仮想化システムの軽量なソフトウェア若化のための ゼロコピー・マイグレーション

大庭 裕貴[1]　光来 健一[1]

概要：仮想化システムには状態が時間とともに劣化していくソフトウェア・エージングが発生しやすいが，この現象にはソフトウェア若化と呼ばれる手法で対処することができる．ソフトウェア若化の際の仮想マシン（VM）のダウンタイムを削減するには，VM をあらかじめマイグレーションしておく必要がある．しかし，VM マイグレーションはシステムに大きな負荷をかけてしまい，そのシステム上で動作する VM の性能にも影響を及ぼす．そこで本稿では，**ゼロコピー・マイグレーション**を用いて，仮想化システムの軽量なソフトウェア若化を実現する *VMBeam* を提案する．VMBeam では，仮想化システムをソフトウェア若化する際には，**ネストした仮想化**を用いて同一ホスト上で別の仮想化システムを動作させ，その仮想化システム上にすべての VM をマイグレーションする．この際に，VMBeam は VM のメモリをコピーすることなく直接新たな仮想化システム上に再配置する．VMBeam を Xen に実装し，マイグレーションの時間の短縮およびシステム負荷の削減を確認した．

1. はじめに

仮想化システムでは一台の計算機の中で複数の仮想マシン（VM）を動作させることによって，物理マシンの台数を減らし，コストを削減することができる．しかし，仮想化システムは長時間連続で運用されることが多いため，ソフトウェア・エージング [1] と呼ばれる現象が発生しやすくなる．ソフトウェア・エージングとは動作しているソフトウェアの状態が次第に劣化していく現象である．この問題に対処するために，ソフトウェア若化 [1] と呼ばれる手法が提案されている．ソフトウェア若化の典型的な例はシステムの再起動である．しかし，仮想化システムを再起動するには，動作しているすべての VM を一旦停止させ，仮想化システムの再起動後に再び VM を起動し直さなければならない．そのため，VM 上のサービスを提供できないダウンタイムが発生する．

このダウンタイムを削減するために，VM を動作させたまま別のホストに移動させるマイグレーションが用いられている．仮想化システムを再起動する前にすべての VM を別のホストにマイグレーションしておけば，VM 上のサービスは再起動の影響を受けない．しかし，マイグレーションを行うにはネットワークを介して大きな VM のメモリイメージを転送しなければならないため，ホストやネットワークに大きな負荷をかけてしまう [2].

本稿では，**ゼロコピー・マイグレーション**を用いて，仮想化システムの軽量なソフトウェア若化を実現する *VMBeam* を提案する．VMBeam では，仮想化システムのソフトウェア若化を行う際には，**ネストした仮想化**を用いて同一ホスト上で別の仮想化システムを動作させる．そして，その仮想化システム上に VM をマイグレーションする．その際に，これらの仮想化システムが同一ホスト上にあることを利用して，ネットワークを介して VM のメモリイメージの転送を行う代わりに，仮想化システム間で直接 VM のメモリを再配置する．マイグレーション中は移送元と移送先の VM にメモリを共有させることにより，マイグレーション中に書き換えられた VM のメモリを再送する必要がない．

我々は VMBeam を Xen 4.2.2 [3] に実装し，仮想化システム間で VM のメモリを共有させる機能である**ゲスト間メモリ共有**を開発した．移送元と移送先の仮想化システムがハイパーバイザに VM のメモリ情報を渡すと，ハイパーバイザがそれらの VM のメモリを共有させる．実験の結果より，ゼロコピー・マイグレーションは従来のマイグレーションよりも最大 6.4 倍高速であった．VM 内でメモリが書き換えられる場合でも，マイグレーション時間，ダウンタイムともに増加しなかった．また，CPU 負荷を 29%に，メモリ負荷，ネットワーク負荷をほぼ 0%に抑えることができた．

以下，2 章で仮想化システムの従来のソフトウェア若化

[1] 九州工業大学
Kyushu Institute of Technology

の問題について述べ，3 章で VMBeam を提案する．4 章で VMBeam の実装について説明し，5 章で VMBeam を用いて行った実験について述べる．6 章で関連研究に触れ，7 章で本稿をまとめる．

2. 仮想化システムのソフトウェア若化

多数の VM を動作させる仮想化システムは長時間連続して運用されることが多いため，ソフトウェア・エージング [1] が発生しやすい．ソフトウェア・エージングはメモリの解放し忘れや，オープンしたファイルの閉じ忘れなどのバグのために，システムの性能が次第に低下していく現象である．例えば，Xen においては，VM のライブマイグレーションを 100 回実行すると管理 VM の空きメモリが 80%減少するバグがあったことが報告されている．また，VM のサスペンド・レジュームを実行すると空きディスク容量が 185MB ずつ減少するバグがあったことも報告されている [4].

このようなソフトウェア・エージングの問題に対処するために，ソフトウェアを正常な状態に戻すソフトウェア若化 [1] と呼ばれる手法が提案されている．ソフトウェア若化を行うことで，仮想化システムをソフトウェア・エージングが起きる前の状態に戻し，性能の低下を防ぐことができる．その最も単純な方法は仮想化システムを再起動することである．仮想化システムの再起動時にはハイパーバイザの再起動が必要となるが，その際にはハイパーバイザ上で動作しているすべての VM を一旦停止し，ハイパーバイザが起動してからすべての VM を起動し直す必要がある．そのため，仮想化システムのソフトウェア若化の際には，VM 上で提供されているサービスにダウンタイムが発生する．

ソフトウェア若化に伴う VM のダウンタイムを削減するために，VM のマイグレーションが用いられている．特に，ライブマイグレーション [5] では，移送元の仮想化システムは VM を動作させたままそのメモリイメージを転送し，移送先の仮想化システムは受け取ったメモリイメージを基に新しい VM を構築する．移送元ではマイグレーション中に書き換えられた VM のメモリを再び転送し，最終的に VM を停止して残りの状態を転送する．全ての VM を別のホストにマイグレーションしてからハイパーバイザを再起動することで，VM のダウンタイムを非常に短く抑えることができる．

しかし，VM マイグレーションはホストやネットワークに大きな負荷をかける．仮想化システム上のすべての VM をマイグレーションする場合には，合計で数 GB から数十 GB ものデータを転送しなければならない．その際に，メモリイメージの盗聴や改ざんを防ぐには，メモリデータを暗号化して転送する必要がある．そのため，ホストの CPU を占有したりメモリ帯域を圧迫したりすることにより，ホストのシステム性能の低下を引き起こす．また，マイグレーション専用のネットワークを用いていない場合は，メモリイメージの転送がネットワーク帯域を圧迫し，ホストのネットワーク性能に大きな影響を及ぼす．これらの負荷により，ホスト上で動作している VM の性能も低下してしまう．実際に，11 台の VM をマイグレーションしている間，VM 内のウェブサーバの性能が平均で 57%低下することが報告されている [2].

マイグレーション中のシステム全体の性能低下を抑えるために，マイグレーションの速度を抑える手法も提案されている [5]．マイグレーションで用いるネットワーク帯域を制限することで，ネットワークへの負荷を軽減し，その結果，システムへの負荷も軽減することができる．しかし，ライブマイグレーションでは書き換えられたメモリを再度，転送する必要があるため，マイグレーション時間が長くなるとトータルの負荷が増大する可能性がある．実際に，ネットワーク帯域を 500Mbps に制限した場合，11 台の VM をマイグレーションするのにかかる時間が 5 倍に増加したことが報告されている [2]．また，マイグレーションに時間がかかると，仮想化システムのソフトウェア若化時間の増大につながる．

3. VMBeam

本稿では，仮想化システムの軽量なソフトウェア若化を実現する *VMBeam* を提案する．VMBeam では，仮想化システムのソフトウェア若化を行う際には，**ネストした仮想化**を用いて同一ホスト上で別の仮想化システムを起動する．そして，**ゼロコピー・マイグレーション**を用いてその仮想化システム上にすべての VM をマイグレーションする．その後で，元の仮想化システムを終了させることで，仮想化システムを再起動することなくソフトウェア若化を完了する．ゼロコピー・マイグレーションは二つの仮想化システムが同一ホスト上にあることを利用して，仮想化システム間で VM のメモリの再配置を行う．

3.1 ネストした仮想化の利用

同一ホスト上で二つの仮想化システムを動作させるために，VMBeam はネストした仮想化を利用する．ネストした仮想化を用いると，VM の中で仮想化システムを動作させることができる．VMBeam におけるソフトウェア若化時のシステム構成を**図 1** に示す．本稿では，通常の仮想化システムにおけるハイパーバイザ，VM をそれぞれ**ホスト・ハイパーバイザ**，**ホスト** *VM* と呼び，ホスト VM 内で動作するものをそれぞれ**ゲスト・ハイパーバイザ**，**ゲスト** *VM* と呼ぶ．VMBeam では，ソフトウェア若化時にはホスト・ハイパーバイザ上で二つのホスト VM を動作させ，それぞれの中でゲスト・ハイパーバイザおよびゲスト VM を動作させる．

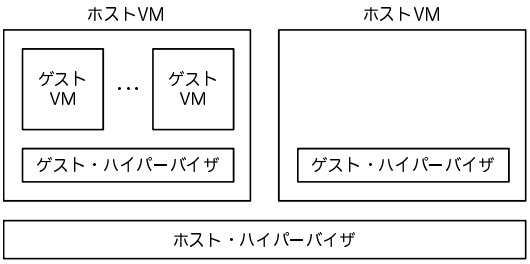

図 1 ソフトウェア若化時の VMBeam のシステム構成

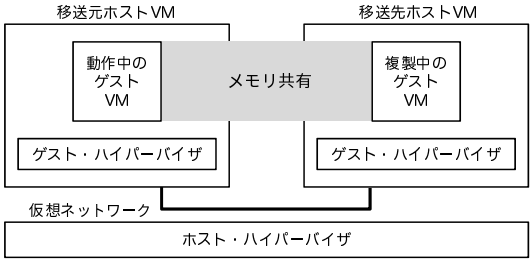

図 2 ゼロコピー・マイグレーション中のゲスト VM 間のメモリ共有

ネストした仮想化を用いることによる性能低下は小さくないが，そのオーバヘッドを軽減するために様々な手法が提案されている [6]．これらの手法を用いることで，性能低下を 6〜8%程度に抑えることができる．さらに，ソフトウェア若化時以外は脱仮想化 [7] を行うことで，ネストした仮想化のオーバヘッドを大幅に削減できる可能性がある．脱仮想化はハイパーバイザによる仮想化を行わないようにする技術である．ただし，本稿ではソフトウェア若化時のオーバヘッド削減に焦点を当てているため，ネストした仮想化のオーバヘッド削減についてはスコープ外である．

VMBeam では，ソフトウェア若化を行っている間，移送先の仮想化システムを動作させるための余剰リソースを必要とする．しかし，ゲスト VM はどちらかのホスト VM 上でしか動作しないため，ゲスト VM のために消費されるリソースは増加しない．そのため，VMBeam が必要とする余剰リソースは追加の仮想化システム内で動作するゲスト・ハイパーバイザの分だけである．

本稿では，ゲスト・ハイパーバイザをソフトウェア若化の軽量化の対象とする．2 章で述べたように，VM のマイグレーションやサスペンド・レジュームなどの処理を行う際にソフトウェア・エージングが起こりやすい [4] ため，それらの処理を頻繁に行うゲスト・ハイパーバイザは定期的なソフトウェア若化を必要とすると考えられる．一方，ホスト・ハイパーバイザはこのような処理を基本的に必要とせず，通常時には脱仮想化を行うことができるため，ソフトウェア・エージングは起こりにくいと考えられる．さらに，ホスト・ハイパーバイザには必要最低限の機能だけを持たせることで，ソフトウェア・エージングの発生を抑制することが可能となる．そのため，ゲスト・ハイパーバイザのソフトウェア若化を軽量化することはシステム全体にとってより効果的である．

3.2 ゲスト VM のゼロコピー・マイグレーション

VMBeam において従来のマイグレーションを行うだけではマイグレーションを高速化するのは難しい．VMBeam では，ゲスト VM のメモリイメージの転送をホスト内の仮想ネットワークを介して行うことができるが，ネストした仮想化における仮想ネットワークはネットワーク仮想化のオーバヘッドのために物理ネットワークよりも低速であることが多い．このオーバヘッドはマイグレーション時間の増加を引き起こす．さらに，VM マイグレーションにおけるクライアントとサーバが一つのホスト上で動作するため，システム負荷も 2 倍となってしまう．ネストした仮想化における高速な仮想ネットワーク [8] も提案されているが，メモリイメージの暗号化がマイグレーションにおけるボトルネックとなってしまう．

これらのオーバヘッドを削減するために，VMBeam は同一ホスト上で動作する仮想化システム間でのゼロコピー・マイグレーションを提供する．ゼロコピー・マイグレーションは，移送元のホスト VM 上で動作するゲスト VM のメモリを，移送先のホスト VM 上に新たに作成されたゲスト VM にコピーするのではなく再配置する．ライブマイグレーションを可能とするために，ゼロコピー・マイグレーションは二つのステップからなる．最初のステップでは，移送元のゲスト VM が VM マイグレーション中も動作し続けられるように，移送元と移送先のゲスト VM のメモリを共有する．本稿では，この機能を**ゲスト間メモリ共有**と呼ぶ．移送元で動作中のゲスト VM と移送先に複製中のゲスト VM がメモリを共有した状態を**図 2** に示す．次のステップでは，移送元のゲスト VM のメモリを解放し，移送先のゲスト VM へのメモリの再配置を完了する．

ライブマイグレーションは，書き換えられたメモリを再送するために差分が小さくなるまで転送を繰り返す必要があるが，ゼロコピー・マイグレーションでは一回の処理で転送を完了することができる．ゲスト間メモリ共有により，移送元のゲスト VM のメモリに対する変更が即座に移送先のゲスト VM に反映され，メモリの再送を必要としないためである．この手法により，マイグレーション時間を大きく削減することが可能となり，メモリに対する負荷が大きい VM に対しては特に有効となる．また，マイグレーションの最終段階で発生するダウンタイムの減少にもつながる．

ゼロコピー・マイグレーションはデータ転送に仮想ネットワークを使用しないため，ネットワーク負荷をゼロにすることができる．結果として，ネットワーク仮想化によるオーバヘッドがなくなり，CPU やメモリの負荷も削減することができる．また，メモリを再配置するだけで済むので，大量のメモリイメージをコピーするオーバヘッドも削減で

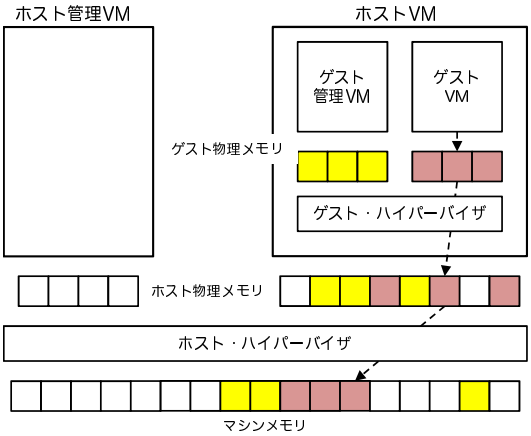

図 3 Xen のネストした仮想化におけるメモリモデル

きる．メモリをコピーする必要がないため，移送元と移送先のゲスト VM の外にデータが漏れることがなくなり，メモリイメージの暗号化の必要性もなくなる．さらに，VM のマイグレーションを行っている間にゲスト VM のメモリが変更されたかどうかを検出する必要がなくなる．

4. 実装

我々は VMBeam を Xen 4.2.2 [3] に実装した．ホスト・ハイパーバイザとして Xen を動作させ，その上でホスト VM を完全仮想化（HVM）ゲストとして動作させた．ホスト VM 上でゲスト・ハイパーバイザとして Xen を動作させ，その上でゲスト VM を HVM ゲストとして動作させた．Xen では VM を管理するためにドメイン 0 と呼ばれる管理 VM が用いられるが，ホスト VM を管理する VM を**ホスト管理** *VM*，ゲスト VM を管理する VM を**ゲスト管理** *VM* と呼ぶ．

4.1 メモリモデル

Xen のネストした仮想化におけるメモリモデルを**図 3** に示す．ホスト・ハイパーバイザは**マシンメモリ**と呼ばれるマシン全体で管理される物理メモリを扱い，その一部をホスト VM に割り当てる．このメモリだけがホスト VM で管理される物理メモリとなり，**ホスト物理メモリ**と呼ばれる．ゲスト・ハイパーバイザはこのホスト物理メモリの一部をさらにゲスト VM に割り当てる．このメモリだけがゲスト VM で管理される物理メモリとなり，**ゲスト物理メモリ**と呼ばれる．マシンメモリにはマシンフレーム番号（MFN），ホスト物理メモリにはホスト物理フレーム番号（HPFN），ゲスト物理メモリにはゲスト物理フレーム番号（GPFN）が順番に割り振られ，管理される．

MFN から HPFN，HPFN から GPFN への対応はそれぞれホスト・ハイパーバイザ，ゲスト・ハイパーバイザが M2P テーブルを用いて管理を行う．また，HPFN から MFN，GPFN から HPFN への対応はそれぞれホスト・ハイパーバイザ，ゲスト・ハイパーバイザが P2M テーブルを用いて管理を行う．Intel VT-x を用いる場合，P2M テーブルはホスト・ハイパーバイザ，ゲスト・ハイパーバイザそれぞれに拡張ページテーブル（EPT），仮想 EPT として実装されている．

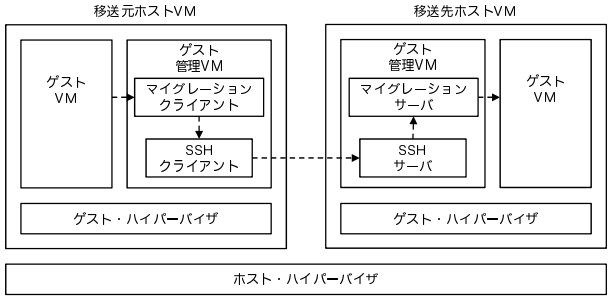

図 4 ネストした仮想化における従来のマイグレーション

4.2 ゼロコピー・マイグレーション

VMBeam では，移送元のゲスト管理 VM が，移送先のゲスト管理 VM にゲスト VM のメモリイメージを転送する．ネストした仮想化を用いたシステムで従来のマイグレーションを行う場合の流れを**図 4** に示す．まず，移送元のゲスト管理 VM 内のマイグレーション・クライアントはゲスト VM に割り当てられているメモリページをマップする．次に，ゲスト VM のメモリの内容を読み込み，そのメモリイメージを仮想ネットワーク上に構築された SSH トンネルを介して移送先へ転送する．一方，ゼロコピー・マイグレーションでは，ゲスト VM のメモリに対応する GPFN をゲスト・ハイパーバイザに渡すだけでメモリイメージの送信処理が完了する．メモリイメージのマップ，SSH による暗号化，ネットワーク転送は行われない．

移送先のゲスト管理 VM では，マイグレーション・サーバが新しく空のゲスト VM を作成する．従来のマイグレーションでは，そのゲスト VM のメモリページをマップし，SSH トンネルを介して受け取ったメモリの内容を書き込む．一方，ゼロコピー・マイグレーションでは，新しいゲスト VM のメモリに対応する GPFN をゲスト・ハイパーバイザに渡すだけでメモリイメージの受信処理が完了する．実際のメモリイメージの転送処理は，ゲスト・ハイパーバイザによって呼び出されるホスト・ハイパーバイザ内で行われる．ホスト・ハイパーバイザはゲスト間メモリ共有により移送元のゲスト VM のメモリページを移送先のゲスト VM と共有させる．それゆえ，メモリページをマップする必要がなく，SSH による復号も行われない．

ゼロコピー・マイグレーションは，一回のメモリ転送で VM の全てのメモリページの転送を完了する．従来のマイグレーションでは，移送元のゲスト管理 VM は，VM によるメモリページの書き換えを検出するために，ゲスト VM をログダーティ・モードで動作させる必要がある．そして，メモリ転送の各イテレーションの最後にゲスト・ハイパー

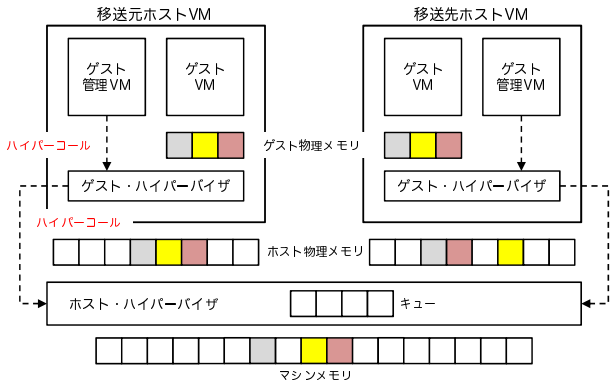

図 5 仮想化システムを経由したゲスト間メモリ共有

バイザからダーティ・ビットマップを取得し，そのダーティ・ビットマップに基づいて書き換えられたメモリページの再送を行う．ゼロコピー・マイグレーションでは，ゲスト VM を性能低下の原因となるログダーティ・モードで動作させる必要がなく，ダーティ・ビットマップをチェックしたり，メモリページを再送したりする必要もない．再送を行わなくても，メモリページに対する全ての変更を即座に移送先のゲスト VM に反映することができる．

4.3 ゲスト間メモリ共有

ゲスト間メモリ共有は，異なるホスト VM 上で動作するゲスト VM 間でメモリページを共有させる機能である．ゲスト管理 VM がゲスト・ハイパーバイザを呼ぶ流れを**図 5** に示す．移送元のゲスト管理 VM はハイパーコールを用いて，対象のゲスト VM の ID と，メモリページに対応する GPFN の配列をゲスト・ハイパーバイザに渡す．移送先のゲスト管理 VM も同様の情報をゲスト・ハイパーバイザに渡す．これらのゲスト管理 VM が動作するホスト VM は異なるため，呼び出されるゲスト・ハイパーバイザも異なる．

それぞれのゲスト・ハイパーバイザは渡された GPFN の配列を HPFN の配列に変換し，ホスト・ハイパーバイザを呼び出す．ゲスト・ハイパーバイザはこの変換を，仮想 EPT を用いて行う．ホスト・ハイパーバイザは，渡された HPFN の配列を基に移送元のホスト VM のメモリページを移送先のホスト VM と共有させる．まず移送元の HPFN に対応する EPT エントリを探し，変換先の MFN を取得する．そして，移送先の HPFN からも EPT エントリを探し，そのエントリに取得した MFN を設定する．その結果，移送元と移送先のホスト VM の HPFN は，EPT によって同じ MFN に変換されるようになる．

二つのゲスト・ハイパーバイザから呼び出されるハイパーコールを同期させずに済ませるために，移送元のホスト VM から渡される HPFN を一時的に保持しておくキューをホスト・ハイパーバイザに用意する．移送元から呼び出された時に，ホスト・ハイパーバイザは HPFN をキューに追加する．移送先からホスト・ハイパーバイザが呼び出された時に，キューが空でなければ，渡された HPFN とキューの中の HPFN とに対応するメモリページを共有させる．キューが空の場合，ホスト・ハイパーバイザはエラーを返し，移送先に再度ハイパーコールを呼び出させる．

4.4 ゲスト管理 VM 間の高速通信

VMBeam では，ゲスト間メモリ共有を利用することで，ゲスト管理 VM 間での通常の通信におけるオーバヘッドも削減する．VM マイグレーションでは，メモリや CPU の状態などのデータも移送先のゲスト管理 VM に転送する必要がある．そのために，まず，ゲスト管理 VM は mmap システムコールを用いてスワップ・アウトされないメモリを確保する．次に，ゲスト管理 VM は，確保したメモリページに対応する GPFN を OS のカーネルから取得する．最後に，ゲスト管理 VM はゲスト・ハイパーバイザを介してホスト・ハイパーバイザを呼び出し，移送元のゲスト管理 VM のメモリを共有する．そして，移送元と移送先のゲスト管理 VM はこの共有メモリを介して高速な通信を行う．

5. 実験

我々はゼロコピー・マイグレーションの有効性を示すために，いくつかの実験を行った．

5.1 実験環境

実験には Intel Xeon E5-2665 (8 コア, 2.40GHz) の CPU, 32GB のメモリ，1TB の HDD，ギガビットイーサネットを搭載したマシンを 2 台用いた．ハイパースレッディングは無効にした．これらのマシンはギガビットスイッチで接続した．

VMBeam と既存システムとの比較を行うために，(1) VMBeam, (2) ネストした仮想化を用いた標準の Xen (Xen-Nest)，(3) Xen-Blanket [8]，(4) 2 つの物理マシンを用いた従来のシステム（Xen-Phys）について実験を行った．Xen-Nest では，**図 4** に示したように，ホスト VM 間でのゲスト VM のメモリイメージ転送に仮想ネットワークを利用する．Xen-Blanket は，ゲスト管理 VM に準仮想化ドライバを導入してホスト管理 VM のバックエンドドライバを利用することで，ネストした仮想化における仮想ネットワーク性能を向上させている．

(1)〜(3) のシステム構成では，ホスト・ハイパーバイザとして Xen 4.2.2 を用い，その上で 1 つのホスト管理 VM と 2 つのホスト VM を動作させた．また，ゲスト・ハイパーバイザとして Xen 4.2.2 または Xen-Blanket 4.1.1 を用い，その上で 1 つのゲスト管理 VM を動作させた．一方のホスト VM 上ではさらにゲスト VM を 1 つ動作させた．5.9 節の実験を除いて，ゲスト VM 内ではアプリケーションを動作させなかった．ホスト VM およびゲスト VM で

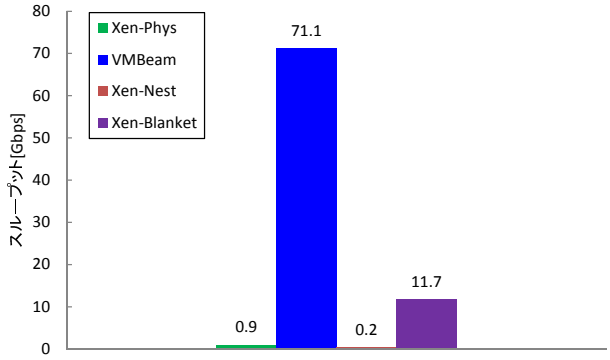

図 6 データ転送性能

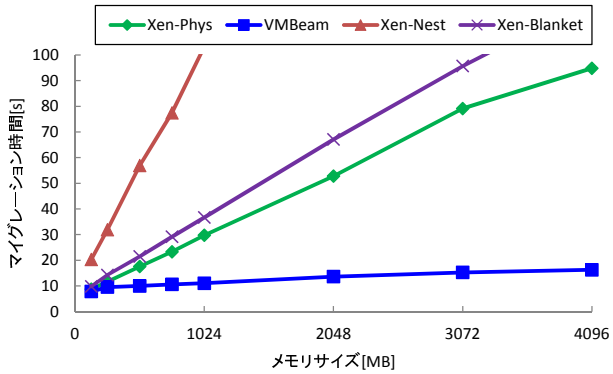

図 7 マイグレーション時間

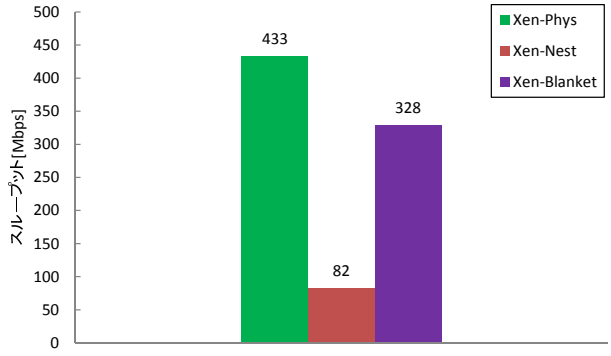

図 8 SSH トンネルを用いた場合のデータ転送性能

は HVM ゲストを動作させた．ホスト管理 VM では Linux 3.2.0，ゲスト管理 VM では Linux 3.5.0 を動作させた．

2 つのホスト VM には CPU を 3 個ずつ，メモリを 10GB ずつ割り当てた．ホスト管理 VM には残りの 2 個の CPU および 9.1GB のメモリを割り当てた．ゲスト VM には，ホスト VM に割り当てたリソースの内，1 個の CPU および 128MB〜4GB のメモリを割り当てた．5.9 節の実験を除いて，ゲスト管理 VM には 1 個の CPU および残りの 9.5GB のメモリを割り当てた．ゲスト管理 VM への CPU 割り当てを 2 個以上にするとほとんどの場合でマイグレーション時間が長くなったため，1 個だけ割り当てるようにした．

(4) のシステム構成では，ハイパーバイザとして Xen 4.2.2 を用い，1 つの管理 VM を動作させた．一方のマシンではさらに VM を 1 つ動作させた．管理 VM には 2 個の CPU および 29GB のメモリを割り当て，VM は上のゲスト VM と同じものを動作させた．以下，マイグレーション対象の VM のことをゲスト VM と総称する．

5.2 データ転送性能

通信性能を比較するために，仮想化システム間でのデータ転送のスループットを計測した．Xen-Nest と Xen-Blanket ではゲスト管理 VM で iperf を実行し，Xen-Phys では管理 VM で iperf を実行した．VMBeam についてはゲスト VM 間メモリ共有を用いたデータ転送性能を測定するベンチマークを作成して測定を行った．このベンチマークはゲスト管理 VM で動作させた．

4 つのシステムにおけるスループットを**図 6** に示す．VMBeam のスループットは他のどのシステムよりも大幅に高かった．Xen-Blanket の仮想ネットワークは Xen-Phys で使われる物理ネットワークの 12.5 倍高速であったが，VMBeam の 6.1 倍低速であった．Xen-Nest における仮想ネットワークは物理ネットワークより 3.8 倍低速であった．

5.3 マイグレーション時間

ゲスト VM に割り当てるメモリサイズを 128MB から 4GB まで変えて，マイグレーションにかかる時間を計測した．xl migrate コマンドの実行時間の平均を**図 7** に示す．VMBeam が最も高速にマイグレーションを行うことができ，ゲスト VM のメモリサイズが 4GB の場合でも 16.3 秒でマイグレーションが完了した．

Xen-Phys と比較すると，VMBeam はマイグレーションを 1.1〜5.8 倍高速に行えた．一方，ネストした仮想化を利用した他のシステムのマイグレーション性能は低下した．Xen-Blanket では 1.1〜1.3 倍低速になり，Xen-Nest で 2.3〜3.7 倍低速になった．Xen-Nest における性能低下の原因は，ネットワークスループットが低いことである．一方，Xen-Blanket における性能低下の原因は**図 8** に示すように，SSH による暗号化のオーバヘッドである．

5.4 ダウンタイム

ゲスト VM に割り当てるメモリサイズを 5.3 節の実験と同じように変えて，マイグレーション中のゲスト VM のダウンタイムを計測した．ダウンタイムの平均を**図 9** に示す．この結果より，ダウンタイムはメモリサイズにほぼ依存しないことが分かる．VMBeam のダウンタイムは 0.6 秒程度であるが，Xen-Phys より 0.2 秒程度長い．これはネストした仮想化のオーバヘッドのためである．マイグレーションの最終段階でゲスト VM の CPU 状態を取得するために多くのハイパーコールを発行する必要があり，ネストした仮想化の影響が大きい．

一方，Xen-Nest のダウンタイムは VMBeam より 1.3 倍程度長くなった．Xen-Nest の仮想ネットワークは低速で

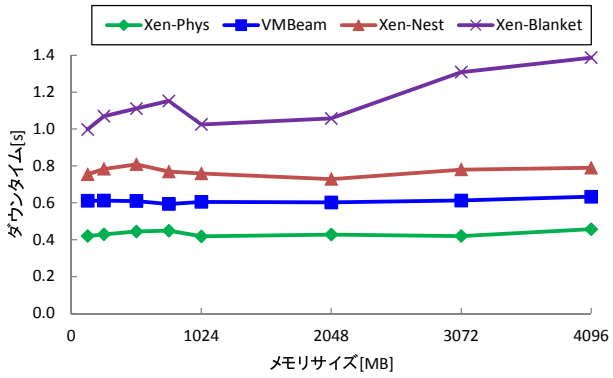

図 9 ダウンタイム

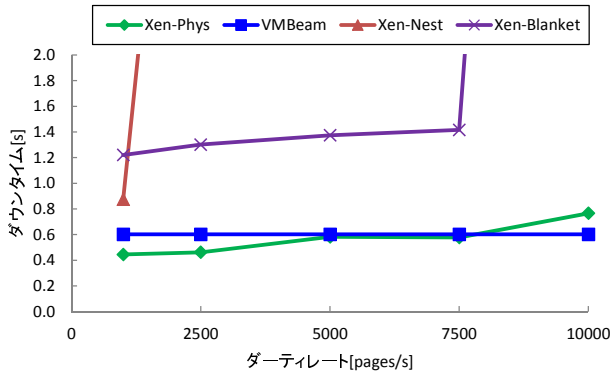

図 11 メモリ書き換えを行うゲスト VM のダウンタイム

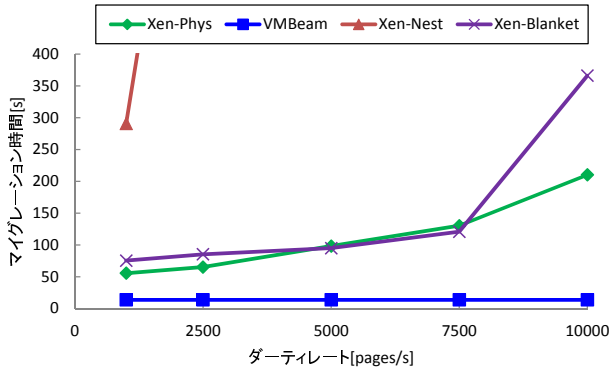

図 10 メモリ書き換えを行うゲスト VM のマイグレーション時間

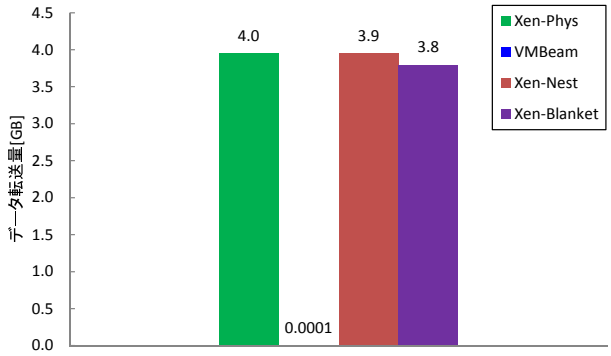

図 12 マイグレーション中のネットワーク転送量

あるため，最終段階での書き換えられたメモリの転送に時間がかかるためである．Xen-Blanket のダウンタイムは Xen-Nest よりもさらに 1.3〜1.7 倍低速であった．この原因は不明である．

5.5 ゲスト VM 内のメモリ書き換えの影響

ゲスト VM 内のメモリ書き換えがマイグレーション性能に及ぼす影響を調べるために，指定したレートでメモリページをダーティにしながらマイグレーションを行った．ダーティレートは毎秒 1000〜10000 ページまで変化させた．この実験では，ゲスト VM のダーティ・ビットマップを書き換えることで，正確なダーティレートを維持するようにした．ゲスト VM のメモリサイズは 2GB とし，ダーティにするメモリ領域は 1GB とした．それぞれのダーティレートにおけるマイグレーション時間およびダウンタイムを**図 10**，**図 11** に示す．

この結果より，VMBeam 以外のシステムではマイグレーション時間，ダウンタイムともに大きく影響を受けることが分かった．Xen-Phys ではマイグレーション時間が 2.7〜157 秒増加し，ダウンタイムが最大で 0.1 秒増加した．Xen-Blanket においてはマイグレーション時間が 8.1 秒〜299 秒増加し，ダウンタイムが最大で 13.8 秒増加した．Xen-Nest ではマイグレーション時間が 95.9〜841 秒増加し，ダウンタイムが最大で 101 秒増加した．

5.6 ネットワーク負荷

4GB のメモリを割り当てたゲスト VM をマイグレーションする際に，ネットワークを介して転送されるデータ量を計測した．マイグレーション中のトータルのデータ転送量を**図 12** に示す．VMBeam では他のシステムと違い，メモリイメージやその他のデータ転送にネットワークを用いないため，データ転送量を 0.003%以下に削減できている．トータルのデータ転送量が 0 にならないのは，実装上の問題により VM のコンフィグなどの転送にまだ仮想ネットワークを使用しているためである．

5.7 CPU 負荷

4GB のメモリを割り当てたゲスト VM をマイグレーションしている間のホスト全体の CPU 負荷を計測した．マイグレーション中の CPU 使用率の変化を**図 13** に示す．また，マイグレーション中に使われたトータルの CPU 時間を**図 14** に示す．VMBeam の CPU 使用率は Xen-Nest や Xen-Blanket より少し低いが，Xen-Phys と比較すると 2 倍程度の負荷がかかっていることが分かる．これは移送元と移送先の処理を同一ホストで行っているためである．しかし，トータル CPU 時間で比較すると，VMBeam の CPU 負荷は Xen-Phys の移送元，移送先と比較してそれぞれ 29%，31%の負荷に抑えられた．

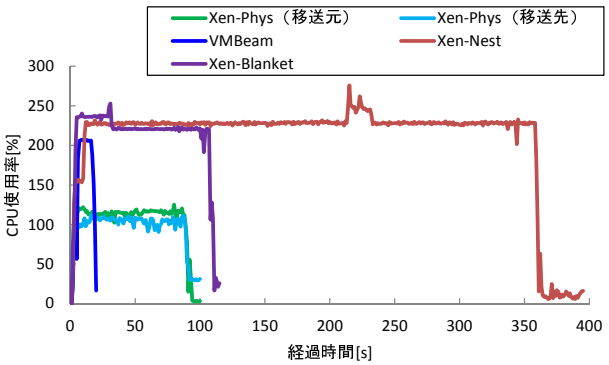

図 13 マイグレーション中の CPU 負荷

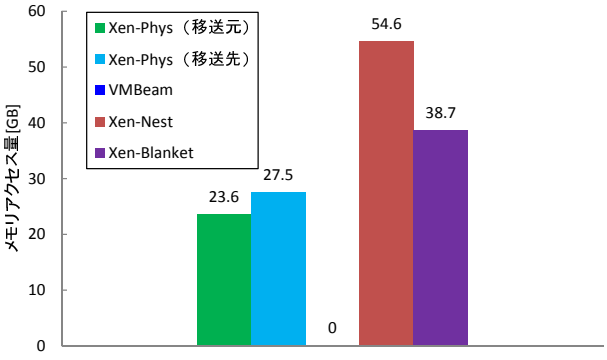

図 15 マイグレーション中のメモリアクセス量の概数

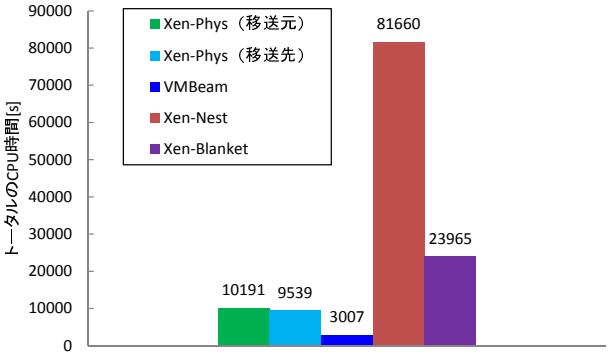

図 14 マイグレーション中のトータル CPU 時間

5.8 メモリ負荷

4GB のメモリを割り当てたゲスト VM をマイグレーションする際のメモリアクセス量を測定した．実際のメモリアクセス量を直接測定することはできなかったため，転送されたメモリページ数とメモリコピー回数から概算した．トータルのメモリアクセス量の概数を**図 15** に示す．VMBeam では移送元のゲスト VM のメモリを移送先に再配置するだけであるため，メモリアクセスは行われない．

Xen-Phys については，移送元の管理 VM のカーネルが VM のメモリを読み出し，ソケットバッファ経由で SSH クライアントのバッファに書き込む．SSH クライアントはそのデータを暗号化して SSH サーバに送信する．その際に，NIC がソケットバッファ上の送信データを DMA で読み出す．一方，移送先の管理 VM では，受信したデータを NIC が DMA でソケットバッファに書き込み，カーネルが SSH サーバのバッファに書き込む．SSH サーバはそのデータを復号してカーネル内のソケットバッファ経由でマイグレーション・サーバに送信する．マイグレーション・サーバは受信したデータを VM のメモリに書き込む．そのため，移送元ではゲスト VM のメモリサイズの 6 倍，移送先では 7 倍のメモリアクセスが行われると考えられる．

Xen-Nest の場合には，移送元のゲスト管理 VM での送信処理の後，ホスト管理 VM 上の QEMU が DMA をエミュレートしてデータを読み込み，TAP デバイス経由で移送先のホスト VM 用の QEMU に送信する．その QEMU は DMA をエミュレートして移送先のゲスト管理 VM のソケットバッファにデータを書き込む．その後，ゲスト管理 VM で受信処理が行われる．そのため，ゲスト VM のメモリサイズの 14 倍のメモリアクセスが行われると考えられる．一方，Xen-Blanket の場合には，移送元のゲスト管理 VM でソケットバッファに書き込まれたデータはホスト管理 VM のカーネル経由で，移送先のゲスト管理 VM の SSH サーバのバッファに直接書き込まれる．そのため，VM のメモリサイズの 10 倍のメモリアクセスで済むと考えられる．

5.9 ゲスト VM への影響

マイグレーションによるシステムへの負荷がゲスト VM の性能に及ぼす影響を調べるために，1GB のメモリを割り当てたゲスト VM 内で Apache ウェブサーバを動作させた．httperf を用いてマイグレーションを行わない通常時とマイグレーション中におけるスループットを測定した．ウェブサーバのためのネットワーク処理とマイグレーション処理がゲスト管理 VM で競合するため，この実験ではゲスト管理 VM に 2 個の CPU を割り当てた．**図 16** の結果より，どのシステムにおいてもマイグレーション中のスループットは低下していることが分かる．しかし，Xen-Phys では 12%しか低下していないのに対して，VMBeam では 29%，Xen-Nest では 65%も低下している．これは，VMBeam と Xen-Nest では仮想ネットワークがボトルネックになったためと考えられる．

一方，Xen-Blanket ではスループットの低下が 14%で，Xen-Phys とほぼ同程度の低下に抑えられている．これは，Xen-Blanket ではゲスト VM が高速な仮想ネットワークを使用できるためだと考えられる．Xen-Blanket と同じ仮想ネットワークを用いることで，VMBeam でもスループットの低下を抑えることができる可能性がある．また，ネストした仮想化を用いるシステムの通常時の性能は Xen-Phys の 34〜40%であったが，文献 [6] で報告されているように今後改善されていくと考えられる．

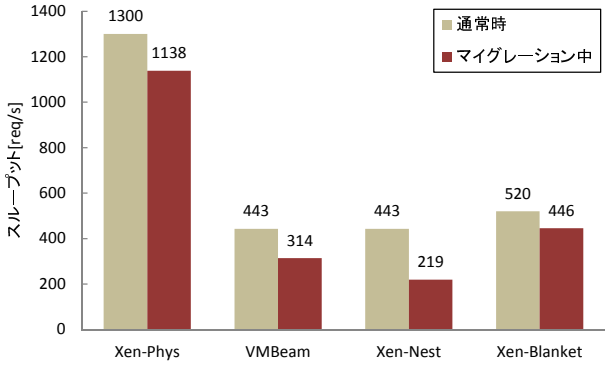

図 16 マイグレーションによるスループットの低下

6. 関連研究

Microvisor [7] は，システムのメンテナンスを別の VM 上で行い，メンテナンス後に VM 間でアプリケーションのマイグレーションを行う．ネストした仮想化を用いる点を除いて VMBeam は Microvisor に似ている．しかし，Microvisor ではメンテナンス時以外は仮想化を行わないようにする脱仮想化に焦点を当てているのに対して，VMBeam ではマイグレーションの高速化に焦点を当てている．

InfiniBand の RDMA を用いたマイグレーション [9] は，ハードウェアによる一回のコピーのみで完了することが可能となっている．VM のメモリイメージは，移送先のホストに新たに作成された VM のメモリに，RDMA を利用して直接コピーされる．しかし，マイグレーション中に変更されたメモリの再送は必要となる．また，メモリイメージの暗号化を行うことができず，暗号化を行うには，3 回のコピーを必要とする．

これまでに，VM マイグレーションを高速化する様々な手法が提案されてきた．ポストコピー・マイグレーション [10] は VM の CPU 状態だけを移送先ホストに移した後でオンデマンドにメモリページの転送を行う．この手法では，書き換えられたメモリを再送する必要がない．デルタ圧縮技術はマイグレーション中に転送されるメモリサイズを減らすのに用いられる [11]．PMigrate [12] はマイグレーションを並列に行うことでマイグレーション時間を削減する．しかし，多くの技術が VM マイグレーションによって使われる CPU 時間を増加させてしまう．VMBeam ではトータル CPU 時間を削減し，システムへの負荷を軽減できる．

VMBeam ではライブマイグレーションのために VM のメモリ共有を利用しているが，VM 間でのメモリ共有技術は主にメモリを節約するために開発されてきた．VMware ESX サーバは定期的に VM のメモリをスキャンし，VM 間で共通しているページを共有する [13]．Satori [14] は共有を意識したブロックデバイスを用いることで，短時間でも共有を行うことができる．Difference Engine [15] は似ているページも共有することができる．Potemkin [16] は既存の VM とそれを基に作成した VM の間でページを共有できる．これらの技術はすべて，共有しているページが書き換えられると共有をやめる．一方，VMBeam のゲスト間メモリ共有では，ページに変更がなされてもページの共有を続ける．

VM マイグレーションを行わずに仮想化システムを高速にソフトウェア若化する研究も行われてきた．Warm-VM Reboot [2], [17] は VM を再起動せずにハイパーバイザと管理 VM だけを高速にソフトウェア若化する．ソフトウェア若化の際に VM のメモリイメージをメインメモリ上に保持しておき，ソフトウェア若化後にそのメモリイメージを再利用して高速に VM を復元する．しかし，ソフトウェア若化中は VM が停止してしまいダウンタイムが発生する．ReHype [18] は管理 VM を再起動せずにハイパーバイザのみをソフトウェア若化することができる．しかし，ハイパーバイザの状態の多くを引き継ぐため，ソフトウェア若化できる箇所は限定される．

7. まとめ

本稿では，仮想化システムの軽量なソフトウェア若化を実現する VMBeam を提案した．VMBeam では，仮想化システムのソフトウェア若化時には，ネストした仮想化を用いて同一ホスト上で別の仮想化システムを起動する．その仮想化システム上にゼロコピー・マイグレーションを用いて高速に VM をマイグレーションする．ゼロコピー・マイグレーションは移送元の VM のメモリを移送先に再配置することができる．Xen を用いて VMBeam の実装を行い，VM のマイグレーションを従来システムよりも高速かつ低負荷で行えることを示した．

今後の課題は，ソフトウェア若化時以外におけるネストした仮想化のオーバヘッド削減のため，ホスト・ハイパーバイザにおける脱仮想化 [7] を可能にすることである．また，ホスト・ハイパーバイザとゲスト・ハイパーバイザにおけるソフトウェア・エージングの違いを調査することも今後の課題である．

参考文献

[1] Huang, Y., Kintala, C., Kolettis, N. and Fulton, N. D.: Software Rejuvenation: Analysis, Module and Applications, *Proc. Intl. Symp. Fault-Tolerant Computing*, pp. 381–390 (1995).
[2] Kourai, K. and Chiba, S.: Fast Software Rejuvenation of Virtual Machine Monitors, *IEEE Trans. Dependable and Secure Comput.*, pp. 839–851 (2011).
[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. Symp. Operating Systems Principles*, pp. 164–177 (2003).
[4] Machida, F., Xiang, J., Tadano, K. and Maeno, Y.: Combined Server Rejuvenation in a Virtualized Data

Center, *Proc. Intl. Conf. Autonomic & Trusted Computing*, pp. 486–493 (2012).

[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. Symp. Networked Systems Design & Implementation*, pp. 273–286 (2005).

[6] Ben-Yehuda, M., Day, M. D., Dubitzky, Z., Factor, M., Har'El, N., Gordon, A., Liguori, A., Wasserman, O. and Yassour, B.-A.: The Turtles Project: Design and Implementation of Nested Virtualization., *Proc. Conf. Operating Systems Disign & Implementation*, Vol. 10, pp. 423–436 (2010).

[7] Lowell, D. E., Saito, Y. and Samberg, E. J.: Devirtualizable Virtual Machines Enabling General, Single-node, Online Maintenance, *Proc. Intl. Conf. Architectural Support for Programming Languages and Operating Systems*, pp. 211–223 (2004).

[8] Williams, D., Jamjoom, H. and Weatherspoon, H.: The Xen-Blanket: Virtualize Once, Run Everywhere, *Proc. European Conf. Computer Systems*, pp. 113–126 (2012).

[9] Huang, W., Gao, Q., Liu, J. and Panda, D. K.: High Performance Virtual Machine Migration with RDMA Over Modern Interconnects, *Proc. Intl. Conf. Cluster Computing*, pp. 11–20 (2007).

[10] Hines, M. R. and Gopalan, K.: Post-copy Based Live Virtual Machine Migration Using Adaptive Pre-paging and Dynamic Self-ballooning, *Proc. Intl. Conf. Virtual Execution Environments*, pp. 51–60 (2009).

[11] Svärd, P., Hudzia, B., Tordsson, J. and Elmroth, E.: Evaluation of Delta Compression Techniques for Efficient Live Migration of Large Virtual Machines, *Proc. Intl. Conf. Virtual Execution Environments*, pp. 111–120 (2011).

[12] Song, X., Shi, J., Liu, R., Yang, J. and Chen, H.: Parallelizing Live Migration of Virtual Machines, *Proc. Intl. Conf. Virtual Execution Environments*, pp. 85–96 (2013).

[13] Waldspurger, C. A.: Memory Resource Management in VMware ESX Server, *Proc. Symp. Operating Systems Design & Implementation*, pp. 181–194 (2002).

[14] Murray, D. G., Steven, H. and Fetterman, M. A.: Satori: Enlightened Page Sharing, *Proc. USENIX Annual Technical Conf.* (2009).

[15] 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. Symp. Operating Systems Design & Implementation*, pp. 85–93 (2010).

[16] Vrable, M., Ma, J., Chen, J., Moore, D., Vandekieft, E., Snoeren, A. C., Voelker, G. M. and Savage, S.: Scalability, Fidelity, and Containment in the Potemkin Virtual Honeyfarm, *Proc. Symp. Operating Systems Principles*, pp. 148–162 (2005).

[17] Kourai, K. and Chiba, S.: A Fast Rejuvenation Technique for Server Consolidation with Virtual Machines, *Proc. Intl. Conf. Dependable Systems and Networks*, pp. 245–255 (2007).

[18] Le, M. and Tamir, Y.: ReHype: Enabling VM Survival Across Hypervisor Failures, *Proc. Intl. Conf. Virtual Execution Environments*, pp. 63–74 (2011).