの転送を完了した．**図 9** および**図 10** が示すように，バックグラウンドコピー完了後はすべてのメモリページが移動先ホストにキャッシュされることになるため，性能低下が発生していない．一方，実行ホスト切替え直後には一時的に応答状態が悪化している．しかし，この結果は VM が対応可能な最大同時接続数でのベンチマークによって得られている．実際には同時接続数が少ないほどメモリアクセスは緩やかになり，実行ホスト切替えの影響は軽微になる．たとえば，同時接続数を 100 に設定した場合（**図 11**）には，実行ホスト切替え直後のリクエスト応答時間の一時的な増加値は 2 秒程度にとどまる．ベンチマークの想定上は，ユーザの体感速度の悪化はほとんど存在しない（**図 12**）．そこで，今後提案実装を VM の配置決定機構に適用する際には，負荷が

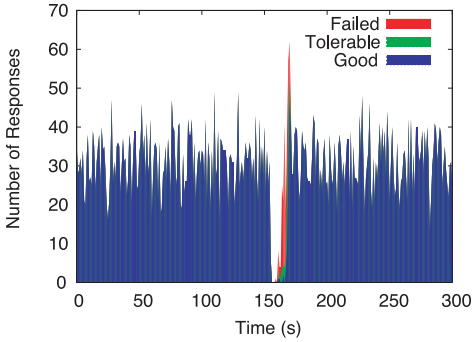

図 10　リクエスト応答品質（QoS）の時間遷移（提案機構，バックグラウンドコピー有効，同時接続数 200）．150 秒直後では Tolerable（緑）および Failed（赤）が出現している．それ以外の時刻には Good（青）のみ出現している

Fig. 10　Response QoS (Post-Copy, Background copy enabled, 200 concurrent connections). Just after 150s, Tolerable (green) and Failed (red) are shown. In the other period, Good (blue) is dominant.

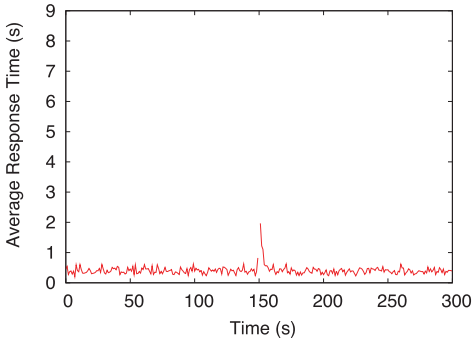

図 11　リクエスト応答時間の時間遷移（提案機構，バックグラウンドコピー有効，同時接続数 100）

Fig. 11　Response time (Post-Copy, Background copy enabled, 100 concurrent connections).

高まった VM を移動するのではなく，負荷が低い VM を優先して移動することが検討に値すると考えている．

本実験では，物理ホストは 2 つの CPU コアを有していながら，VM に割り当てた仮想 CPU は 1 つだけである．起動した VM の数も 1 つだけである．VM の CPU 負荷が 100%の場合でも，VM は 1 つの物理 CPU コアのみを占有する．そのため，バックグラウンドコピーを担うスレッドは，VM が占有する CPU コアとは別の CPU コアにスケジュールされて，VM に大きな影響を及ぼさずに実行できるため，800 Mbps という転送速度設定が可能であった．今後データセンタを想定した動的な VM 配置の最適化等に取り組む際には，物理 CPU コア数以上の VM を配置する場合や，複数の VM を同時に再配置する場合も想定される．利用できる CPU 資源やネットワーク帯域を考慮した再配置戦略を検討していきたい．

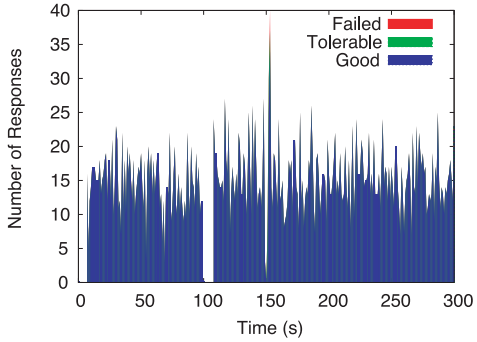

図 12　リクエスト応答品質（QoS）の時間遷移（提案機構，バックグラウンドコピー有効，同時接続数 100）．150 秒直後にわずかに Tolerable（緑）および Failed（赤）が出現している．それ以外の時刻には Good（青）のみ出現している

Fig. 12　Response QoS (Post-Copy, Background copy enabled, 100 concurrent connections). Just after 150s, Tolerable (green) and Failed (red) are slightly shown. In the other period, Good (blue) is dominant.

6.3　プレコピー型

KVM 本来のプレコピー型のライブマイグレーション機能を利用して再度実験を行った．開始後 150 秒付近でライブマイグレーションを開始した．しかし，**図 13** のネットワーク通信量の時間遷移が示すように，この実験中にはマイグレーションが完了することはなかった．KVM のプレコピー型マイグレーションの実装においては，マイグレーション開始後，移動元で更新されたページを繰り返し移動先に転送する．そして，残りページが一定以下になれば，移動元で VM を停止し残りのページを CPU やデバイスの状態とともに移動先へ転送する．しかし，KVM は実行ホスト切替え前後のダウンタイムを一定時間以下に抑えるため，残りのページ数が残りわずか[⋆1]になるまでは，移動元で VM を停止することはない．

⋆1 デフォルトの設定では転送レートから換算してダウンタイムが 3 秒以内となるページ数である．

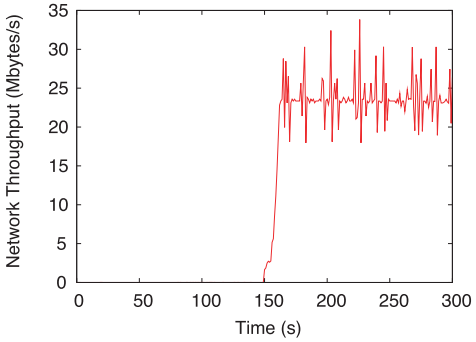

図 13　ライブマイグレーション通信量の時間推移（プレコピー型，同時接続数 200）
Fig. 13　Network traffic of live migration (Pre-Copy, 200 concurrent connections).

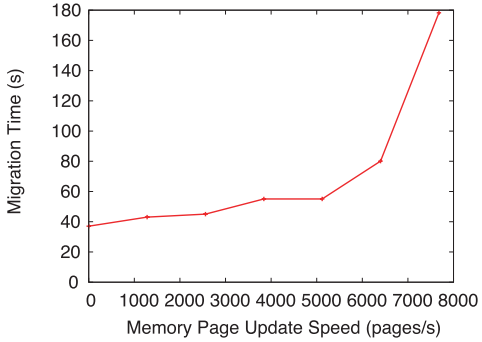

図 14　ページ更新速度に対するマイグレーション完了時間（プレコピー型）
Fig. 14　Migration time with different page update speeds (Pre-Copy).

このベンチマークにおいては，メモリページの更新がネットワーク越しの転送速度を上回ったため，残りのページ数が一定以下に減少することがなく，測定時間内においてマイグレーションを完了できなかった．

任意の速度で VM のメモリページを更新し続けるプログラムを作成して，VM のページ更新速度とマイグレーション時間との関係を調べた結果を**図 14** に示す．作成したプログラムは 900 MB のメモリ領域を確保し，各ページごとに 1 byte のデータを書き込み続ける．ページ更新速度が大きくなるとプレコピー型のマイグレーション時間は増加してしまい，ページ更新速度が 10,240 pages/s を超えるとマイグレーションは完了することはなかった．また，マイグレーションの完了までには最低でも約 40 秒程度時間を要してしまっている．2 章で述べたように，実行ホストの切替え時間が遅い点，その切替え時間がメモリ更新速度に依存してしまう点で，我々が目指す overcommit を許容する VM 配置システムには適していないと考える．

6.4　マイグレーション完了時間の比較

図 15 および**図 16** において，VM を移動元・移動先双方の計算機間で往復させた結果を示す．3 つのグラフは上から順に，プライベートネットワークの使用帯域，移動元・移動先での CPU 使用率である．グラフ中の縦線は実行ホストが切り替わった時刻である．ゲスト OS 上では，6.3 節のメモリ更新プログラムをメモリ更新速度を 5120 pages/s に設定して動かしている．初めに移動元ホストで VM を起動し，50 秒が経過してから移動先へライブマイグレーションさせた．マイグレーションが完了してから 60 秒後に，再び移動元ホストへ VM をライブマイグレーションさせた．KVM 本来のプレコピー型ライブマイグレーションにおいては，実行ホストの切替えおよびマイグレーションの完了に約 65 秒を要している．一方，提案機構では迅速に実行ホストを切り替えて，その後約 10 秒ほどでメモリコピーも完了している．提案機構においては，メモリコピーの間に移動元のメモリイメージは更新されることはなく，バックグラウンドコピーは大きな転送バッファで効率的に転送できる．一方，プレコピー型の機構では，メモリコピー中であっても，移動元でメモリページが更新され続けている．VM の全メモリページの中から更新されたページを抽出し，繰り返し移動先に転送する必要がある．結果，メモリの転送速度は実質 30 Mbyte/s 程度となってしまい，マイグレーション完了時間が長期化している．

なお，提案機構はプレコピー型と比べて CPU 使用率が高い．図 16 の CPU 使用率グラフにおいて移動先ホストでみられる最初の山は，VMEM デバイスが VM メモリ領域を確保し，受け手側の QEMU プロセスが起動したためである．その後，移動元ホストでみられる最初の山の時点で，CPU レジスタとデバイス状態のコピーが発生し，実行ホストが切り替わっている．移動先ホストで，メモリコピー完了後に約 5%の CPU を消費しているのは，ページフォルトが発生しているためである．時刻 120 秒付近までにすべてのメモリページが一度アクセスされたため，それ以降はメモリアクセスにともなってページフォルトが発生することはない．現在の実装では，すでに VM のメモリ領域内にページ内容がコピーされていても，`mmap()` の仕組み上，VM があるページに最初にアクセスしたときにページフォルトが発生してしまう．今後の発展的な実装において，ページテーブルエントリを直接変更

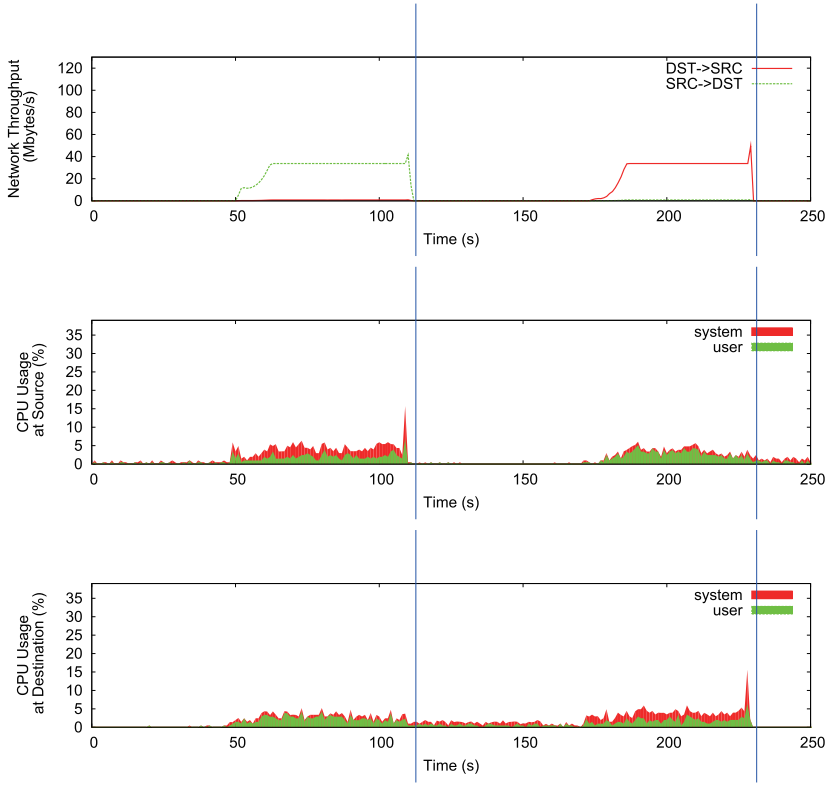

図 15　プライベートネットワーク通信量および移動元・移動先ホストの CPU 使用率の時間遷移（プレコピー型）
Fig. 15　Private network throughput and CPU usage at source/destination hosts (Pre-Copy).

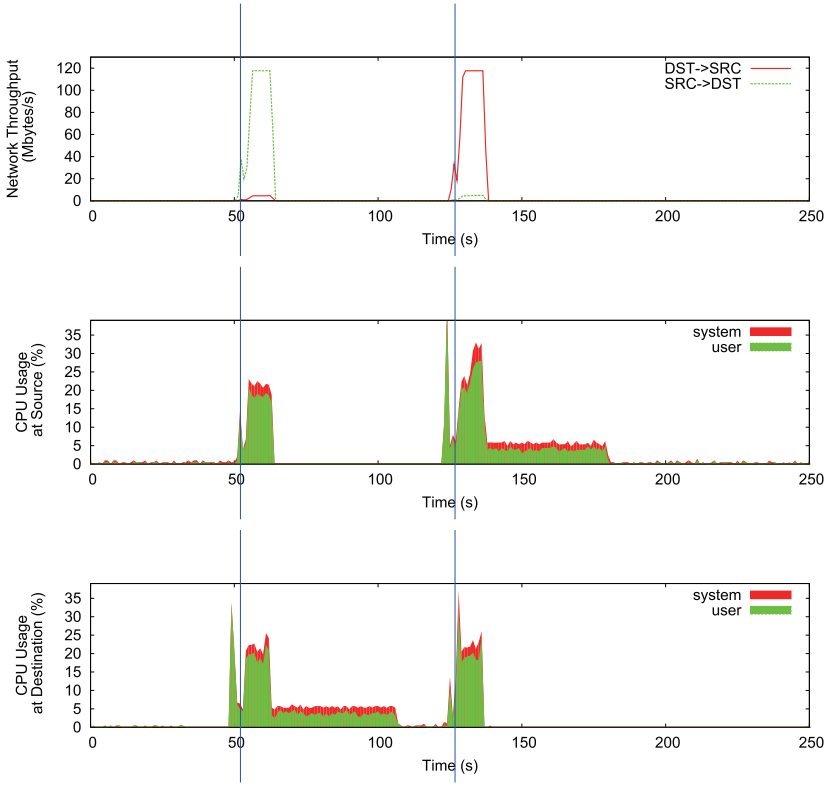

図 16　プライベートネットワーク通信量および移動元・移動先ホストの CPU 使用率の時間遷移 (提案機構)
Fig. 16　Private network throughput and CPU usage at source/destination hosts (Post-Copy).

するよう拡張すれば不要なページフォルトを避けられると考えている．

また，提案機構において，時刻 50 秒付近および 126 秒付近で 10 秒間ほど，移動元と移動先双方において CPU 使用率が 25%程度に上昇している．提案機構はデータ転送期間が短い反面，データ転送の利用帯域が大きく，その間の CPU 使用率も高くなる．ただし，大量のデータ転送を担うバックグラウンドコピーのスレッドは，VM と同時並列的に動作する．マルチコア環境において，バックグラウンドコピーのスレッドが，QEMU プロセスとは別の物理 CPU コアでスケジューリングされていれば，VM の性能への影響は避けられると考えている．また，今後の課題として，マイグレーションのオーバヘッドを考慮した，複数 VM の再配置戦略を検討していきたい．

6.5　実装の現状

Linux および Windows 7（RC Build 7100）をゲスト OS として，ポストコピー型ライブマイグレーションの動作を確認している．たとえば，ゲスト OS の Windows 7 上で YouTube の動画を再生しながら，数百ミリ秒で異なる物理ホストへ VM を移動できる．実行ホストの切替え直後には，遠隔のメモリページ取得が大量に発生するものの，Gigabit Ethernet

の LAN 環境では，もたつくことなくスムーズに再生できている．実行ホスト切替え時点で，移動先ホストに転送するデータは約 8 MB であり，大半を VGA デバイスの状態が占める．VGA デバイスがなければ 256 KB 程度まで小さくなり，さらにすばやい実行ホストの切替えが可能になる．ゲスト OS でメモリ更新速度の速いスクリーンセーバを動作させながら，2 つの物理ホストの間を往復移動させるデモも安定的に動作している[21]．

7. 議　　論

提案機構は，最も一般的なオープンソース VMM の 1 つである KVM を念頭に設計した．KVM が VM のメモリ領域の割当てを，ユーザ空間で `mmap()` を利用して行う点に着目している．他方代表的な VMM である Xen は KVM とは大きく構造が異なるため，提案機構を適用することは難しいと考えられる．3 章において取り上げた Snowflock では，VM のメモリアクセスをトラップするために，Xen ハイパーバイザのシャドページテーブル作成処理を拡張している．移動元ホストからのページ内容の取得はドメイン 0 が担っている．ハイパーバイザが提供する API を利用して，ドメイン 0 で準備したページを，VM（ドメイン U）のページフォルトアドレスにマップし直している．提案機構も Snowflock も，対象 VM やハイパーバイザとは別のコンポーネント（それぞれ VMEM およびドメイン 0）で，移動元ホストからページ内容を取得している点は類似している．しかし提案機構においては，メモリアクセスのトラップ処理も，対象 VM やハイパーバイザとは別のコンポーネント（VMEM）が担当し VMM への変更を小さく抑えている点が異なっている．

8. ま と め

本稿では，既存 VMM への適用が容易でゲスト OS の改変も不要な，ポストコピー型ライブマイグレーション機構を提案した．VM のメモリアクセスをトラップする機能と移動元ホストからページ内容を取得する機能の両方を KVM の外部で実装することで，KVM への変更量を軽微なものにとどめることができた．また提案機構全体を単純な実装とすることができた．できるだけ早い時期に，オープンソースコミュニティで開発される KVM に対して提案機構を統合し，実証的な環境で広く利用することを意図している．評価実験においては，負荷が高いウェブサーバ VM であっても，迅速に実行ホストを切り替えられることを確認した．オンデマンドコピーと並行して動作するバックグラウンドのメモリコピーによって，実行ホスト切替え後の性能低下を一時的なものにとどめることができた．また，メモリページを含む VM のすべての状態を再配置するのに要する時間も，プレコピー型のマイグレーション機構よりも短縮できた．今後，本稿で提案した高速マイグレーションを利用して，計算機クラスタにおける動的な VM 配置の決定・変更機構の開発に取り組む．実際の資源消費量に応じて VM 配置を動的に再構成することで，より高密度な VM 集約が可能になる．その際，負荷が低い VM を再配置対象とすることで，ゲスト OS に対する再配置の影響を小さくできると考えている．

謝辞　本研究は科研費（20700038）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）の助成を受けたものである．

参 考 文 献

1) Amazon Elastic Compute Cloud. http://aws.amazon.com/ec2
2) Nakada, H., Hirofuchi, T., Ogawa, H. and Itoh, S.: Toward virtual machine packing optimization based on genetic algorithm, *Distributed Computing, Artificial Intelligence, Bioinformatics, Soft Computing and Ambient Assisted Living* (*Proc. International Symposium on Distributed Computing and Artificial Intelligence 2009*), Vol. 5518 of Lecture Notes in Computer Science, pp.651–654. Springer (June 2009).
3) Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I. and Warfield, A.: Xen and the art of virtualization. *Proc. 19th ACM Symposium on Operating Systems Principles*, ACM Press (2003).
4) VMware Infrastructure. http://www.vmware.com/
5) 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 Symposium on Networked Systems Design and Implementation*, pp.273–286, USENIX Association (2005).
6) Nelson, M., Lim, B.-H. and Hutchins, G.: Fast transparent migration for virtual machines, *Proc. USENIX Annual Technical Conference*, pp.25–25, USENIX Association (2005).
7) Lagar-Cavilla, H.A., Whitney, J.A., Scannell, A., Patchin, P., Rumble, S.M., deLara, E., Brudno, M. and Satyanarayanan, M.: SnowFlock: Rapid Virtual Machine Cloning for Cloud Computing, *Proc. 4th ACM European Conference on Computer Systems*, pp.1–12, ACM Press (2009).
8) Hines, M.R. and Gopalan, K.: Post-copy based live virtual machine migration using adaptive pre-paging and dynamic self-ballooning, *Proc. 5th International Conference on Virtual Execution Environments*, pp.51–60, ACM Press (2009).
9) Kivity, A., Kamay, Y., Laor, D. and Liguori, A.: kvm: the Linux virtual machine monitor, *Proc. Linux Symposium*, pp.225–230, The Linux Symposium (2007).
10) Mirkin, A. Kuznetsov, A. and Kolyshkin, K.: Containers checkpointing and live

migration, *Proc. Linux Symposium*, pp.85–92, The Linux Symposium (July 2008).
11) Gupta, D., Lee, S., Vrable, M., Savage, S., Snoeren, A.C., Varghese, G., Voelker, G.M. and Vahdat, A.: Difference engine: Harnessing memory redundancy in virtual machines, *Proc. 8th USENIX Symposium on Operating Systems Design and Implementation*, pp.309–322, USENIX Association (2008).
12) Magenheimer, D., Mason, C., McCracken, D. and Hackel, K.: Paravirtualized paging, *Proc. Usenix First Workshop on I/O Virtualization*, USENIX Association (2008).
13) Waldspurger, C.A.: Memory resource management in VMware ESX server, *Proc. 5th Symposium on Operating Systems Design and Implementation*, pp.181–194. ACM Press (2002).
14) Arcangeli, A., Eidus, I. and Wright, C.: Increasing memory density by using KSM, *Proc. Linux Symposium*, pp.19–28, The Linux Symposium (July 2009).
15) 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.13, pp.167–178 (2006).
16) Advanced Micro Devices: *AMD64 Architecture Programmer's Manual Volume 2: System Programming, Revision 3.14*. (Sep. 2007).
17) Bellard, F.: Qemu, a fast and portable dynamic translator, *Proc. USENIX Annual Technical Conference*, pp.41–41, USENIX Association (2005).
18) 広渕崇宏，小川宏高，中田秀基，伊藤　智，関口智嗣：仮想計算機遠隔ライブマイグレーションのための透過的なストレージ再配置機構．情報処理学会論文誌：コンピューティングシステム，Vol.ACS26, pp.152–165 (July 2009).
19) Hirofuchi, T.: xNBD. http://bitbucket.org/hirofuchi/xnbd/
20) Standard Performance Evaluation Corporation: SPECweb2005. http://www.spec.org/web2005/
21) 広渕崇宏，中田秀基，伊藤　智，関口智嗣：仮想マシンの瞬間的な実行ホスト切り替えを可能とするポストコピー型ライブマイグレーション機構，コンピュータシステム・シンポジウム 2009（ComSys2009）ポスター・デモンストレーション，情報処理学会 (Nov. 2009).
22) 広渕崇宏，中田秀基，伊藤　智，関口智嗣：瞬間的な実行ホスト切り替えを可能とする仮想マシンの高速ライブマイグレーション機構，日本ソフトウェア科学会研究会資料シリーズ No.62（インターネットカンファレンス 2009 論文集），日本ソフトウェア科学会 (Oct. 2009).

(平成 22 年 1 月 26 日受付)
(平成 22 年 6 月 2 日採録)

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 年京都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）．

中田　秀基（正会員）

1995 年東京大学大学院工学系研究科情報工学専攻博士課程修了．同年電子技術総合研究所入所．2001 年より独立行政法人産業技術総合技術研究所主任研究員．2001〜2005 年東京工業大学学術国際情報センター客員助教授．分散並列コンピューティング，プログラミング言語に興味を持つ．ACM 会員．博士（工学）．

伊藤　　智（正会員）

1982 年千葉大学理学部物理学科卒業．1987 年筑波大学大学院物理学専攻博士課程修了後，株式会社日立製作所に入社．中央研究所において材料シミュレーションの研究に従事し，以来ハイパフォーマンスコンピューティングに関する応用アルゴリズムの研究を推進．1999 年からは金融工学等を含むビジネス分野の研究にも従事．2002 年 6 月に独立行政法人産業技術総合研究所に入所．グリッド技術を中心に産業分野等への応用について研究開発を推進中．

関口　智嗣（正会員）

1982 年東京大学理学部情報科学科卒業．1984 年筑波大学大学院理工学研究科修了．同年電子技術総合研究所入所．情報アーキテクチャ部主任研究官．以来，データ駆動型スーパコンピュータ SIGMA-1 の開発等の研究に従事．2001 年独立行政法人産業技術総合研究所に改組．2002 年 1 月より同所グリッド研究センターセンター長．2008 年 4 月より同所情報技術研究部門長．並列数値アルゴリズム，グリッドコンピューティング，IT による地球環境の理解に興味を持つ．市村賞，情報処理学会論文賞，文部科学大臣表彰科学技術賞（研究部門）賞受賞．地理情報システム学会理事，日本応用数理学会評議員，SIAM，IEEE 各会員．Open Grid Forum 諮問委員．