仮想化システムのソフトウェア若化のための軽量なVM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典型的な例はシステムの再起動であるが，その際の仮想マシン（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，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，ネストした仮想化を用いて，仮想化システムのソフトウェア若化のための軽量な VM マイグレーションを実現する *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，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，VM のメモリイメージを仮想ネットワーク経由で転送する代わりに，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メモリコピーを行う．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．

Lightweight VM Migration for Software Rejuvenation of Virtualized Systems

Hiroki Ooba[1]　Kenichi Kourai[1]

Abstract:
Virtualized Systems tend to suffer from software aging, which is the phenomenon that the state of running software degrades with time. It can be counteracted by a technique called software rejuvenation, whose typical example is a system reboot. To reduce the downtime of virtual machines (VMs) during the reboot, VMs have to be migrated in advance. However, VM migration stresses the system and affects the performance of the VMs running on top of it. In this paper, we propose *VMBeam*, which enables lightweight VM migration with nested virtualization for software rejuvenation of virtualized systems. When rejuvenating a virtualized system, VMBeam runs another virtualized system at the same host and migrates the VMs to it. At this time, it directly copies VM memory images between the two virtualized systems located in the same host, instead of network transfers. We have implemented VMBeam in Xen and confirmed the reduction of migration time and system loads.

1. はじめに

クラウドコンピューティングをはじめとして，様々なシステムで仮想化が行われるようになってきている．このような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

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

ⓒ 2014 Information Processing Society of Japan

トウェアの状態が次第に劣化していく現象であり，メモリリークなどが原因として挙げられる．ソフトウェア・エージングが起こるとシステムの性能が次第に低下していき，最悪の場合には想定外のシステムダウンが発生する可能性もある．

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，ソフトウェア若化と呼ばれる手法が提案されている [1]．ソフトウェア若化はシステムの状態をソフトウェア・エージングが起こる前の状態に戻すための手法である．ソフトウェア若化を行うことでシステムの性能を回復することができ，システムダウンを防ぐことができる．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．多くの VM の停止や起動には時間がかかるため，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．一方で，システム全体への影響を考えてマイグレーション速度を抑えた場合には，ライブマイグレーションにおけるトータルのメモリ転送量が増大する上，ソフトウェア若化の完了までに時間がかかってしまう．

本稿では，ネストした仮想化（Nested Virtualization）を用いて，仮想化システムのソフトウェア若化のための軽量な VM マイグレーションを実現する *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，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 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**ゲスト *VM* 間メモリコピー**を行うことにより，VM のメモリイメージの転送を高速化する．この手法では VM を動かしたままメモリイメージを転送することが可能であるため，従来と同様にライブマイグレーションを行うことができる．

我々は VMBeam を Xen 4.2.2 [6] に実装した．VMBeam では VM をマイグレーションする際に，二つの仮想化システムの下で動作するハイパーバイザにメモリのページフレーム番号だけを渡す．そして，ハイパーバイザ内で移動元の VM のメモリの内容を移動先に作成した VM のメモリに高速にコピーする．4GB のメモリを割り当てた VM をマイグレーションする際に，VMBeam では従来の物理マシン間のマイグレーションを行うより 4.4 倍高速であった．また，CPU 負荷，メモリ負荷，ネットワーク負荷をそれぞれ 43%，33%，0.5%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．仮想化システムは一般に，ハイパーバイザや VM を管理するための管理 VM などで構成される．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 [2]．

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．その最も単純な方法として，システムを再起動することでソフトウェア若化を行うことができる．仮想化システムの場合，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．VM 上で動作している OS の再起動には時間がかかる上，近年の計算機の高性能化により多くの VM が集約されて動作していることが多い．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のダウンタイムが長くなる傾向にある．

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 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．仮想化システムのソフトウェア若化を行う前に，**図 1** のように，仮想化システム上で動作しているすべての VM をマイグレーションにより別のホストに移動する．その際にライブマイグレーションを行うことにより VM のダウンタイムを非常に短く抑えることができる．その後で仮想化システムを再起動することで，VM にほとんど影響を与えることなくソフトウェア若化を行うことができる．

しかし，VM の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．VM のマイグレーションを行うには移動元のホストから移動先のホストに VM のメモリ

ⓒ 2014 Information Processing Society of Japan

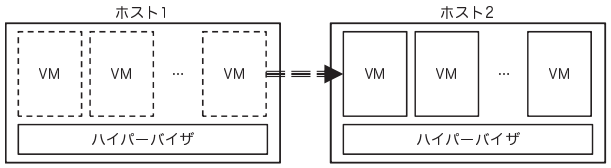

図 1 マイグレーションを活用したソフトウェア若化

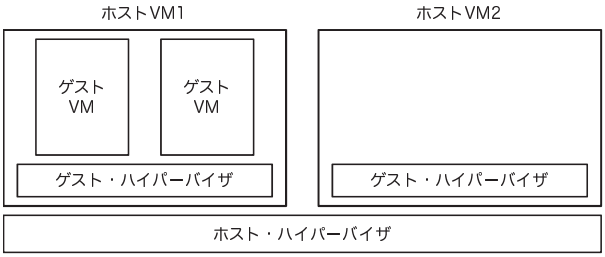

図 2 ソフトウェア若化時の VMBeam のシステム構成

イメージを転送する必要がある．このメモリデータの転送は，すべての VM を移動させる場合には合計で数 GB から数十 GB ものデータ量になることもある．その際に，メモリイメージの盗聴や改ざんを防ぐためには，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，移動元と移動先のホストの 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意していない場合は，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，その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，多くの 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 [9]．

このような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 [3]．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トータルの負荷が増大する可能性がある．これは，VM が動作している間に変更されたメモリを差分として転送する必要があるためである．マイグレーション時間が長くなるとそれだけ差分も増加する．また，マイグレーションに時間がかかるようになることにより，ハイパーバイザの再起動が可能になるまでの時間が長くなり，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，仮想化システムのソフトウェア若化のための軽量な VM マイグレーションを実現する *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，同一ホスト上で別の仮想化システムを起動し，その仮想化システム上に VM をマイグレーションする．そして，元の仮想化システムを終了させることでソフトウェア若化と同じ効果を得る．この際に，VM のメモリイメージを仮想ネットワーク経由で転送する代わりに，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高速に転送する．この手法により，マイグレーションの高速化および負荷の軽減を図る．

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 2** 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．

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 [4] [8]．これらの手法を用いることで，性能低下を 6〜20%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 [5] 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．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．既存の脱仮想化は通常の仮想化を対象としているが，この技術を応用してホスト・ハイパーバイザによる仮想化を行わないようにすることで，ネストした仮想化を用いない場合とほぼ同等の性能を得ることができると考えられる．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．

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．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 [2] 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しないため，比較的，ソフトウェア・エージングが起こりにくいと考えられる．また，通常時には脱仮想化を行うことで，その間，ホスト・ハイパーバイザはほとんど何も処理を行わないため，ソフトウェア・エージングはより起こりにくくなる．さらに，ホスト・ハイパーバイザにゲスト・ハイパーバイザのよう

ⓒ 2014 Information Processing Society of Japan

な豊富な機能を持たせず，必要最低限の機能だけを持たせるようにすることで，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，ホスト・ハイパーバイザのソフトウェア若化の頻度は相対的に低く抑えることができると考えられる．その結果，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となる．もし，ホスト・ハイパーバイザのソフトウェア若化を行う場合には，ホスト VM を別のホストにマイグレーションしてから行う．

3.2 軽量な VM マイグレーション

ゲスト VM のマイグレーションを行う際には，二つのホスト VM 内で動作している仮想化システム間でメモリイメージを転送する必要がある．移動元の仮想化システムはマイグレーションするゲスト VM のメモリイメージを仮想ネットワークを介して移動先の仮想化システムに転送する．そして，移動先では受け取ったメモリイメージを空のゲスト VM に書き込むことで VM のマイグレーションを実現する．VMBeam では，ゲスト VM の移動元と移動先が同一ホスト上にあるため，仮想ネットワークを用いたデータ転送を物理ネットワークより高速に行える可能性がある．しかし，仮想化が二重に行われるため，ネットワーク仮想化のオーバヘッドも大きくなる．

VMBeam は，同一ホスト上の仮想化システム間での軽量なデータ転送手法として，**ゲスト *VM* 間メモリコピー**を提供する．この手法では，移動元のゲスト VM のメモリの内容を移動先のゲスト VM のメモリに直接コピーすることができる．ゲスト VM 間メモリコピーによるデータ転送を**図 3** に示す．ゲスト VM のマイグレーション時には，ゲスト・ハイパーバイザ経由でホスト・ハイパーバイザを呼び出し，ホスト・ハイパーバイザがゲスト VM のメモリ全体をコピーする．このように，ゲスト VM 間メモリコピーを用いることでネットワーク仮想化のオーバヘッドを回避することができる．また，メモリイメージが盗聴される可能性のある仮想ネットワーク上で転送されないため，データの暗号化も不要になる．それに加えて，ゲスト VM 間メモリコピーではゲスト VM を動かしたままメモリデータをコピーすることができるため，従来と同様にライブマイグレーションを行うことができる．

4. 実装

我々は VMBeam を Xen 4.2.2 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．

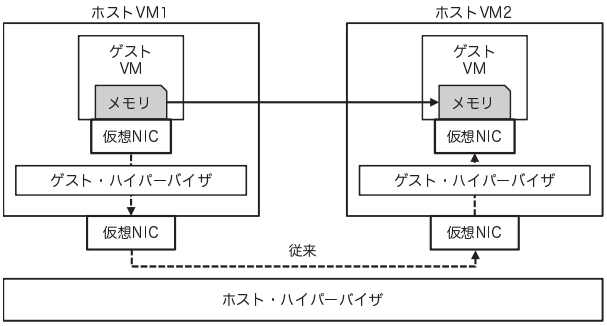

図 3 VM 間メモリコピーによるデータ転送

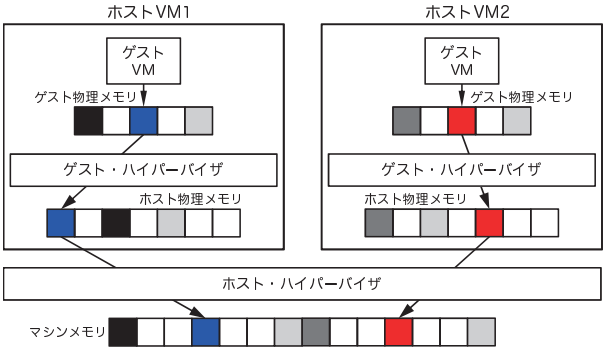

図 4 Xen のネストした仮想化におけるメモリモデル

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 4** 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．それぞれのメモリにはページフレーム番号が順番に割り振られ管理される．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）が割り振られる．MFN と HPFN，HPFN と GPFN の間の対応はそれぞれホスト・ハイパーバイザ，ゲスト・ハイパーバイザで管理される．

4.2 ゲスト VM 間メモリコピー

VMBeam はゲスト管理 VM に対して，異なるホスト VM 上のゲスト VM との間でメモリをコピーする機能を提供する．**図 5** のように，コピー元のゲスト VM を管理するゲスト管理 VM は，ゲスト VM の ID とコピーするメモリのページフレーム番号（GPFN）の配列を指定してゲスト・

ⓒ 2014 Information Processing Society of Japan

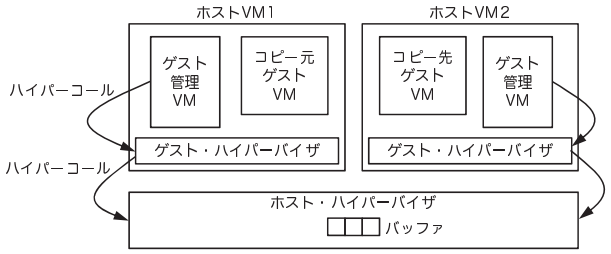

図 5 ゲスト VM 間メモリコピー

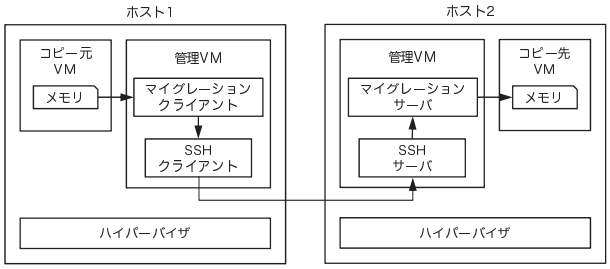

図 6 物理マシン間におけるメモリ転送の流れ

ハイパーバイザを呼び出す．一方，コピー先のゲスト VM を管理するゲスト管理 VM も，ゲスト VM の ID とコピー先のメモリページに対応する GPFN の配列を指定してゲスト・ハイパーバイザを呼び出す．ゲスト・ハイパーバイザを呼び出す際にはハイパーコールが用いられる．二つのゲスト管理 VM は異なるホスト VM 上で動作しているため，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．GPFN と HPFN の対応はゲスト・ハイパーバイザが管理しているため，この変換は容易に行うことができる．ゲスト・ハイパーバイザからホスト・ハイパーバイザを呼び出す際には，ゲスト OS からハイパーバイザを呼び出す場合と同様にハイパーコール・ページを用いたハイパーコール呼び出しを行う．ゲスト・ハイパーバイザはハイパーコール・ページを起動時に初期化し，ホスト・ハイパーバイザから取得したハイパーコール呼び出し命令列を格納しておく．

ホスト・ハイパーバイザは，ホスト VM から渡された HPFN の配列を MFN の配列に変換し，対応するコピー元のメモリの内容をコピー先のメモリにコピーする．HPFN と MFN の対応はホスト・ハイパーバイザが管理しているため，この変換は容易に行うことができる．コピー元とコピー先からのハイパーコールを非同期に実行できるようにするために，ホスト・ハイパーバイザはコピー元から渡された HPFN を格納するためのバッファを用意している．コピー元から呼び出された時に，ホスト・ハイパーバイザは渡された HPFN をバッファに格納する．一方，コピー先から呼び出された時にバッファが空でなければ，その中の HPFN と渡された HPFN に対応するメモリ間でコピーを行う．コピーが完了した HPFN はバッファから削除する．呼び出し時にバッファが空であればエラーを返し，コピー先のゲスト管理 VM に再試行させる．

4.3 コピー・マイグレーション

VMBeam が提供するコピー・マイグレーションは，ゲスト VM のメモリイメージをゲスト VM 間メモリコピーを用いて転送する．従来のマイグレーションでは，**図 6** のように，移動元のゲスト管理 VM はゲスト VM の GPFN に対応するメモリページを順にマップし，メモリの内容を仮想ネットワーク上に構築された SSH トンネルを用いて転送していた [*1]．コピー・マイグレーションでは，ハイパーコールを用いて 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．メモリページをマップする必要はなく，SSH による暗号化も行われない．

移動先のゲスト管理 VM でもハイパーコールを用いて，新しく作成したゲスト VM の 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ホスト・ハイパーバイザ内で行われ，移動元のゲスト VM のメモリの内容が移動先のゲスト VM のメモリにコピーされる．従来のマイグレーションでは，ゲスト管理 VM がゲスト VM のメモリをマップし，SSH トンネル経由で受信したメモリの内容を書き込んでいた．コピー・マイグレーションでは，メモリページをマップする必要がなく，SSH による復号化も行われない。

このように，コピー・マイグレーションでは移動元で送信処理を行った時点ではゲスト VM のメモリがコピーされないため，メモリイメージの一貫性に問題が生じる可能性がある．実際のメモリのコピーは移動先がハイパーコールを呼び出した時点で行われるためである．ライブマイグレーションではゲスト VM のメモリは随時書き換えられるため，送信処理を行った時と実際にコピーが行われる時とでゲスト VM のメモリの内容が同じとは限らない．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の内容は再度送られるため，このことは問題にはならない．送信処理を行った後で更新されたメモリの内容をコピーしてしまったとしても，次の送信処理の際に上書きされるだけである．最終的にはゲスト VM を停止させて送信処理が行われるため，マイグレーション完了時には移動元と移動先のゲスト VM のメモリの内容は一致する．

5. 実験

我々は VMBeam の有効性を示すために，マイグレーション性能を調べる実験を行った．

[*1] xl コマンドの場合．xm コマンドでは SSL を用いて転送．

ⓒ 2014 Information Processing Society of Japan

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．

既存システムとの比較を行うために，(1) VMBeam を用いたホスト VM 間のマイグレーション，(2) Xen 標準のネストした仮想化（標準ネスト）を用いたマイグレーション，(3) Xen-Blanket [7] を用いたマイグレーション，(4) 物理マシン間のマイグレーションについて実験を行った．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使うことで，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．**図 7** に Xen-Blanket のシステム構成を示す．

(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．ホスト VM およびゲスト VM では HVM ゲストを動作させた．オリジナルの Xen-Blanket はゲスト VM では準仮想化（PV）ゲストを動作させているが，今回はマイグレーション処理に違いが出ないように HVM ゲストを用いた．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 個ずつ割り当て，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 および残りの 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．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．この際に，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．5.5 節の実験を除いて，ゲスト VM ではアプリケーションを動かさず，アイドル状態とした．

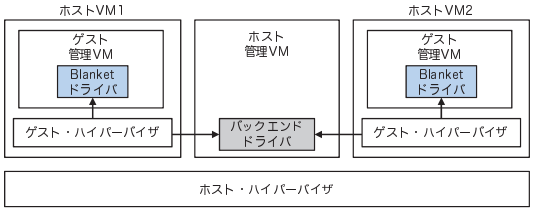

図 7 Xen-Blanket におけるゲスト管理 VM 間の通信

5.2 マイグレーションの動作テスト

VMBeam によるマイグレーションの動作確認を行った．VMBeam によってゲスト VM を移動させた後もゲスト VM への仮想コンソール接続を行うことができ，ゲスト VM が移動先のホスト VM 上で正常に動作を続けていることが確認できた．また，ライブマイグレーションでは SSH 接続が途切れることなくゲスト VM の移動を行えることを確認した．

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 4096MB 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 コマンドでマイグレーションを完了するまでにかかる時間をそれぞれ 3 回ずつ計測した平均値を**図 8** に示す．この結果より，どのシステムでもマイグレーションにかかる時間はゲスト VM のメモリサイズに比例することが分かる．VMBeam を用いると，4GB のメモリを割り当てたゲスト VM のマイグレーションを 22 秒で完了させることができた．**図 9** にマイグレーション時間のうち，ゲスト VM 間メモリコピーにかかった時間を示す．この結果より，メモリコピー時間はゲスト VM のメモリサイズにほぼ比例することが分かる．

VMBeam におけるマイグレーションを物理マシン間のマイグレーションと比較すると，1.0〜4.4 倍高速に行えることが分かった．一方，標準ネストとの比較では，VMBeam におけるマイグレーションのほうが 2.1〜13.3 倍高速であった．また，Xen-Blanket との比較では，VMBeam によるマイグレーションのほうが 1.1〜5.7 倍高速であった．このような結果になる原因を調べるために，iperf を用いて SSH トンネルを用いる場合と用いない場合のデータ転送性能を計測した．VMBeam についてはゲスト VM 間メモリコピー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．

図 10，**図 11** の結果より，Xen-Blanket のネットワーク性能は十分に高いため，マイグレーションに SSH トンネルを用いることによる暗号化のオーバヘッドが原因だと考えられる．それに対して，標準ネストでマイグレーションに時間がかかるのは，このオーバヘッドに加えて，仮想ネットワークのオーバヘッドが大きいせいだと考えられる．一方，SSH トンネルを用いた場合のデータ転送性能の差よ

ⓒ 2014 Information Processing Society of Japan

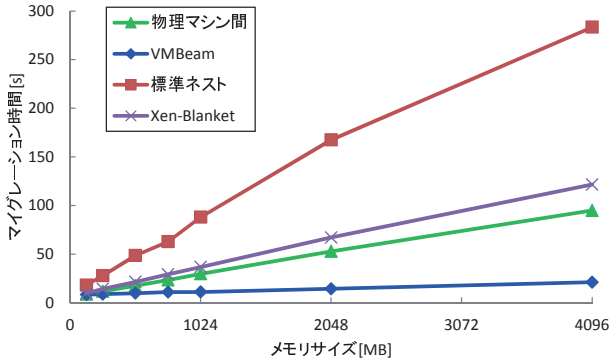

図 8　マイグレーション時間

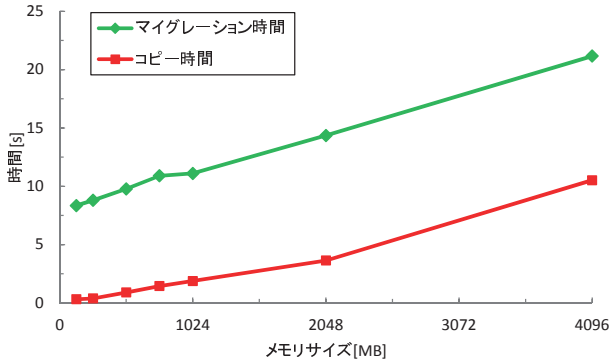

図 9　VMBeam におけるマイグレーション時間の内訳

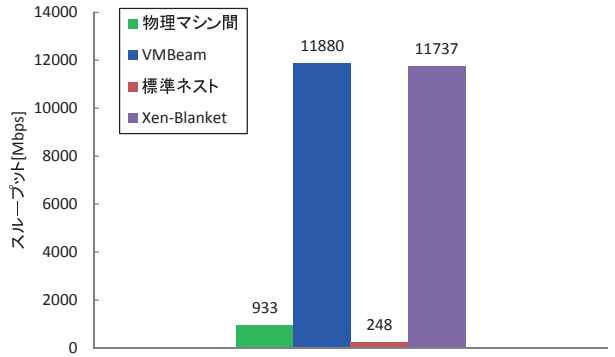

図 10　データ転送性能

りもマイグレーション時間の差が小さいことから，帯域を十分に活かせていない可能性があることが分かる．特に，VMBeam の場合には一括してゲスト VM 間メモリコピーを行うことで，マイグレーションをさらに高速化できる可能性がある．

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 128MB から 4096MB まで変えて，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．ゲスト VM が停止している時間を 3 回ずつ計測した平均値を**図 12** に示す．この結果より，マイグレーション中のゲスト VM の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，物理マシン間のマイグレーションのダウンタイムより 0.2 秒程度長い．これは VMBeam がマイグレーションの最終段階で標準ネストと同じ低速な仮想ネットワークを使って CPU 状態を送っているせいだと考えられる．一方，VMBeam におけるダウンタイムは標準ネストより 16〜30%短く，Xen-Blanket より 40〜57%短かった．

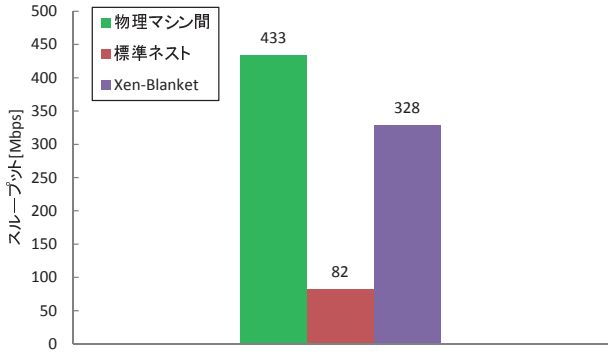

図 11　SSH トンネルを用いた場合のデータ転送性能

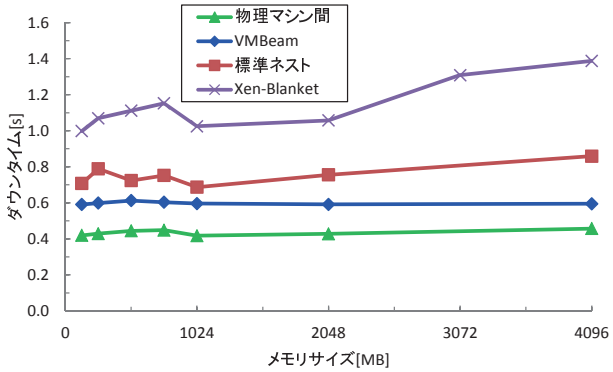

図 12　マイグレーション中のダウンタイム

5.5　ゲスト VM 内のメモリ書き込みの影響

ゲスト VM 内のメモリ書き込みがマイグレーション性能に及ぼす影響を調べるために，ゲスト VM 内で毎秒 5000 ページのメモリ書き込みを行うプログラムを動作させながらライブマイグレーションを行った．このメモリ書き込みプログラムはシステムへの負荷を最小に抑えるために，各ページに対して 1 バイトだけ書き込み，ページをダーティにする．ゲスト VM に 1GB のメモリを割り当てた時のマイグレーション時間およびダウンタイムを**図 13**，**14** に示す．

この結果より，VMBeam ではマイグレーション時間，ダウンタイムともにゲスト VM 内のメモリ書き込みの影響を受けていないことが分かる．一方，物理マシン間のマイグレーションではマイグレーション時間が 12.6 秒増加したが，ダウンタイムへの影響は見られなかった．Xen-Blanket においてはマイグレーション時間が 15.5 秒増加し，ダウンタイムも 0.26 秒増加した．しかし，標準ネストではマイグレーション時間が 382 秒，ダウンタイムは 27.9 秒もの増

ⓒ 2014 Information Processing Society of Japan

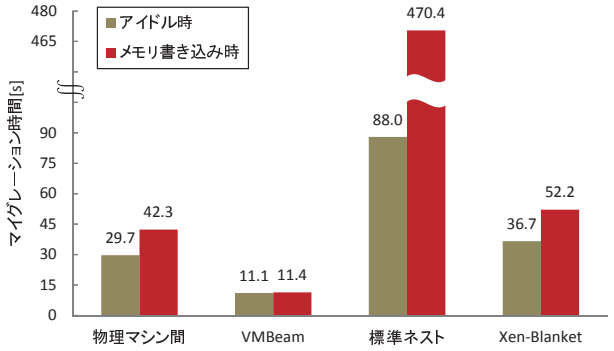

図 13 メモリ書き込みを行うゲスト VM のマイグレーション時間

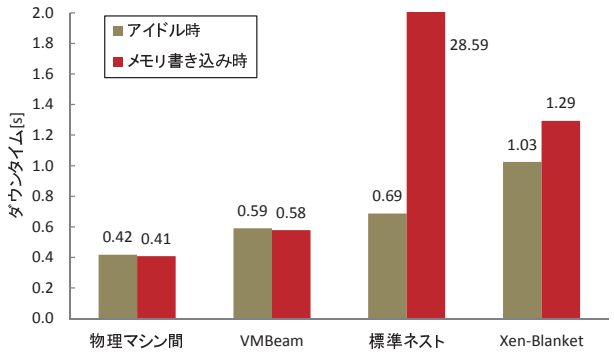

図 14 メモリ書き込みを行うゲスト VM のダウンタイム

加が見られた．マイグレーション時間，ダウンタイムともにダーティページ数に比例して時間が増加するため，マイグレーション性能が低いほど大きな影響を受けていることが分かる．ダウンタイムがアイドル時より若干短くなっている場合もあるが，誤差の範囲と考えられる．

5.6 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．ネストした仮想化を用いる場合にはホスト管理 VM で CPU 負荷の計測を行い，ホスト管理 VM とホスト VM の CPU 使用率を合計した．物理マシン間のマイグレーションの場合には管理 VM で計測を行い，管理 VM とゲスト VM の CPU 使用率を合計した．マイグレーション中の CPU 使用率の変化を**図 15** 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 16** に示す．物理マシン間でのマイグレーションに関しては VM の転送元と転送先の両方で計測を行った．

VMBeam における最大の CPU 使用率は標準ネストや Xen-Blanket と同程度であった．一方，物理マシン間のマイグレーションと比較すると最大で 2 倍の負荷がかかっていることが分かる．これは VMBeam では転送元と転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 負荷は物理マシン間のマイグレーションと比較して 41〜43%の負荷に抑えられた．また，標準ネストと比べると

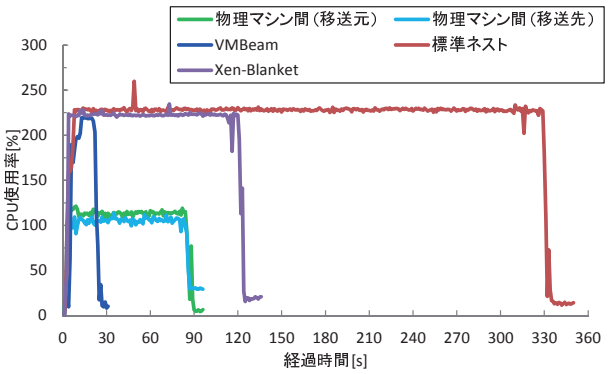

図 15 マイグレーション中の CPU 負荷

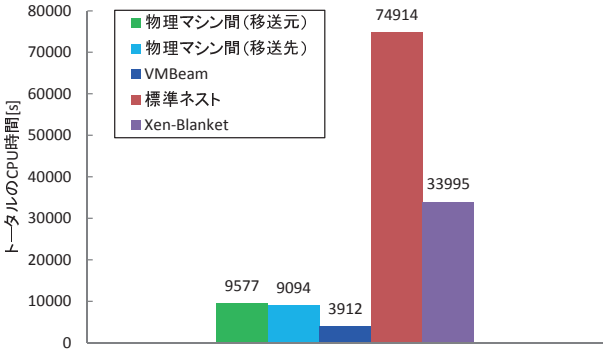

図 16 マイグレーション中のトータルの CPU 負荷

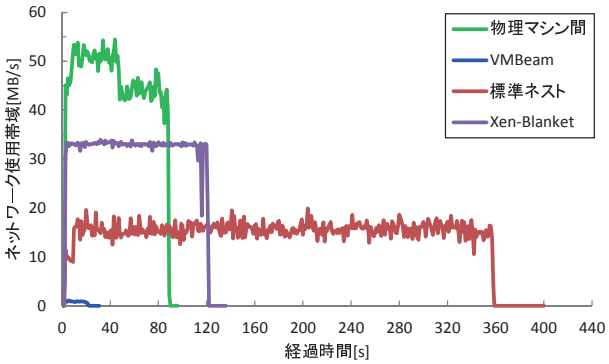

図 17 マイグレーション中のネットワーク負荷

5%，Xen-Blanket と比べると 12%の負荷であった．

5.7 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，移動元から移動先へネットワークを介して転送されるデータ量を計測した．消費されたネットワーク帯域の変化を**図 17** に示す．また，マイグレーション中のトータルのデータ転送量を**図 18** に示す．VMBeam ではメモリイメージの転送にネットワークを用いないため，データ転送量を 0.5%以下に削減できていることが分かった．標準ネストのデータ転送量が多くなっているのは，マイグレーションに時間がかかる分だけ再送されるダーティページが増えてしまうためである．

ⓒ 2014 Information Processing Society of Japan

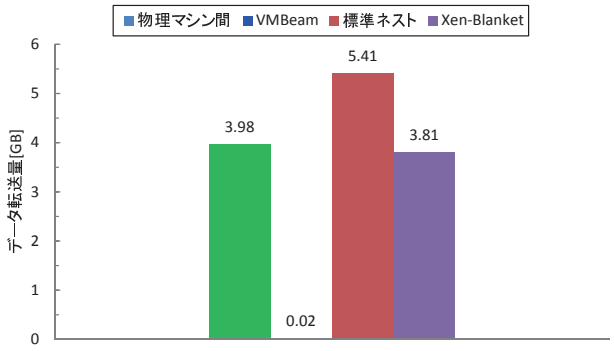

図 18 マイグレーション中のネットワーク転送量

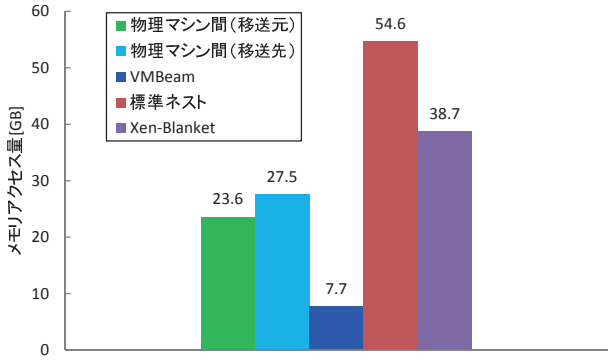

図 19 マイグレーション中のメモリアクセス量の概数

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 19** に示す．転送されたメモリイメージのサイズはどのシステム構成でも 4GB 弱であった．VMBeam ではホスト・ハイパーバイザが移動元のゲスト VM のメモリを読み出し，移動先のゲスト VM のメモリに書き込むため，ゲスト VM のメモリサイズの 2 倍のメモリアクセスが行われると考えられる．

物理マシン間のマイグレーションについては，移動元の管理 VM のカーネルが VM のメモリを読み出してソケットバッファに書き込み，それを 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別のバッファに書き込み，それを SSH サーバに送信する．その際に，カーネルが送信データをソケットバッファに書き込み，NIC がそのデータを DMA でメモリから読み出す．DMA は CPU キャッシュからではなくメモリからの読み出しを伴う．一方，移動先の管理 VM では，NIC が受信したデータを DMA でソケットバッファに書き込み，カーネルがそのデータをメモリから読み出して 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別のバッファに書き込み，それをマイグレーション・サーバに送信する．その際に，カーネルが送信データをソケットバッファに書き込み，それをマイグレーション・サーバのバッファに書き込む．マイグレーション・サーバはそのデータを VM のメモリに書き込む．そのため，移動元ではゲスト VM のメモリサイズの 6 倍，移動先では 7 倍のメモリアクセスが行われると考えられる．これより，VMBeam ではメモリ負荷を 28〜33%程度に削減できると期待できる．

標準ネストの場合には，移動元のゲスト管理 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バッファに読み込む．そのデータは TAP デバイス経由で移動先のホスト VM に仮想 NIC を提供している QEMU に送られる．その際に，カーネルがデータをソケットバッファに書き込み，それを QEMU のバッファに書き込む．QEMU は NIC の DMA をエミュレートして移動先のゲスト管理 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，移動元のゲスト管理 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 のカーネルだけを経由して，移動先のゲスト管理 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 ネストした仮想化のオーバヘッド

ネストした仮想化のオーバヘッド削減は本稿のスコープ外であるが，参考のために，ネストした仮想化を用いることでどの程度の性能低下が生じるのかを調べた．Xen 4.2.2, Xen 4.4.0, Xen-Blanket 4.1.1 上のゲスト VM と Xen 4.2.2 のホスト VM 上で UnixBench を実行した．ホスト VM 上での UnixBench のスコアを 1 とした相対スコアを**図 20** に示す．この結果より，ゲスト VM においても演算処理，ファイルのコピー，システムコールのオーバヘッドに関しての性能低下は小さいが，execl 実行，プロセス生成，シェルスクリプト処理に関しては非常に性能が低下してしまうことが分かった．また，Xen 4.4.0 は一部の項目を除いて Xen 4.2.2 よりもゲスト VM の性能が向上していることが分かった．

6. 関連研究

Microvisor [5] は，システムのメンテナンスを別の仮想マシン上で行い，メンテナンス後に仮想マシン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ための技術である脱仮想化に焦点を当てているのに対して，VMBeam ではゲスト VM のマイグレーションの高速化に焦点を当てている．VMBeam でも Microvisor の脱仮想化技術を用いることで，通常時のオー

ⓒ 2014 Information Processing Society of Japan

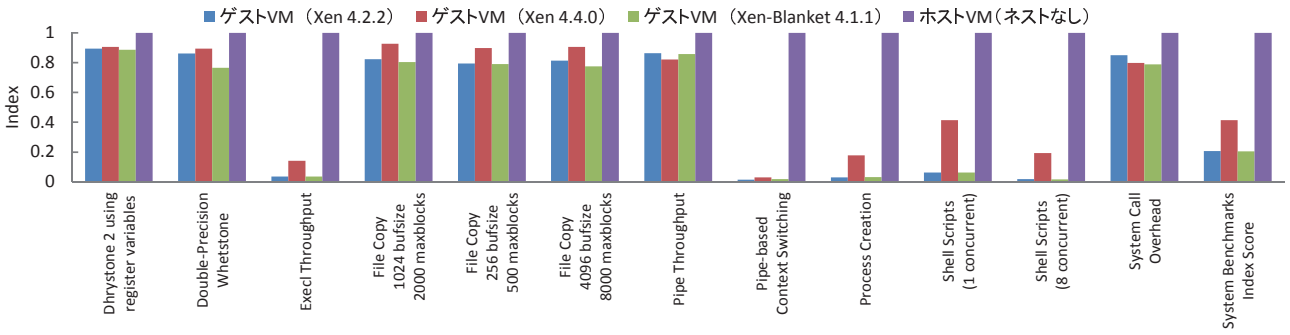

図 20 UnixBench によるスコア比較

バヘッドを削減することができる．

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．PMigrate [11] 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，メモリ更新頻度の高い VM 以外ではマイグレーションに使われるトータル CPU 時間は削減されない．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．InfiniBand の RDMA を用いたマイグレーション [12] では，マイグレーション時間，ダウンタイム，CPU 負荷を削減することができている．しかし，高価な機器が必要となる．

同一マシン上の VM 間の通信を高速化する研究も盛んに行われてきた。XenSocket [13] は VM 間の共有メモリを用いて高速な一方向通信を提供する。IVC [14] は VM 間通信を高速化する MPI ライブラリを提供し、MPI アプリケーションの互換性を保つ。XenLoop [15] や XWAY [16] はソケットを用いたプログラムのバイナリ互換性を保って VM 間の通信を高速化する。VMBeam はこれらとは異なり、ネストした仮想化におけるゲスト管理 VM 間の通信を高速化する。さらに、通信の際に共有メモリを介さず、VM 間でデータを直接コピーする。

ネストした仮想化のオーバヘッドを削減する研究もいくつか行われている．Turtles Project [4] は KVM ベースのホスト・ハイパーバイザを提供し，その上で既存の仮想化システムを動作させることができる．ホスト・ハイパーバイザにはネストした仮想化を高速化する機構が実装されており，通常の仮想化システムの 6〜8%のオーバヘッドに抑えられている．一方，Xen-Blanket [7] はネストした仮想化の特別なサポートを行っていない既存の仮想化システムの上でユーザ独自の仮想化システムを動作させることができる．Xen-Blanket ではゲスト管理 VM に Blanket ドライバと呼ばれる準仮想化ドライバを導入することで，VM をネストすることによる I/O オーバーヘッドを削減している．

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いる．Warm-VM Reboot [10] は VM を再起動せずにハイパーバイザだけを高速にソフトウェア若化するための手法である．ハイパーバイザを再起動する際に VM のメモリイメージをディスクなどの外部記憶ではなくメインメモリ上にそのまま保持することで高速に VM をサスペンドする．ハイパーバイザの再起動後にメインメモリ上に残されたメモリイメージを再利用することで高速に VM のレジュームを行うことができる．しかし，この手法ではハイパーバイザおよび管理 VM を再起動している間は VM が停止してしまいダウンタイムが発生する．

ReHype [17] はハイパーバイザの障害時にハイパーバイザのみを再起動することができるシステムである．この際に，通常の VM だけでなく，管理 VM の再起動を行う必要もない．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．また，文献 [18] のシステムでは，管理 VM のみを再起動することも可能である．ただし，管理 VM を再起動する際にはゲスト VM にダウンタイムが発生する可能性がある．

7. まとめ

本稿では，ネストした仮想化を用いて，仮想化システムのソフトウェア若化のための軽量な VM マイグレーションを実現する 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同一ホスト上で別の仮想化システムを起動し，その仮想化システム上に VM をマイグレーションする．このマイグレーションはゲスト VM 間メモリコピーを用いて高速化される．我々は 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，メモリページだけでなく CPU 状態も VM 間メモリコピーで転送し，ダウンタイムを削減することである．さらに，より詳細な性能評価を行うことを計画している．例えば，マイグレーションがゲスト VM 内のアプリケーションに及ぼす影響の違いや，ホスト・ハイパーバイザがゲスト・ハイパーバイザよりもエージングを起こしにくいかどうかなどの評価を行う必要がある．また，ゲスト VM 間でメモリを共有できるようにして，より軽量なマイグレーションを可能にすることを検討している．マイグレーション中に移動元と移動先でメモリを共有できれば，

ⓒ 2014 Information Processing Society of Japan

ライブマイグレーションにおける差分の転送が必要なくなる．

参考文献

[1] Huang, Y., Kintala, C., Kolettis, N., Fulton, N. D.: Software Rejuvenation: Analysis, Module and Applications. in *Proc. Intl. Symp. Fault-Tolerant Computing*, pp.381–391, 1995.

[2] Machida, F., Xiang, J., Tadano, K., Maeno, Y.: Combined Server Rejuvenation in a Virtualized Data Center, in *Proc. Intl. Conf. Autonomic and Trusted Computing*, pp.486–493, 2012.

[3] Clark, C., Fraser, K., Hand, S., Hansen, J. G., Jul, E., Limpach, C., Pratt, I., Warfield, A.:Live migration of virtual machines, in *Proc. Symp. Networked Systems Design and Implementation*, pp.273–286, 2005.

[4] 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. in *Proc. Conf. Operating Systems Design and Implementation*, pp.1–6, 2010.

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

[6] 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, in *Proc. Symp. Operating Systems Principles*, pp.164–177, 2003.

[7] Williams, D., Hani, J. and Hakim, W.: The Xen-Blanket: Virtualize Once, Run Everywhere, in *Proc. European Conf. Computer Systems*, pp.113–126, 2012.

[8] Nakajima, J.: Making Nested VIrtualization Real by Using Hardware Virtualization Features, in *LinuxCon Japan*, 2013.

[9] Kourai, K. and Chiba, S: Fast Software Rejuvenation of Virtual Machine Monitors, in *IEEE Trans. Dependable Secur. Comput.*, pp.839–851, 2011.

[10] Kourai, K. and Chiba, S: A Fast Rejuvenation Technique for Server Consolidation with Virtual Machines, in *Proc. Intl. Conf. Dependable Systems and Networks*, pp.245–254, 2007.

[11] Song, X. Shi, J. Liu, R. Yang, J. CHen, H.: Parallelizing Live Migration of Virtual Machines, in *Proc. Intl. Conf. Virtual Execution Environments*, pp.85–96, 2013.

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

[13] Zhang, X. McIntosh, S. Rohatgi, P. Griffin, J. L.: XenSocket: A High-Throughput Interdomain Transport for Virtual Machines, in *Proc. Intl. Conf. Middleware*, pp.184–203, 2007.

[14] Huang, W. Koop, M. J. Gao, Q. Panda, D. K.: Virtual Machine Aware Communication Libraries for High Performance Computing, in *Proc. Intl. Conf. Supercomputing*, pp.9:1–9:12, 2007.

[15] Wang, J. Wright, K.-L. Gopalan, K.: XenLoop: A Transparent High Performance Inter-VM Network Loopback, in *Proc. Intl. Symp. High Performance Distributed Computing*, pp.109–118, 2008.

[16] Kim, K. Kim, C. Jung, S.-I. Shin, H.-S. Kim, J.-S.: Inter-domain Socket Communications Supporting High Performance and Full Binary Compatibility on Xen, in *Proc. Intl. Conf. Virtual Execution Environments*, pp.11–20, 2008.

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

[18] Le, M. and Tamir, Y.: Applying Microreboot to System Software, in *Proc. Intl. Conf. Software Security and Reliability*, pp.11–20, 2012.

ⓒ 2014 Information Processing Society of Japan