Vol.2015-OS-134 No.20
2015/8/5

PaaS 環境におけるメモリ情報を用いた LiveMigration の負荷軽減

上司　陽平[†1,a)]　古藤　明音[†1,b)]　河野　健二[†1,c)]

概要：近年，オペレーティングシステム (OS) の仮想化技術がクラウドコンピューティングの基盤技術として広く用いられている．仮想化技術で構築された環境の管理には Live Migration が有用である．Live Migration は仮想マシン (VM) を稼働した状態でホスト間を移送する技術である．例えば，物理マシン上の VM の負荷が増大し，計算資源が不足した場合に負荷を分散するために使用する．しかし，Live Migration の現在の手法では CPU 時間やネットワーク帯域などの資源を多く消費してしまう．そのため負荷分散ために Live Migration を実行しても，Live Migration の負荷により資源競合が深刻化してしまう可能性がある．本研究では PaaS 環境上でホスト間のメモリ共有情報を用いて，移送先で共有できるページを送信しないことで Live Migration のメモリ転送量を削減する．メモリ転送量を削減することで Live Migration 実行に伴う CPU 時間とネットワーク帯域を軽減し移送時間を削減する．本提案により既存手法と比べて CPU 使用率を最大 26.3%, ネットワーク使用量を最大 10.0%，移送時間を最大 27.8% 削減することができた．

キーワード：仮想化，Live Migration, Page Sharing，サーバコンソリデーション

1. はじめに

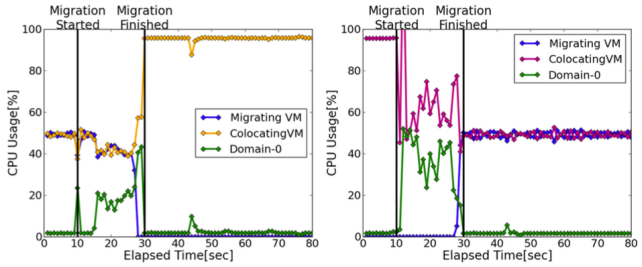
近年，オペレーティングシステム (OS) の仮想化技術がクラウドコンピューティングの基盤技術として広く用いられている．OS の仮想化技術とは 1 台の物理マシン上で複数の仮想マシン (VM) を稼働させる技術である．クラウド環境では，数百万台に及ぶ物理サーバで構成される大規模な計算資源を，OS の仮想化技術により管理している．これにより，クラウドの利用者は大規模な計算資源のうち，必要な計算資源のみを利用することができる．例えば Amazon Web Services [1] や Windows Azure [2] ではクライアントがサービス提供に必要な OS などの環境やソフトウェアが予め構築された VM をサービス基盤として貸し出す．このサービスは Platform as a Service(PaaS) と呼ばれる．

OS の仮想化技術の一つとして大規模な計算資源を柔軟に管理することができる Live Migration [3] がある．Live Migration はサービスを停止することなく VM をホスト間で移送させることができる．例えばあるホスト上で VM の負荷が増大して資源競合が発生する場合がある．この時，Live Migration を用いることでクラウドサービスの稼働を停止することなく他のホストへ負荷を分散させることができる．また，ホスト上の全 VM を他のホストに Live Migration することで不使用になった物理マシンのメンテナンスを行うこともできる．

しかし既存の Live Migration では多くのメモリページを転送する．そのため移送に CPU 時間やネットワーク帯域などの計算資源を多く消費してしまう．これにより負荷分散時に移送元ホストの負荷を大きく増やしてしまう．それが原因でホスト上の VM に割り当てられる CPU 時間やネットワーク帯域などの計算資源が減ってしまう場合がある．特に PaaS 環境では，複数の物理マシン上で同じ構成の VM が稼働しているため，ホスト間で同じページを確実に保持している (メモリの共通性)．その場合でも Live Migration では移送先にもあるページを含め全てのページを送っている．

本研究では，PaaS 環境において移送負荷を緩和する Live Migration 手法を提案する．本手法は，PaaS 環境における VM のメモリの共通性を利用し，移送の両ホスト (移送元ホストと移送先ホスト) に存在する共通のページの転送を行わないことでメモリ転送量を削減する．メモリの転送量を削減することで Live Migration による CPU 時間やネットワーク帯域などの計算資源消費を軽減する．本提案を XEN 4.3.3-rc1, Ubuntu14.04 LTS を用いて実装した．

[†1] 現在，慶應義塾大学
Presently with Keio Uniersity
[a)] uhoi@sslab.ics.keio.ac.jp
[b)] kono@ics.keio.ac.jp
[c)] koto@sslab.ics.keio.ac.jp



(a) Source host (b) Target host

図 1 Live Migration による CPU 使用率への影響 [4]

本提案により CPU 使用率を最大 26.3%, ネットワーク使用量を最大 10.0%，移送時間を最大 27.8% 既存手法と比べて削減することができた．

本論文の構成を以下に示す．2 章では Live Migration の負荷について，3 章では本提案についての概要や実装などを，4 章では本提案の評価，5 章では関連研究，最後に 6 章ではまとめを述べる．

2. Live Migration による PaaS の性能低下

Live Migration は実行に多くの CPU 時間やネットワーク帯域などの計算資源を要することから，PaaS の性能を低下させることがある．本章では，Live Migration が引き起こすクラウドサービスの性能低下について述べる．

図 1 は Live Migration 時の各 VM の CPU 使用率を示したものである．(a) の Source host が移送元のマシン，(b) の Target host が移送先のマシンとなる．Migrating VM は移送される VM で Colocating VM は移送に関係のない host 上にある VM である．

Live Migration は実行に多くの計算資源を消費するので Migration VM の資源使用率が低下する場合がある．図 1(a) の Migration VM (青線) を見ると，CPU 使用率が Live Migration 時に 50% 近くから約 40% まで低下している．これは Live Migration と Migration VM 間での資源競合が発生したためである．

同様に Live Migration により Colocating VM でも計算資源の使用率が低下する場合がある．図 1(a) では Colocating VM の CPU 使用率が Migrating VM 同様 40% 近くに低下している．図 1(b) では Colocating VM の CPU 使用率が Live Migration 開始前と比べて 20-50% 程低下していることがわかる．ここで注意したい点は，Migrating VM が稼働しているホストと共に稼働していない移送先ホストでも Live Migration により資源使用率が低下することである．これらは Live Migration と Colocating VM 間での資源競合が発生したためである．

また Live Migration の移送時間が長い場合，負荷分散を行う際に移送中のワークロード変化により不必要な移送が行われる場合がある．例えば負荷分散の時ある VM を他のマシンに移送することを考える．その時，移送中にホスト上のある VM のワークロードが低下したことによりホスト上での負荷分散が必要なくなってしまう場合がある．そのような場合は移送の必要がないにも関わらず，Live Migration を上記で説明したような影響を及ぼしながら行うことになる．

3. 本提案手法の説明

3.1 概要

本提案ではメモリ転送量を削減し PaaS 環境において発生する Live Migration の負荷を軽減する．本提案ではメモリ転送量を削減するために，移送 VM が移送先と共通に持つ同じページは送信しない．送信しなかった分のページは移送先で既にあるページと共有を行う．これによりメモリ転送量を減らすことで Live Migration の移送時間を減らし，消費する CPU 時間やネットワーク帯域などの計算資源消費を軽減することができる．

3.2 アプローチ

本提案は PaaS のメモリの共通性を利用して移送先にもあるページは転送しない．これによりメモリ転送量を削減する．メモリの共通性とはホスト間に同じページが多く存在する性質である．PaaS 環境では各ホスト上に同じ OS，同じソフトウェアで構築された VM が複数存在する．そのため各 VM がテキスト領域に同じ内容を持つことになるのでこのような性質ができる．PaaS のようなメモリの共通性がない場合，移送先ホストにまったく同じページがない可能性がある．

また PaaS のメモリの共通性を利用すると，移送時に送らないページの確認を効率的に行うことができる．具体的には移送元で移送 VM が既に共有しているページに限定して送らないページの確認を行う．これは，メモリの共通性があるので移送元で共有できているページが移送先で同様に共有できるためである．このように共有できるページをメモリ共有情報を用いて的確に確認することで非転送ページの確認を効率的に行うことができる．

送らないページを確認するためのページ比較にはハッシュ値を用いる．ハッシュ値を利用することでページの全体を送らなくても，移送先と共通の移送 VM が持つ同じページを確認することができる．

3.3 実装

本提案の実装を図 2 に示す．ハイパーバイザには XEN4.3.3-rc1 を用いて実装した．本提案のためのページ管理モジュールをハイパーバイザに，Migration モジュールを Domain-0 に実装した．

ページ管理モジュールは本提案で必要となるメモリの共有情報と比較に用いるハッシュ値を，CBPS システム [5] が管理しているメモリ管理機構から得ることができる．

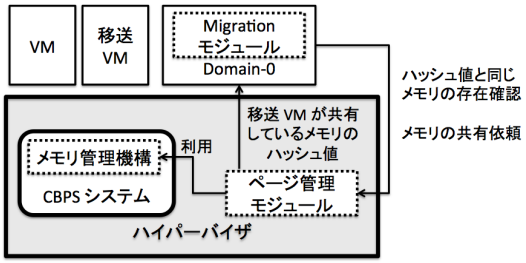


図 2　実装

CBPS システムは XEN にはないため実装した．また移送先に送られるページのハッシュ値によるメモリ比較もこの CBPS システムのメモリ管理機構を利用して行うことができる．

Migration モジュールはハイパーバイザから移送 VM の共有しているメモリとそのハッシュ値を得ることができる．また移送元から受け取ったメモリ比較用のハッシュ値をページ管理モジュールに渡すことでそのハッシュ値を持つメモリの存在確認をすることができる．これによって送信しないで済むページの確認をする．Migration モジュールは送信しないページの確認が済んだらそれ以外のページを Pre-copy [3] で移送する．移送しなかったページは移送先でページ管理モジュールに共有を依頼することで送らないで済む．

本提案ではハッシュ値の比較のみでメモリが同等であると判定している．今回の実装ではハッシュ値計算アルゴリズムには SHA-1 を用いている．この時，ランダムに選ばれたハッシュ値が衝突しない確率は SHA-1(160bit) で 48 nines （”0.9999...9” の 9 の数が 48 個という意味）で，TCP がネットワーク上でのブロック転送をする時に発見できないエラーが起きない確率の 8（または 9) nines より大きいため無視しても構わない．[6]

4. 評価

4.1 実験環境

マシン配置を図 3 に示す．移送元（Source)，移送先 (Destination) ホストは XEN 4.3.3-rc1 で Domain-0 には Ubuntu14.04 LTS を用いた．どちらのマシンも同じ機種で Intel Xeon CPU X5650, 16GB Memory , Gigbit Ethernet である．それぞれのマシンには Migration の専用線として物理的に直接ケーブルを繋げてあり, Migration 通信のみこのケーブルを使用して通信を行う．また VM のストレージには共有ストレージとして NFS を用いた．NFS サーバには Ubuntu 14.04 LTS, Intel Xeon CPU X3480, 16GB Memory, Gigbit Ethernet を用いた．それぞれの VM は Ubuntu 14.04.1 LTS, 512 MB Memory で起動する．VM は全部で移送元の移送しない VM(Source VM), 移送する VM(Migration VM), 移送先の移送しない VM(Destination VM) の 3 つを起動する．

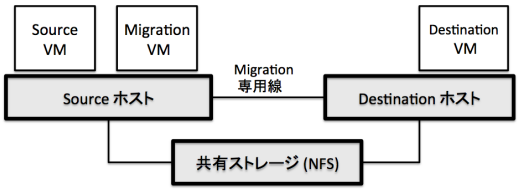


図 3　マシン配置図

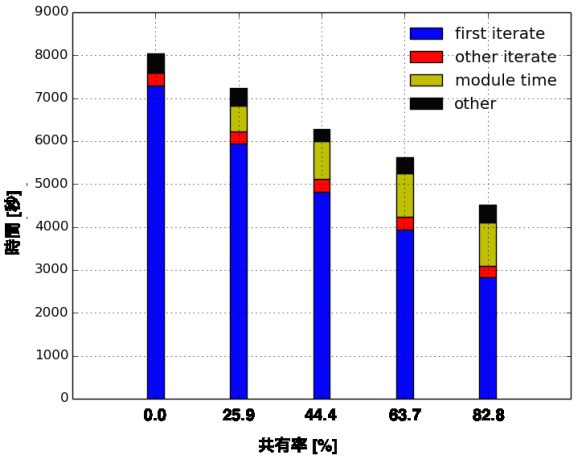


図 4　Micro Benchmark における非転送ページ量と移送時間

実験では Source VM, Migration VM, Destination VM でそれぞれ同じワークロードを動かしながら CBPS による Page Sharing が行われた状態で Migration VM を Destination ホストに Live Migration する．その時のメモリ転送量と CPU 使用率を測定した．ワークロードには以下のものを用いた．

- Micro benchmark：mmap で各 VM 間で共有できるページの量を変更して非転送ページの量を調整するベンチマーク．
- HTTPERF：HTTP リクエストを送信するベンチマーク．レスポンス量などを測定することができる．[7]
- MEMASLAP： memcached のリクエストを送信するベンチマーク．スループットなどを測定することができる．[8]
- KERNEL BUILD： kernel コンパイルを行うワークロード．

4.2 マイクロベンチ

Micro Benchmark の共有率と移送時間の関係を図 4 に示す．X 軸は移送先に送信しなかったページ量の VM の全体のメモリ量を元にした百分率となっている (共有率)．つまり共有率が高いほど移送しなかったページが多いということになる．0% の時は本提案のモジュールを動作させておらず，それ以外は動作させている．なお，Page Sharing の機能は常に動作している．Y 軸は Live Migration にかかった時間で，最初のメモリ全体を移送する First Iterate

表 1　Micro Benchmark における共有率に伴った転送ページ数

提案モジュール	なし	あり	あり	あり	あり
共有率 (%)	0.0	25.9	44.4	64.7	82.8
送信したページ数	135158	101172	76960	51269	25920

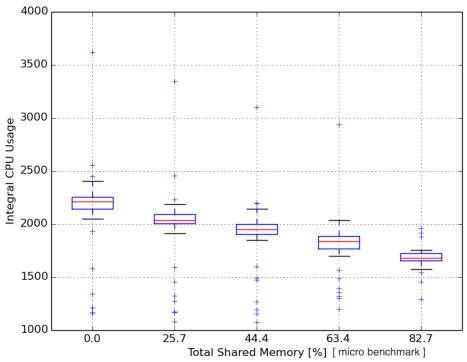


図 5　Migration 中の Domain-0 CPU 積分値 (Source)

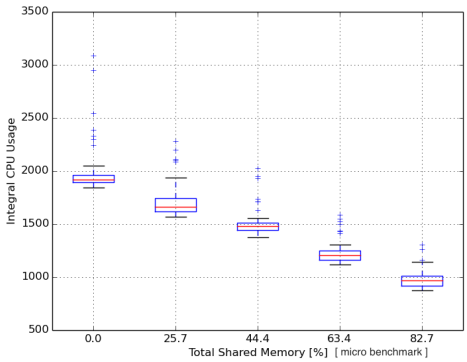


図 6　Migration 中の Domain-0 CPU 積分値 (Destination)

とそれ以外の Iterate, 本提案のモジュール，それ以外の初期設定などにかかった時間を内訳とする．図を見てみると，移送元での共有率が上がるほど移送にかかる時間が減少していることがわかる．内訳を見ると共有率が増えるほど最初のイテレート時に送信するページが減るので First Iterate の時間が短くなり，それを処理するためのモジュールの時間が長くなっていることがわかる．50 回の試行で得た実験結果の中央値で比較したとき，移送時間を最大 44.5% 削減することができた．

表 1 は Micro Benchmark の実験結果を示したものである．送信したページとは実際に送信したページ数である．この結果から共有率が増えるほど移送しないページが増えているので移送するページ数の総量が減少していることがわかる．ページを送らないことでネットワークの資源消費を抑えることができる．この実験では最大 80.8% のネットワーク使用量を削減している．Satori [9] の I/O 動作をモニタリングし，それをヒントに共有を行う方法では HTTPERF で約 63 % のシェアを実現している，その方法を用いればネットワークの資源消費を約 60% 程抑えられることがこの実験結果から予測できる．

図 5 と図 6 は Live Migration 中の domain-0 の CPU の積分値の箱ヒゲ図となっている．試行回数を 50 回にして実験を行った．X 軸は実験で得た共有率の中央値である．実験結果では Source ホストも Desitination ホストも統計的に共有率が上ると CPU の使用率が下がることがわかる．これは移送しなかった分のページ処理が減ったため CPU の使用を抑えることができたからである．つまり本提案により，共有率に依存しながら CPU の資源使用を移送先，移送元の両方で軽減することができる．

Micro Benchmark では本提案により共有率が増えれば増えるほど移送時間が減り，CPU 時間と ネットワーク帯域などの資源消費を抑えることができることがわかった．では実際に使用されるようなワークロードではどのような結果になるかを次に示す．

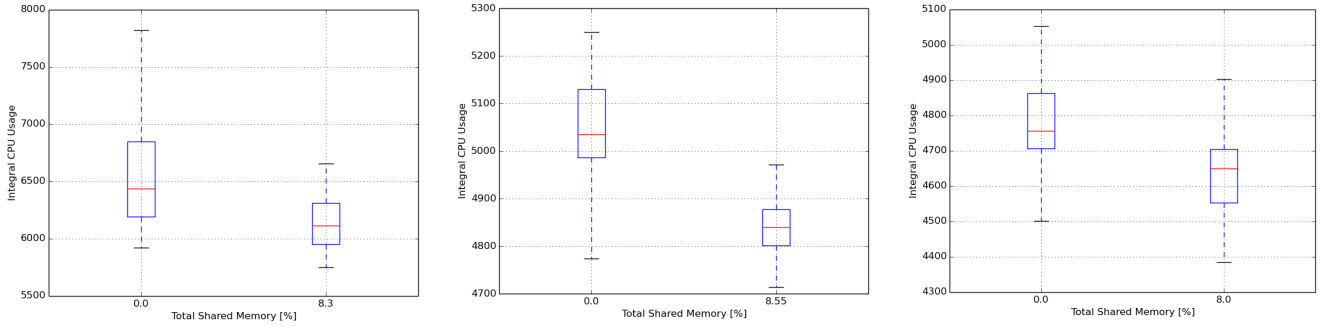
4.3　リアルベンチ

実際に使われるようなワークロードを使用して実験を行った．ワークロードには HTTPERF, MEMASLAP, KERNEL BUILD を用いる．4.3.1 では前節同様，既存手法と本提案の移送時間と計算資源の消費量について考察する．4.3.2 では本提案により Live Migration の負荷を軽減できた場合の VM への影響について考察する．

4.3.1　移送時間と計算資源の消費量

共有率は全てのワークロードで約 8%であった．最大の共有率でも MEMASLAP の 8.5% となった．実際に使われるワークロードではアプリケーションのメモリは書き換えが多く共有する前に書き換えられたり，共有しても共有がすぐとかれたりしてしまう場合がある．またワークロードによってはアプリケーションは独自のデータを生成するためそもそもアプリケーションのデータは共有が難しい．このためあまり共有率が上がらなかった．しかし PaaS のメモリの共通性により，OS などの同じ部分は確実に共有が行えるため 8% 近くは確保できた．

移送時間は既存手法と比べて, 最大 HTTPERF で 27.8%, KERNEL BUILD で 11.4%, MEMASLAP で 8.4% 削減できた．HTTPERF はメモリの書き換えが他のワークロードに比べて多かった．そのため移送時間が早くなる分書き換えの時間が少なくなるので送り直すメモリが減る．送り直すメモリが減ると移送時間が減るので移送時間は指数関数的に減少する．そのため他のワークロードと比較して良



(a) HTTPERF (b) MEMASLAP (c) KERNEL BUILD

図 7 各ワークロードでの Live Migration 中の Domain-0 CPU 積分値

い結果がでたと推測される．

ネットワークの資源消費は HTTPERF で 10.0%, KERNEL BUILD で 5.9%, MEMASLAP で 8.5% 抑えることができた．Satori [9] とは異なり，本提案の CBPS による共有では I/O の動作をヒントとして使用しないため HTTPERF の共有率が Satori より少なかった．また KERNEL BUILD では共有ストレージの NFS への書き込みが頻繁に発生するため NFS 依存な動きになってしまう．そのため NFS の処理待ちなどのために Migration 中のメモリへの書き込み回数が少なくなり Live Migration がモジュールの有無に関係なく他のワークロードより早く終わる．そのため差が少ない結果となった．

Live Migration 中の Domain-0 の CPU の積分値の実験結果を図 7 に示す．この図は移送元と移送先の Domain-0 CPU USAGE を積分した値である．どのワークロードでも移送先，移送元の両方で CPU の積分値は減少した．そのため合計値も減少しているのがわかる．全てのワークロードにおいて CPU 使用率積分値の中央値が本提案のモジュールでメモリ転送量を削減したことにより軽減できている．HTTPERF で最大 26.3% の CPU 積分値を軽減できた．

これらの結果から実際に用いられるアプリケーションでも本提案により Live Migration の移送時間を減らし，CPU 時間とネットワーク帯域などの計算資源消費を抑えることができる．

4.3.2 VM への影響

4.3.1 で本提案により Live Migration の負荷を軽減できることを示した．負荷を軽減した分ホスト上の VM へどのような影響があるかを示す．図 8 は HTTPERF での Source VM のスループットを示している．赤線が本提案のモジュールなし，緑線がモジュールありで 7.2% の共有率であることを示している．図を見ると，本提案 (緑線) の方が Live Migration が早く終わる分スループットが上がるのが早いことがわかる．スループットが上がるのは Migration VM が使っていた分の計算資源を移送後に Source VM で使用できるようになったためである．Live Migration の移送時間を短くすることで移送に関係のない VM の資源競合を既存手法より早く解決することができる．

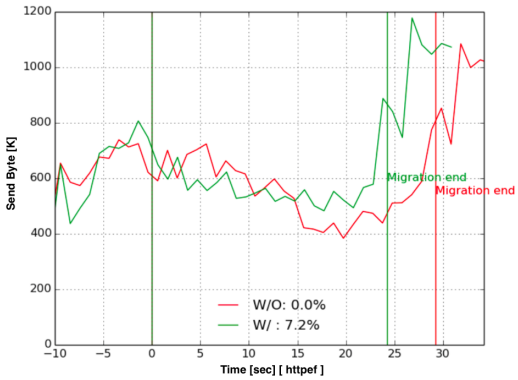


図 8 Source VM のスループット [httpef]

表 2 は MEMASLAP の Live Migration 中における CPU USAGE の平均値である．それぞれの値は 30 回の試行をした時の中央値である．結果は移送先で Domain-0 の CPU が減りその分 Destination VM の CPU が大きくなった．これは移送先で Domain-0 が処理するページ数が少なくなったので CPU 使用率が減り，その分 Destination VM に CPU が割り当てられるためである．しかし今回の結果では移送元にあった VM の Domain-0, Source VM, Migration VM の CPU USAGE の値は同じとなった．これは本提案による処理がページを送信しなかった分削減した CPU を使用してしまったため結果が既存手法と同じになったと考えられる．しかし Micro Benchmark でも示したように共有率が上がれば CPU USAGE はさらに削減できるのでワークロードによってはこれ以上の効果を期待できる．

本提案によって移送に関係のない VM に対する Live Migration に伴う負荷を削減することができた．

5. 関連研究

Live Migration の負荷軽減についてはいくつかの研究がなされている．Miyakodori[10] ではサーバコンソリデーション時に一度 Live Migration した VM が同じホストに戻ってくる場合があることを利用して, Live Migration 時

表 2 Memaslap の Live Migration 中の CPU Average

		CPU USAGE AVERAGE (%) [MEMASLAP]	
モジュール	共有率 (%)	Domain-0(Destination)	Destination VM
なし	0.0	56.0	32.0
あり	8.3	55.0	36.0

に VM のメモリを完全に解放しないことで再度移送されて来た時まだ変更されていないページを再利用する．Sonic Migration [11] ではメモリキャッシュなど移送先で再構築可能なメモリなどを移送しないことでメモリ転送量を削減し Live Migration の負荷を軽減する手法を提案している．本提案では PaaS 上では共通のデータが既に移送先にあるので VM が以前ホスト上になかった場合でも適用できる．またメモリキャッシュなどは送るため移送後のメモリ再構築などの必要がなくなる．

また Page Sharing の研究も多くなされており，様々な手法が提案されている．Disco[12] で VM にはわからない VMM 上で共有作業を行う TPS(Transparent Page Sharing) が考案され，後に VM Ware [5] で CBPS が考案された．その後 Satori [9] やデータセンタなどで複数のホストマシンがある時，VM 内で最大の共有を得ることができる VM の配置を中央管理のホストで計算する Memory Buddies [13] などが提案されている．またメモリ共有を多く得るためには OS が同じであることが望ましいこと，GUI アプリケーションの共有率が高いこと，ASLR によるメモリ共有への影響などメモリ共有について調査した研究 [14] などもある．Memory Buddies により PaaS による静的なメモリの共有以外にも動的なメモリによる共有についても考慮して最大の共有率を得ることができるので本提案のメモリ転送量をさらに下げることが可能だと考えられる．

Live Migration や Page Sharing については多くの研究がなされてきたが，Page Sharing によるページ共有情報を Live Migration に適用したものはない．またそれによるメモリ転送量削減手法も提案されていない．

6. まとめ

PaaS は多くの会社が提供しており，その基盤構築には仮想化技術が用いられる．そのような大規模サービスの基盤では複数の物理マシン上で複数の同一構成された VM が稼働する．それらの管理手法として Live Migration は有用である．しかし既存の Live Migration では全てのメモリを移送するため，移送時間が長く CPU 時間やネットワーク帯域などの計算資源消費が大きい．そのため不必要な移送を行ってしまったり，負荷軽減に用いる場合などに Live Migration がホストにさらに負荷をかけてしまい移送に関係するホスト上での VM と Live Migration で資源競合が起きてしまう場合がある．それによって PaaS の提供しているサービス性能が低下してしまう可能性がある．

本提案では PaaS 環境において移送先でも共有できるページを移送しないことでメモリ転送量を削減し，Live Migration の負荷を軽減する．本提案により CPU 使用率を最大 26.3%, ネットワーク使用量を最大 10.0%，移送時間を最大 27.8% 削減することができた．また，移送に関係するホスト上の VM への Live Migration による負荷を軽減することができた．

参考文献

[1] Amazon: Amazon Web Services. http://aws.amazon.com/jp/.
[2] Microsoft: Azure. http://azure.microsoft.com/ja-jp/.
[3] Clark, C., Fraser, K., Hand, S., Hansen, J. G., Jul, E., Limpach, C., Pratt, I. and Warfield, A.: Live Migration of Virtual Machines, *Proceedings of the 2Nd Conference on Symposium on Networked Systems Design & Implementation - Volume 2*, NSDI'05, Berkeley, CA, USA, USENIX Association, pp. 273–286 (online), available from ⟨http://dl.acm.org/citation.cfm?id=1251203.1251223⟩ (2005).
[4] Koto, A., Kono, K. and Yamada, H.: A Guideline for Selecting Live Migration Policies and Implementations in Clouds, *Cloud Computing Technology and Science (CloudCom), 2014 IEEE 6th International Conference on*, pp. 226–233 (online), DOI: 10.1109/CloudCom.2014.36 (2014).
[5] Waldspurger, C. A.: Memory Resource Management in VMware ESX Server, *SIGOPS Oper. Syst. Rev.*, Vol. 36, pp. 181–194 (online), available from ⟨http://doi.acm.org/10.1145/844128.844146⟩ (2002).
[6] Henson, V.: Guidelines for Using Compare-by-hash (2005).
[7] Hewlett-Packard Development Company, L.: httperf. http://www.hpl.hp.com/research/linux/httperf/.
[8] Zhuang, M.: memaslap. http://docs.libmemcached.org/bin/memaslap.html.
[9] Miłós, G., Murray, D. G., Hand, S. and Fetterman, M. A.: Satori: Enlightened Page Sharing, *Proceedings of the 2009 Conference on USENIX Annual Technical Conference*, USENIX'09, Berkeley, CA, USA, USENIX Association, pp. 1–1 (online), available from ⟨http://dl.acm.org/citation.cfm?id=1855807.1855808⟩ (2009).
[10] Akiyama, S., Hirofuchi, T., Takano, R. and Honiden, S.: MiyakoDori: A Memory Reusing Mechanism for Dynamic VM Consolidation, *Proceedings of the 2012 IEEE Fifth International Conference on Cloud Computing*, CLOUD '12, Washington, DC, USA, IEEE Computer Society, pp. 606–613 (online), DOI:

10.1109/CLOUD.2012.56 (2012).

[11] Koto, A., Yamada, H., Ohmura, K. and Kono, K.: Towards Unobtrusive VM Live Migration for Cloud Computing Platforms, *Proceedings of the Asia-Pacific Workshop on Systems*, APSYS '12, New York, NY, USA, ACM, pp. 7:1–7:6 (online), DOI: 10.1145/2349896.2349903 (2012).

[12] Bugnion, E., Devine, S., Govil, K. and Rosenblum, M.: Disco: Running Commodity Operating Systems on Scalable Multiprocessors, *ACM Trans. Comput. Syst.*, Vol. 15, No. 4, pp. 412–447 (online), DOI: 10.1145/265924.265930 (1997).

[13] Wood, T., Tarasuk-Levin, G., Shenoy, P., Desnoyers, P., Cecchet, E. and Corner, M. D.: Memory Buddies: Exploiting Page Sharing for Smart Colocation in Virtualized Data Centers, *Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, VEE '09, New York, NY, USA, ACM, pp. 31–40 (online), DOI: 10.1145/1508293.1508299 (2009).

[14] Barker, S., Wood, T., Shenoy, P. and Sitaraman, R.: An Empirical Study of Memory Sharing in Virtual Machines, *Proceedings of the 2012 USENIX Conference on Annual Technical Conference*, USENIX ATC'12, Berkeley, CA, USA, USENIX Association, pp. 25–25 (online), available from ⟨http://dl.acm.org/citation.cfm?id=2342821.2342846⟩ (2012).