Vol.2013-OS-126 No.15
2013/7/31

異種クラスタを跨がる仮想マシンマイグレーション機構

高野 了成[1,a)]　中田 秀基[1]　広渕 崇宏[1]　田中 良夫[1]　工藤 知宏[1]

概要：災害復旧 (DR) や事業継続計画 (BCP) の観点から、クラウド間を跨がった仮想計算機マイグレーションの重要性が高まっている。しかし、ヘテロなクラウド環境におけるマイグレーション技術に関して、今まで十分に議論されていない。本論文では、プライベートクラウドとパブリッククラウドなど、通信デバイスが異なるヘテロなクラウド環境間で仮想クラスタをマイグレーションする際の問題点を明らかにし、解決策を提案する。本提案の特長は、仮想クラスタを構成するノード VM に対して、実行環境の変化に応じて、最良の通信性能を達成する通信デバイスをアプリケーションから透過的に選択し、利用できることである。インターコネクト透過型マイグレーションである Ninja migration、OpenFlow を用いたエッジオーバレイネットワーク、ストレージマイグレーションなどの要素技術から成るプロトタイプシステムを実装した。このシステムを用いて、産総研の Infiniband クラスタと商用パブリッククラウド間で仮想クラスタをマイグレーションできることを確認した。

1. はじめに

クラウドの利用が拡大し、災害復旧 (Disaster Recovery) や事業継続計画 (Business Continuity Planning) の観点から、仮想計算機 (VM) マイグレーション技術の重要性が高まっている。DR や BCP は自然災害や障害に対応するため、本質的に計算やデータを遠隔地に分散して配置することが重要となる。しかし、広域環境におけるマイグレーション技術は数多くの先行研究 [1], [2] があるにも関わらず、いまだ実用化に向けた課題は多い。

本論文では、オンプレミスのプライベートクラウドとパブリッククラウドなど、ヘテロなクラウド環境間で仮想クラスタをマイグレーションする際の問題点とその解決方法について議論する。なお、仮想クラスタとは、クラスタ計算機として動作する VM の集合、およびそれらの VM を接続するネットワークを指す。また、仮想クラスタを構成する VM をノード VM と呼ぶ。ヘテロな環境における VM マイグレーション技術に関しては、いくつかの先行研究 [3] があるが、本論文では、特にノード VM 間の通信デバイスが異なる場合の対応に焦点を当てる。仮想クラスタを構成するノード VM に対して、クラウド間の差異を極力吸収しつつ、通信デバイスに関しては、実行環境の変化に応じて最良の通信性能を有するもの選択することを考える。これは CPU の仮想化対応により CPU やメモリの仮想化オーバヘッドはほぼ問題にはならなくなっているのに対して、I/O デバイスはいまだ仮想化オーバヘッドが無視できないからである [4]。さらに通信デバイスの切替は仮想クラスタで実行中のアプリケーションに対して透過的に実行することが求められる。例えば、プライベートクラウドで Infiniband を用いた HPC 向け仮想クラスタを、災害時に Ethernet しか持たないパブリッククラウドに移送して、実行を継続し、災害から復旧したら、プライベートクラウドに戻すといったユースケースを考える。

このような仮想クラスタマイグレーションを実現するための課題として、(1) 利用可能な通信デバイスに応じた最適な通信デバイスの切替え、(2) クラウド間のローカルエリアネットワーク環境の差異の隠蔽、(3) 広域環境における共有ストレージの利用、が挙げられる。我々は上記の各課題に対して、(1) 異なる通信デバイスを有するクラスタを跨いで仮想クラスタをマイグレーションする機構 Ninja migration [5] の適用、(2) OpenFlow 技術を用いたエッジオーバレイネットワークの構築、(3) 共有ストレージが不要なストレージマイグレーションの利用により解決を図る。

本論文の構成は以下のとおりである。2 節でパブリッククラウドとの仮想クラスタマイグレーションの問題を詳細化し、その解決策について述べる。3 節でそのプロトタイプシステムの実装について示し、その実現性を検証するための実験を 4 節で行う。5 節で関連研究について言及し、最後に 6 節でまとめを行う。

[1] 独立行政法人 産業技術総合研究所 情報技術研究部門
Information Technology Research Institute, National Institute of Advanced Industrial Science and Technology (AIST)
[a)] takano-ryousei@aist.go.jp

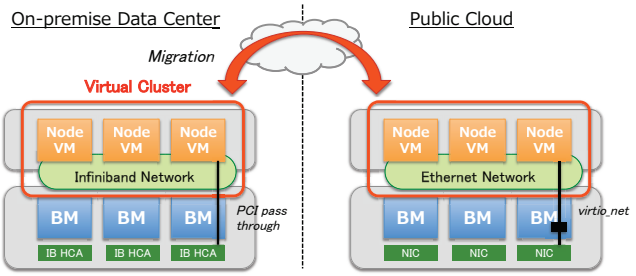

図 1　仮想クラスタマイグレーション

2. パブリッククラウドとの仮想クラスタマイグレーション

2.1 目的と問題

本研究の目的は、プライベートクラウドとパブリッククラウドなど、ヘテロなクラウド環境間で仮想クラスタをマイグレーションすることである。さらに、この際に仮想クラスタを構成するノード VM に対して、実行環境の変化に応じて、アプリケーションが最良の通信性能を達成する通信デバイスを透過的に選択し、利用できることを目指す。仮想クラスタ上で実行するアプリケーションとして、HPC 分野で標準的に使われる MPI (Message Passing Interface) プログラムを対象とする。また、仮想クラスタのネットワーク構成としては、virtio を利用した準仮想化ネットワークの他、仮想計算機モニタ (VMM) をバイパスして、VM から直接 Ethernet や Infiniband デバイスを用いる PCI パススルー方式も考慮する。

仮想クラスタマイグレーションの概要を **図 1** に示す。左がプライベートクラウドであり、右がパブリッククラウドである。プライベートクラウドでは、Infiniband を提供しており、各ノード VM は PCI パススルー経由で Infiniband HCA を利用できる。一方、パブリッククラウドでは準仮想化 Ethernet デバイスを提供している。ここで、プライベートクラウド上の仮想クラスタをパブリッククラウドへマイグレーションする場合、Infiniband を利用していたアプリケーションは、Ethernet へと通信路を切り替えて、動作を継続する。

上記の要求を実現するためには、次に示す問題がある。

(1) 仮想クラスタで稼働するアプリケーションは、利用可能な通信用ネットワークに応じて動的に通信デバイスを切り替える必要がある。
(2) クラウド間でクラスタ環境をシームレスに移送するために、両環境で同一のサブネットを構築し、各ノード VM の IP アドレスをマイグレーション前後で同じに保つ必要がある。
(3) 広域分散環境のような高遅延環境において、共有ストレージを用いることは性能上現実的ではない。

本節の残りでは、上記 3 つの問題の詳細化と我々のアプローチについて述べる。

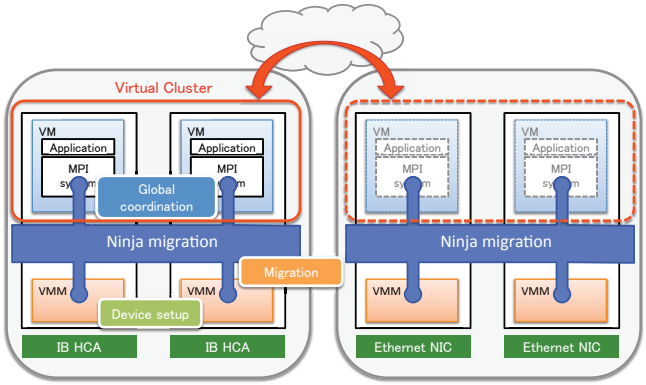

図 2　Ninja migration の概要

2.2 インターコネクト透過型マイグレーション

利用可能な通信用ネットワークに応じて動的に通信路を切り替えられるように、我々が提案しているインターコネクト透過型マイグレーション機構 Ninja migration [5] を利用する。Ninja migration の概要を**図 2** に示す。Ninja migration は、MPI システムと密接に連携して動作する。

PCI パススルー技術を用いて VM にデバイスと直接割り当てた場合、VMM はデバイスのレジスタに書き込まれたデータ（Infiniband であれば、Local ID や Queue Pair 番号など）を保存・復帰できないので、マイグレーションは動作しない。PCI Hotplug 機能を使えば、一時的に PCI パススルーデバイスを VM から取り外すことでマイグレーションは可能になるが、VMM が安全にデバイスを取り外すタイミングを知ることは困難である。メッセージ送信中にデバイスを取り外すと、メッセージが失われたり、アプリケーションが異常終了してしまう。そこで、MPI ランタイムと VMM が連携することで、全ノード VM で送信中のメッセージがない安全な状態にしてから、デバイスを取り外す。なお、VMM とゲスト OS 内の MPI ランタイムが協調する仕組みは、SymVirt (Symbiotic Virtualization) [6] を利用して実現している。

さらに Infiniband クラスタと Ethernet クラスタなど、計算ノード間のインターコネクトが異なるクラスタ間でのマイグレーションも実現する。MPI ランタイムでは複数の通信デバイスを利用するために通信層が抽象化されている。また、チェックポイント・リスタートに対応するため、全プロセスが送信途中のメッセージが存在しない状態を作り、かつリスタート時にコネクションを再確立する機能を有している。この機能を流用することで、チェックポイントの代わりにマイグレーションを行い、マイグレーション後に、MPI ランタイムが利用可能な通信デバイスを検出し、アプリケーションが利用するコネクションを張り直す。これはアプリケーションからは透過に実行されるので、マイグレーション中に通信デバイスが切り替わったとしても、アプリケーションの再実行は不要である。

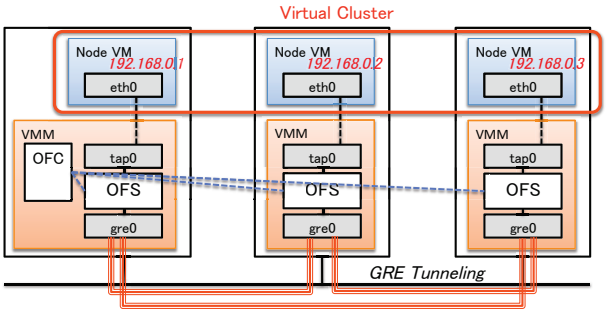

図 3 仮想クラスタとエッジオーバレイネットワーク

2.3 エッジオーバレイネットワーク

MPI は IP アドレスのリストを与えることで、通信相手を決定するが、アプリケーション実行中に IP アドレスが変わることは想定されていない。したがって、ノード VM では、マイグレーション前後で同一の IP アドレスを維持する必要がある。パブリッククラウドの中には VM あたり複数のネットワークインタフェースに対応するものもあるが、ここでは 1 つのネットワークインタフェース、1 つの IP アドレスが割り当てられていると仮定する。

プライベートクラウドと同様のネットワーク環境をパブリッククラウド上に構築するために、OpenFlow 技術を用いて、パブリッククラウドの VM 上にオーバレイネットワークを構築する。その概要を**図 3** に示す。まずノード VM 間を GRE などのトンネル技術を用いてメッシュ接続することで、ノード VM の Ethernet フレームを相互に授受できるようにする。さらに、このように複数の OpenFlow スイッチ (OFS) で構成されたネットワークを、1 つの Ethernet スイッチとして仮想化するために、OpenFlow コントローラ (OFC) でフローを制御する。コントローラは、`packet_in` で入ったきたパケットの MAC アドレスと受信スイッチ、ポートの組を学習し、パケットを受信したスイッチから宛先までの最短経路を計算し、OpenFlow スイッチにその経路を設定する。

2.4 ストレージマイグレーション

マイグレーションを行う際、VM イメージをどこに配置するかが問題になる。方法として次の 2 つに大別できる。VM イメージを NFS や iSCSI、FC SAN などの共有ストレージに配置し、メモリのみをマイグレーションする方法と、メモリに加えて VM イメージも転送するストレージマイグレーション（ブロックマイグレーションとも呼ぶ）である。前者の方が転送量が少ないので、マイグレーション時間自体は短くなる。しかし、マイグレーション後の VM はマイグレーション元の共有ストレージにアクセスすることになるので、ストレージ性能の悪化が問題になる。特に広域環境では致命的な性能問題になる可能性が高い。

そこで本論文では、後者のストレージマイグレーションを採用する。QEMU の実装では、VM イメージをすべて転送するフル・ストレージマイグレーションと、ベースイメージをあらかじめマイグレーション先と共有しておくことで、マイグレーション時には差分のみを転送するインクリメンタル・ストレージマイグレーションが存在する。一般的なクラスタ環境は均質なノード構成であり、仮想クラスタにおいてもノード VM 間の差分は小さいと考える。そこで仮想クラスタに一つのベースイメージを用意し、ノード VM のイメージはそこからの差分として保持することで、クラウド間のデータ転送量を削減できる。

なお、共有ストレージの問題を解決する提案として、広域分散ストレージの利用 [7] も検討されているが、本論文では可能性として言及するにとどめる。また、ストレージマイグレーション時の転送量を削減する方式として、事前に緩やかにストレージを同期する研究 [1] やファイルキャッシュの WAN 間転送を削減する研究 [2] も行われている。これらの研究との統合は今後の課題であるが、本論文ではこれ以上言及しない。

3. プロトタイプ実装

前節で述べた提案方式の動作を商用パブリッククラウド上で実証するために、プロトタイプ実装を開発した。

まず、プライベートクラウドからパブリッククラウドへ仮想クラスタを移送するには、パブリッククラウド側が VM マイグレーションに対応している必要がある。しかし、我々が知る限りそのようなパブリッククラウドは現存しない。この制限を回避するために、入れ子型仮想化 (Nested Virtualization) 技術 [8], [9] を利用した。入れ子型仮想化の一実装である Nested KVM は、Intel VT や AMD-V などの CPU の仮想化支援機構を、KVM のゲスト OS から利用できる仕組みであり、KVM を入れ子状に実行することを可能にする。近年、KDDI Web CloudCore VPS [10] など、Nested KVM に対応したクラウドサービスも提供され始めている。そこで、クラウド事業者から提供される VM (L1 VM とも呼ぶ) の上に、ノード VM (L2 VM とも呼ぶ) を作成することで、仮想クラスタを構築した。

Ninja migration の実装は KVM への修正を要求する。そこで L1 VM 内の KVM を修正した。一般的なクラウドで VMM を改変することは不可能であるが、入れ子型仮想化に対応した環境では、これが可能になるという利点がある。

ネットワークは、SSH などの管理用と MPI 通信で用いる通信用の 2 種類に分けられる。管理用には Ethernet を想定する。通信用には Ethernet や Infiniband が考えられるが、既存のパブリッククラウドに Infiniband を提供するものが存在しないので、本論文ではパブリッククラウド側は Ethernet を前提とする。また、パブリッククラウド上の VM はグローバル IP が割り当てられたインタフェースを 1 つしか持たないことが多いので、管理と通信を 1 つのインタフェースで兼用する。したがって、準仮想化デバイ

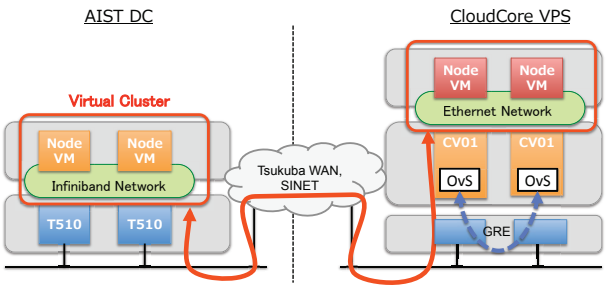

図 4 実験環境

スが前提となる。

仮想ネットワークの構築では、ノード VM 間を GRE トンネルでメッシュ接続する。OpenFlow スイッチとして Open vSwitch を用い、コントローラとして、OpenFlow コントローラフレームである Trema [11] を用いて実装された routing switch を用いた。なお、GRE トンネルの実装として、Linux カーネル実装や Open vSwitch が提供するユーザレベル実装が存在するが、今回は後者を用いた。

4. 実験

4.1 実験環境

提案方式の実現性を検証するために、産総研の計算機室とパブリッククラウド KDDI Web CloudCore VPS（以下、CloudCore と呼ぶ）を接続した実験環境を準備した。その概要を**図 4** に示す。産総研と CloudCore 間は SINET 経由で接続され、往復遅延は約 7.5 ミリ秒であった。Iperf で計測した帯域は、産総研から CloudCore 向きが 920 Mbps であるのに対して、逆向きは 490 Mbps と非対象であった。特に後者の変動は大きかったが、今回の実験規模には十分な帯域である。

本実験では、産総研内の物理サーバ（Dell PowerEdge T410）の 2 台と、CloudCore の仮想インスタンス（CV01 モデル）の 2 台を利用した。各諸元を**表 1** にまとめる。なお、CloudCore 側は VM の仕様であり、物理計算機の仕様は公開されていない。T410 間は Infiniband QDR で接続され、スイッチとして Mellanox IS5022 を用いている。T410 では、OS として Debian GNU/Linux 7.1 (Linux kernel 3.2.0-4-amd64) を用いた。一方、CloudCore では、OS として CentOS 6.2 (Linux kernel 2.6.32-220.el6.x86_64) を用いた。QEMU はバージョン 1.4.1 を用い、VM マイグレーション関連のオプションはデフォルト設定のままとした。仮想ネットワーク構築に用いた、Open vSwitch はバージョン 1.9.1、Trema はバージョン 0.3.20 である。仮想クラスタのノード VM には仮想 CPU 1 個、メモリ 512 MB、HDD 5 GiB (QCOW2 フォーマット) を割当て、ゲスト OS として CentOS 6.4 をインストールした。また、MPI 処理系として Open MPI 1.6.5、Infiniband スタックとして OFED 3.5-1-RC1 を用いた。

表 2 ストレージマイグレーション時間 [秒]

	#VM	AIST → CloudCore	CloudCore → AIST
all	1	172	179
	2	215	299
inc	1	9	9
	2	9	9

4.2 ストレージマイグレーションの実行時間

産総研・CloudCore 間でのストレージマイグレーションの実行時間を測定した。QEMU は 2.4 節で述べたように、フル・ストレージマイグレーション (storage-all) と、インクリメンタル・ストレージマイグレーション (storage-inc) に対応している。1 VM と 2 VM のマイグレーション時間を測定した結果を、**図 2** に示す。なお、VM はアイドル状態であった。

インクリメンタル・ストレージマイグレーションによるマイグレーション時間の削減効果は明白である。フル・ストレージマイグレーションの場合、メモリとストレージを合計した総転送量が 5.4 GB に達するのに対して、インクリメンタル・ストレージマイグレーションの場合は 260 MB 程度であった。フル・ストレージマイグレーションでは広域網がボトルネックになっているのに対して、インクリメンタル・ストレージマイグレーションの場合、この程度の VM 数では転送量が少なく、広域網がボトルネックにならない。事前にベースイメージをマイグレーション先と同期しておく必要があるが、インクリメンタル・ストレージマイグレーションは広域網のマイグレーションにおいて有効であることがわかった。

4.3 Ninja migration の実行時間

Ninja migration を用いて、産総研と CloudCore 間の仮想クラスタマイグレーションに掛かる時間を測定した。なお、ここではストレージマイグレーションに掛かる時間は除き、Ninja migration のオーバヘッドだけに焦点を当てる。

実験の結果、産総研から CloudCore への場合で約 4 秒、CloudCore から産総研への場合で約 22 秒かかった。オーバヘッドの内訳は、PCI hotplug によるデバイス挿抜および検出処理、リンクアップ時間に大別できる。PCI パススルーデバイスに関わる処理時間が主であり、virtio_net に対する処理時間は無視できる。特に Infiniband のリンクアップ時間が約 20 秒と大きい。一般的な HPC クラスタ環境では、リンクアップが頻繁に発生することはないので問題とはならないが、本提案では無視できない影響を及ぼす。この問題が Mellanox 社製 HCA に依存するのか切り分けるために、Qlogic QLE7340 を用いて実験を試みた。しかし、実験環境では PCI パススルーを用いた動作に失敗するので、投稿時までに確認できなかった。さらなる解析は今後

表 1　サーバの諸元

	Dell PowerEdge T410	CloudCore CV01
CPU	hexa-core Intel Xeon X5650/2.66GHz	dual-core Intel Core2 duo T7700/2.40GHz
Chipset	Intel 5520	-
Memory	6 GB (DDR3-1333)	2 GB
Infiniband	Mellanox ConnectX-2 (QDR)	-
Ethernet	Broadcom NetXtreme II BCM5716 (GbE)	virtio_net
HDD	SATA HDD 1 TiB	100 GiB

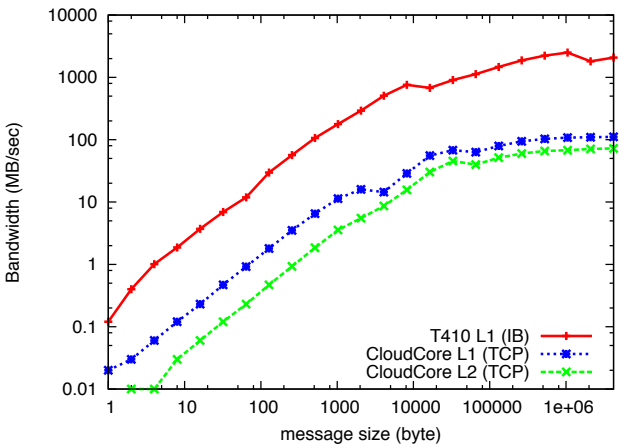

図 5　MPI Point-to-Point 通信性能

の課題としたい。

4.4　仮想クラスタの通信性能

Ninja migration を用いることで、Infiniband クラスタ上では Infiniband を、パブリッククラウド上では Ethernet と使い分けることが可能になる。Intel Microbench Benchmark (IMB) の pingpong ベンチマークを用いて、各仮想クラスタにおける MPI 通信性能を比較した。結果を**図 5** に示す。参考値として、CloudCore CV01 自体の性能 (CloudCore L1) も合わせて計測した。メッセージサイズを大きくするに従い、グッドプットが向上する。メッセージサイズが十分に大きな場合、Infiniband の L1 VM では 2.1 GB/s、Ethernet (virtio_net) の L1 VM では 111 MB/s、L2 VM では 72.4 MB/s であった。CloudCore のインターコネクトはギガビットイーサネットなので、物理性能比は L1 VM で 89%、L2 VM で 58%となる。この性能低下には仮想化と GRE トンネルのオーバヘッドが含まれる。入れ子型仮想化における I/O 通信性能向上は大きな課題であるが、5 節でも言及するようにハードウェア仮想化支援機構の進展により、いずれ無視できるようになると楽観的に予測している。

5.　関連研究

VM マイグレーションはクラウドやエンタープライズ環境で広く利用されているが、我々の知る限り、ヘテロな環境を対象にした研究は多くない。CPU や I/O デバイスなどは、仮想化によって物理環境の不均質性を隠蔽できるが、特に I/O デバイスではエミュレーションによるオーバヘッドが問題になる。この問題を解決する手段として、VMM をバイパスして、VM から直接デバイスにアクセスする PCI パススルーや SR-IOV などの機能が提案されている [4]。Ninja migration は、PCI パススルーを用いて物理デバイスに直接アクセスしている場合でも、そのアプリケーションを再起動することなく、異なる物理計算機へのマイグレーションを可能にする。Vagrant [3] は、Xen と KVM など、VMM が異なる環境間におけるマイグレーションを実現する。デバイスに関しては、エミュレーションを前提としている。Vagrant と Ninja migration は相補的な提案であり、組み合わせることでより柔軟性の高いマイグレーションが可能となる。

入れ子型仮想化は、主要な VMM で利用可能な技術であり、本論文では Nested KVM [8] を用いた。現時点の実装では、4 節で示したように性能オーバヘッドは無視できない。しかし、Intel は入れ子型仮想化を支援するハードウェア機能である APICv や VMCS shadowing などを拡張する予定であり、今後大幅な性能改善が期待できる。論文 [9] では、Xen の入れ子型仮想化を用いて、Amazon EC2 との VM マイグレーションに成功したと報告している。しかし、あくまで単体 VM のマイグレーションであり、本提案のような仮想クラスタでの利用については言及されていない。また、入れ子型仮想化は技術的には興味深いが、ユースケースの議論はまだ十分であるとは言えない。Virtage が提供する KVM on LPAR も入れ子型仮想化技術を利用しており、論文 [12] では、VMM as a Service やバックアップの効率化などのユースケースが示されている。我々が提案する HaaS (Hardware as a Service) モデル [13] も要素技術として入れ子型仮想化を用いている。

6.　まとめと今後の予定

本論文では、プライベートクラウドとパブリッククラウドなど、通信デバイスが異なるヘテロなクラウド環境間で仮想クラスタをマイグレーションする手法を提案した。本提案の特長は、仮想クラスタを構成するノード VM に対して、実行環境の変化に応じて、アプリケーションが最良の通信性能を達成する通信デバイスを透過的に選択し、利

用できることである。本機構は、異なる通信デバイスを有するクラスタを跨いで仮想クラスタをマイグレーションする機構 Ninja migration、OpenFlow を用いたエッジオーバレイネットワーク、ストレージマイグレーションなどの要素技術から構成される。プロトタイプ実装を開発し、産総研の Infiniband クラスタと KDDI Web CloudCore VPS 間で仮想クラスタをマイグレーションできること実験により確認した。本提案により、マイグレーション先の候補が増加することが期待でき、大規模災害時の障害回避技術として有効であると考える。一方、実用化に向けては、ストレージマイグレーションの効率化や、外部ネットワークアクセスの最適化など、本論文で対象外とした課題が残されている。

今後の予定として、実装の完成度を上げ、より大規模な実証実験を行うことが挙げられる。また、現在の Ninja migration の実装は、MPI システムに強く依存しているという制限がある。MPI に依存しないゲスト OS・VMM 間通信機構を実現することで適用範囲をさらに広げたい。さらに、本論文では単純な仮想クラスタをマイグレーション対象としたが、我々が提案している HaaS システム [13] に導入すれば、IaaS 基盤を丸ごと遠隔地にマイグレーションする、IaaS マイグレーションにも応用可能であると考える。

謝辞 本研究の一部は、JSPS 科研費 (24700040) の成果を活用している。

参考文献

[1] 広渕崇宏，マウリシオツガワ，中田秀基，伊藤　智，関口智嗣：仮想マシンの超広域ライブマイグレーションにむけたベストエフォート型状態同期機構の試作，情報処理学会研究会報告 2012-OS-121，pp. 1–8 (2012).

[2] Akiyama, S., Hirofuchi, T., Takano, R. and Honiden, S.: Fast and Low-Overhead Wide Area Live Migration by Restoring Page Cache from Disk Image, *Proc of the IEEE/ACM 13th International Conference on Cluster, Cloud and Grid Computing (CCGrid 2013), Doctoral Symposium* (2013).

[3] Liu, P., Yang, Z., Song, X., Zhou, Y., Chen, H. and Zang, B.: Heterogeneous Live Migration of Virtual Machines, *Proc. of the International Workshop on Virtualization Technology (IWVT)* (2008).

[4] 高野了成，池上　努，広渕崇宏，田中良夫：HPC クラウドの実現に向けた仮想化クラスタの性能評価，情報処理学会論文誌コンピューティングシステム（ACS），Vol. 5, No. 2, pp. 111–120 (2012).

[5] Takano, R., Nakada, H., Hirofuchi, T., Tanaka, Y. and Kudoh, T.: Ninja Migration: An Interconnect-transparent Migration for Heterogeneous Data Centers, *Proc. of the 10th High-Performance Grid and Cloud Computing Workshop (HPGC)*, pp. 992–1000 (2013).

[6] Takano, R., Nakada, H., Hirofuchi, T., Tanaka, Y. and Kudoh, T.: Cooperative VM Migration: a Symbiotic Virtualization Mechanism by Leveraging the Guest OS Knowledge, *IEICE Transactions on Information and Systems* (2013).

[7] 北口善明，近堂　徹，柏崎礼生，中川郁夫，下條真司：広域分散ストレージを用いた長距離ライブマイグレーションの評価実験，電子情報通信学会技術研究報告 IA2013-7，pp. 37–42 (2013).

[8] 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. of the 9th USENIX conference on Operating systems design and implementation (OSDI)*, pp. 1–6 (2010).

[9] Williams, D., Jamjoom, H. and Weatherspoon, H.: The Xen-Blanket: Virtualize Once, Run Everywhere, *Proc. of the ACM European Conference on Computer Systems (EuroSys)* (2012).

[10] KDDI Web Communications CloudCore VPS: `http://www.cloudcore.jp/`.

[11] Trema: Full-Stack OpenFlow Framework in Ruby and C: `http://trema.github.io/trema/`.

[12] 水野和也，服部直也，田窪俊二：2 段仮想化構成のユースケース提案と評価，電子情報通信学会全国大会予稿集，p. 91 (2012).

[13] 高野了成，中田秀基，竹房あつ子，柳田誠也，工藤知宏：インタークラウドにおける仮想インフラ構築技術の提案，情報処理学会研究会報告 2013-OS-124，pp. 1–8 (2013).