Vol.2013-OS-127 No.13
Vol.2013-EMB-31 No.13
2013/12/3

物理マシン間のライブマイグレーション手法の提案

深井 貴明[1]　表 祐志[1]　品川 高廣[2]　加藤 和彦[1]

概要：仮想マシン間での OS のライブマイグレーションは，仮想マシンモニタが提供する機能の一つであり，ハードウェアの障害対応や負荷分散などに有効な技術である．しかし従来の手法では，物理マシン間でのライブマイグレーションは実現されていなかったり，実現手法が OS やファームウェアに依存したものになっていた．本研究では，仮想マシンモニタを用いて OS から透過的に物理マシン間の OS ライブマイグレーションを実現する手法を提案する．準パススルー型仮想マシンモニタを用いて，移動元のマシンで CPU やメモリ，物理デバイスの状態を取得し，移動先のマシンでこれらの状態を復元する．実際に割り込みコントローラ（PIC）やタイマーデバイス（PIT）について状態を取得・復元する機構のプロトタイプを実装し，機能を制限した Linux をマイグレーションできることを確認した．

1. はじめに

OS のライブマイグレーション [1] とは，ある物理マシンで動作している OS を停止することなく別の物理マシン上へ移動する機能である．クラウドコンピューティングにおいて，OS のライブマイグレーションはメンテナンスや負荷分散のために用いられる重要な機能である．例えば，物理マシンのメンテナンスをおこなう際，メンテナンス対象の物理マシン上で動作するサービスを別のマシンに移す必要がある．この時に，OS をライブマイグレーションによって他の物理マシンへ移動することで，サービスを停止することなく物理マシンをメンテナンスできる．

現在，OS のライブマイグレーションは仮想化技術によって実現されている．従来の仮想化技術ではハードウェアを完全に仮想化するため，一定のオーバヘッドが生じる．このオーバヘッドは，データベースや HPC などの高負荷な計算の用途には無視できないものとなる [2]．

既存研究における物理マシン環境上の OS のライブマイグレーション手法は，OS やファームウェアに依存する手法である [3] [4] [5] ．OS に依存する手法は，手法を実現するための変更をなされた OS でしか利用できない．このため，クラウドの顧客が利用可能な OS を制限する．同様に，ファームウェアに依存する手法は，手法を実現するための変更をなされたファームウェアが存在するハードウェアでしか利用できない．このため，クラウド事業者が利用可能なハードウェアを制限する．

そこで，我々は，OS やファームウェアに依存せずに，物理マシン上の OS をライブマイグレーションする手法を提案する．提案手法では，準パススルー型仮想マシンモニタを用いる．準パススルー型仮想マシンモニタとは，デバイスを仮想化せず，OS からハードウェアへのアクセスを可能な限りパススルーしつつ，一部のアクセスを捕捉する仮想マシンモニタである．この準パススルー型仮想マシンモニタを用いて，移動元マシンの CPU，メモリ，物理デバイスの状態を取得し，移動先のマシンでこれらの状態を復元する．仮想マシンモニタを用いることで，OS やファームウェアに依存せずに，OS のライブマイグレーションを実現する．また，準パススルー型仮想マシンモニタはデバイスを仮想化しないため，仮想化のオーバヘッドも削減できる．

準パススルー型仮想マシンモニタは，デバイスを仮想化しないため，デバイスの状態を保持しない．このため，マイグレーションの際，準パススルー型仮想マシンモニタは物理デバイスの状態を取得する必要がある．デバイスは一部の状態を書き込み専用のレジスタに格納する．このため，デバイスの状態の一部をレジスタから読み出せないため，物理デバイスの状態の取得はレジスタからデータを読み込むだけでは不十分である．そこで，本手法では，準パススルー型仮想マシンモニタで OS からデバイスへ発行される入出力命令 (以下，I/O) を監視することで，デバイスの状態を把握する．この方法によって，レジスタから読み出せないデバイスの状態も取得する．また，物理デバイス

[1] 筑波大学大学院システム情報工学研究科コンピュータサイエンス専攻
Department of Computer Science, Graduate School of Systems and Information Engineering, University of Tsukuba
[2] 東京大学情報基盤センター
Information Technology Center, The University of Tokyo

の状態の復元は，移動元マシンで取得したI/Oと同じI/Oを移動先マシンの仮想マシンモニタで発行しおこなう．

以降，2章では研究背景について述べ，3章で設計方針と想定環境について述べる．4章では設計を述べ，5章では現在の実装について述べる．6章で実験とその結果について述べる．7章で関連研究について述べる．8章で今後の課題について述べ，9章でまとめる．

2. 研究背景

2.1 デバイスの仮想化とライブマイグレーション

Xen[6] などの仮想マシンモニタで提供されている完全仮想化や準仮想化のアーキテクチャでは，OSに物理デバイスを直接アクセスさせるのではなく，仮想マシンモニタが提供する仮想デバイスへアクセスさせる．これらのアーキテクチャでは，デバイスを仮想化することで複数の仮想マシン間での物理デバイスの共有や，物理デバイス間の差異の隠蔽を実現している．

また，デバイスの仮想化はOSのライブマイグレーションを容易にしている．仮想マシンモニタがデバイスを仮想化し，仮想化したデバイスをOSにアクセスさせることで，仮想マシンモニタはOSがアクセスするデバイスの状態を容易に把握できる．

一方，デバイスの仮想化は一定のオーバヘッドを生む．これは，データベースやHPCなどの高負荷な計算の用途には無視できないものとなる．そこで，デバイスを仮想化せず，OSに物理デバイスを直接アクセスさせることによって，仮想化によるオーバヘッドを削減する手法 [7] が提案されている．

OSが直接デバイスへアクセスすることにより，性能は改善するものの，ライブマイグレーションは困難になる．なぜなら，OSが物理デバイスへ直接アクセスすると，仮想マシンモニタはOSがアクセスしているデバイスの状態を把握できないためである．

そこで，本研究では，OSがデバイスへ直接アクセスする物理マシン環境において，ライブマイグレーションを実現する手法を提案する．

2.2 準パススルー型仮想マシンモニタ

本研究で用いる準パススルー型仮想マシンモニタを図1に示し，以下で説明する．

準パススルー型仮想マシンモニタは，OSからハードウェアへのアクセスを可能な限りパススルーしつつ，一部のアクセスのみを捕捉する仮想マシンモニタである．これは準パススルードライバによって実現されている．捕捉対象のI/Oは準パススルードライバにより捕捉され，そうでないI/Oは直接ハードウェアへ送られる．補足したI/Oは，単に内容を確認してそのまま物理デバイスへ命令を発行することも，命令の内容を変換することもできる．この方式では，仮想マシンモニタが機能を提供するために必要最低限の入出力命令のみを捕捉し，それ以外の命令をパススルーすることができる．また，準パススルー型仮想マシンモニタでは，デバイスを仮想化せず，OSのデバイスドライバがデバイスの制御をおこなう．

本手法では，この準パススルー型仮想マシンモニタを用いて，ライブマイグレーションに必要な入出力命令を捕捉する．

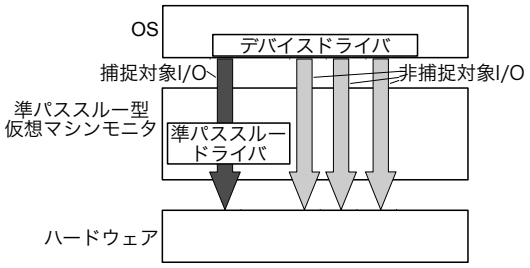

図 1 準パススルー型仮想マシンモニタ

3. 設計方針と想定環境

3.1 目的と方針

我々の研究では，高負荷な処理をおこなうOSをライブマイグレーションすることを目的とする．高負荷な処理をおこなうOSは，多くの計算機資源を必要とするため，一つのマシンを専有する．

本研究で提案する手法は，以下の2つの方針に沿って設計する．

(方針 1) デバイスを仮想化しない

デバイスの仮想化はオーバヘッドを生む．オーバヘッドを避けるために，デバイスの仮想化はおこなわない．

(方針 2) OSやファームウェアに依存しない

クラウドの事業者や顧客が利用できる環境を制限しないために，OSやファームウェアの依存を避ける．

3.2 想定環境

以下では，本研究で想定する環境を示す．

(前提 1) 移動元と移動先の物理マシンのハードウェア構成は同じ

データセンタでは，同じハードウェア構成の物理マシンを多数設置し，サービスを提供することが多い．このため，同じハードウェア構成のマシン間でのみ利用可能なライブマイグレーション手法も十分有用である．

(前提 2) 物理マシン1つに対して，動作するOSは1つのみ

本手法を利用する場面として，高負荷な処理をおこなう場面を想定する．このような状況では，ひとつのOSが物理マシン1つを専有すると考えられる．本手法は，このような状況において，利用可能なものを目

指す．

4. 設計

本章では，提案するライブマイグレーションの設計について述べる．

4.1 本手法の全体像

本手法の全体像を図 2 に示し，以下で説明する．

本手法では，準パススルー型仮想マシンモニタを用いる．仮想マシンモニタのレイヤでライブマイグレーションの機能を実現することにより，OS やファームウェアに依存しない手法とする．本手法において，準パススルー型仮想マシンモニタは I/O の監視とマイグレーションの処理のみをおこなう．通常時は，物理デバイスの状態を取得するために一部 I/O の監視のみをおこなう．I/O の監視の際，I/O の変換や I/O の結果の改変はおこなわない．OS は物理デバイスに直接アクセスし，準パススルー型仮想マシンモニタが I/O を改変することもないため，物理マシン環境とほぼ同じ環境である．

一方，マイグレーション時には，ハードウェア状態の取得や復元，リモートマシンとのハードウェア状態の送信や受信をおこなう．マイグレーションが終了すると，準パススルー型仮想マシンモニタは，I/O の監視のみをおこなう状態へ戻る．

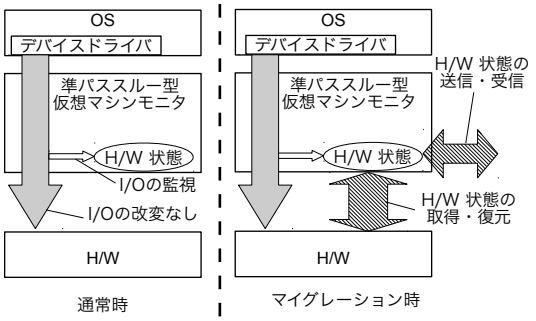

図 2 本手法の全体像

4.2 ライブマイグレーションの処理流れ

ライブマイグレーション処理の流れを図 3 に示し，以下で説明する．

まず，移動元マシンの仮想マシンモニタがハードウェアから状態を取得する．次に，移動元マシンの取得した状態を，移動先の仮想マシンモニタへ転送する．最後に，移動先の仮想マシンモニタが受け取った状態を復元する．これらの処理において，OS は一切介入しない．

ライブマイグレーションは大きく分けてメモリ, CPU, デバイスの 3 つのハードウェア状態を転送する．以降の節では，これらの状態のマイグレーションに関する設計を述べる．

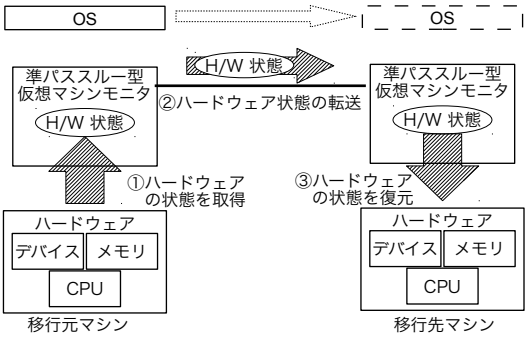

図 3 マイグレーションの流れ

4.3 メモリ

移動元マシンの仮想マシンモニタでメモリを読み込み，転送する．この時，移動元のマシンにおいて，データが置かれていたアドレスの情報も付与する．移動先のマシンの仮想マシンモニタは，受け取ったデータから，アドレス情報を取り出す．その後，メモリにデータを書き込む．メモリのマイグレーションについては Pre-copy [1]，Post-copy [8] [9] のアルゴリズムを用いることができる．

4.4 CPU

CPU の状態は仮想マシンモニタが管理するものと，CPU が管理するものがある．それぞれについて状態を転送し，移動先の仮想マシンモニタで復元する．

4.5 デバイス

本手法では，仮想マシンモニタはデバイスを仮想化せず，OS がデバイスへ直接アクセスしているため，仮想マシンモニタはデバイスの状態を保持しない．このため，仮想マシンモニタは，何らかの方法を用いて，物理デバイスの状態を取得する必要がある．

物理デバイスの状態を取得する方法としてまず考えられるのは，デバイスのレジスタから値を読み出す方法である．しかし，デバイスは一部の状態を書き込み専用のレジスタに保持するため，レジスタから値を読むだけでは取得できない状態がある．このため，レジスタから値を読むだけでは物理デバイスの状態を取得するには不十分である．

デバイスの書き込み専用レジスタは，デバイスを設定するために用意されたものなど，OS から値を書き込まれて初めて意味を成す．このため，書き込み専用レジスタが意味のある状態を持つためには，必ずソフトウェアによる書き込みがおこなわれている．

そこで，書き込み専用レジスタが保持するデバイスの情報を取得するため，OS が発行する I/O を監視，記録する．この方法によるデバイス状態のマイグレーションについて，図 4 に示し，以下で説明する．書き込み専用レジスタへの書き込みの監視は，OS 起動開始時からおこなう．通

常時，準パススルー型仮想マシンモニタは，OS からデバイスへ発行される I/O のうち，書き込み専用レジスタへの書き込み I/O のみを監視，記録する．マイグレーション時には，レジスタの情報に加え，記録した I/O 命令の情報も転送する．移動先マシンでは，レジスタ情報を書き込みに加え，移動元で記録した I/O を実際に発行し，デバイスの状態を復元する．

デバイスが書き換える状態については，I/O を監視することでは得られない．しかし，デバイスが書き換える状態のうち OS の動作に影響を与えるような状態は，OS が把握する必要があるため，読み出し可能なレジスタへ格納されると考えられる．このため，デバイスが書き換える状態の取得はレジスタから値を読み出すことで得られるもので十分である．

5. 実装

本章では，提案するライブマーグレーション手法の実装について述べる．まず，本手法で用いる準パススルー型仮想マシンモニタの BitVisor について述べる．その後，メモリ，CPU，物理デバイスのそれぞれについてマイグレーションの実装について述べる．最後に，プロトタイプの実装状況について述べる．

5.1 仮想マシンモニタ

今回の実装では，準パススルー型仮想マシンモニタである BitVisor[10] を用いる．BitVisor は OS の起動より前に起動し，デバイスへの I/O を監視する．

5.2 メモリ

移動元マシンでのメモリの読み込み，メモリ情報の転送，移動先マシンでのメモリ書き込みは全て BitVisor がおこなう．転送の際，メモリのデータが移動元の OS から見てどのアドレスにあったかの情報も付与する．移動先では，OS から見たアドレスが変わらないように BitVisor がマッピングし，メモリの情報を書き込む．メモリは，BitVisor が占有する領域を除いて，すべて転送する．

5.3 CPU

本実装では，CPU の仮想化支援機能を用いている．CPU の仮想化支援機能の例には，Intel 製 CPU に搭載されている Intel VT や AMD 製 CPU に搭載されている AMD-V が挙げられる．

ゲスト OS が使用する CPU は CPU の仮想化支援機能によって仮想化されており，状態の一部は CPU が保持している．例えば，Intel VT では，Virtual Machine Control Structure (VMCS) というデータ構造の中に，ゲスト OS が用いる仮想 CPU の状態を格納している部分がある．このため，CPU 状態のマイグレーションには，BitVisor が管理する状態と仮想化支援機能が管理する CPU の状態を転送する必要がある．仮想化支援機能が保持する状態は，専用のアセンブラ命令で読み書きできるため，これを用いて BitVisor が仮想 CPU の状態を読み出す．移動先では，BitVisor が専用のアセンブラ命令を実行し，受信した仮想 CPU の状態を VMCS に書き込む．

5.4 物理デバイス

5.4.1 I/O の監視による状態の取得

BitVisor は，特定の I/O をフックする機能がある．Port-mapped I/O のフックは，対象の I/O をポート単位で指定できる．これを用いて，デバイスの書き込み専用レジスタへの書き込み命令をフックする．デバイスによっては，同じポートへの書き込みが必ずしも同じレジスタへの書き込みとなるとは限らない．書き込み専用レジスタへの書き込みか否かは，書き込み先ポートに加え，書き込むデータの内容，直近の I/O 命令から判断する必要がある．

図 5 に割り込みコントローラの Programmable Interrupt Controller(PIC) への I/O の例を示す．PIC の仕様として，ポート 0x20 への書き込み I/O のうち，書き込むデータの 4 ビット目がセットされているものは，PIC の設定を開始することを通知する I/O であることが決まっている．また，この書き込み I/O の後，0x21 ポートへの連続する 3 回の書き込みは，PIC の設定するために発行される書き込み専用レジスタへの書き込み I/O であることが決まっている．

書き込み専用レジスタへの書き込み I/O は，BitVisor 内で記録する．記録の際，I/O の順序も保持する．これは，PIC のように，I/O の順序関係が I/O の意味に関係するためである．

5.4.2 I/O の復元

移動元で取得した I/O 命令の情報に沿って，BitVisor が書き込み専用レジスタへの書き込み I/O を発行する．この際，I/O を発行する順序も移動元と同一となるようにする．これによってデバイスの状態を復元する．

5.5 プロトタイプの実装状況

現在，マイグレーション可能なデバイスは PIT と PIC のみである．また，単一のコアを使用する OS のみマイグレーション可能であり，複数コアを使用する OS をマイグレーションすることはできない．メモリのマイグレーションは，現在，全て stop-and-copy でおこなっている．このため，プロトタイプでは，メモリのマイグレーション中に OS は完全に停止する．

6. 実験

本章では，今回実装したプロトタイプについて，通常時にシステムの性能に与える影響とマイグレーションに要す

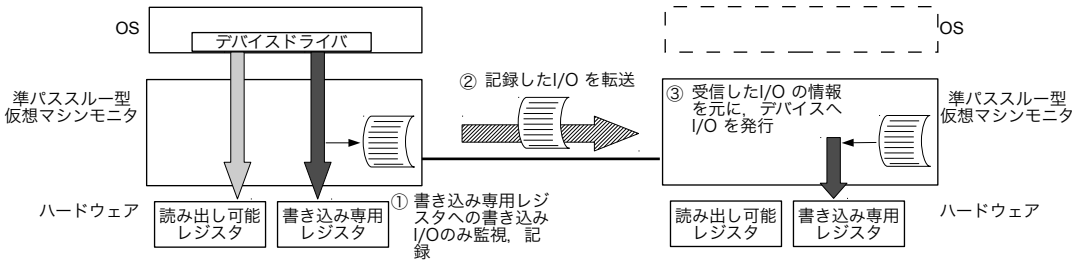

図 4 デバイス状態のマイグレーション

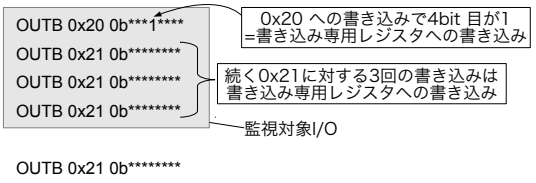

図 5 PIC への I/O の例

る時間について示す．

6.1 実験環境

実験環境を表 1 に示す通りである．

表 1 実験環境

CPU	Core 2 Duo E8500(3.16GHz)
メモリ	2GB
OS	Debian 7.2
カーネル	Linux kernel 3.2.0

今回，実験でマイグレーションする OS の Linux カーネルにはいくつか機能を制限している．まず，ディスクを利用しないよう制限している．これはディスクのマイグレーションは未実装であるためである．ファイルシステムは全てメモリ上に存在する状態でマイグレーションをおこなう．また，利用するデバイスを制限している．割り込みコントローラは現在マルチコアプロセッサマシンで主流の Advanced Programmable Interrupt Controller(APIC) ではなく，PIC を用いる．タイマは Programmable Interval Timer(PIT) を用いる．

これらの制限は，Linux カーネルのビルドオプションと init スクリプトの変更のみでおこない，カーネルのソースコードを改変していない．

6.2 通常時の性能への影響

今回実装したプロトタイプが，通常時の OS の性能に対してどの程度影響を与えるか調べるため，UnixBench[11] を用いたベンチマークテストをおこなった．結果のグラフを図 6 に示し，以下で説明する．ベンチマークテストはベアメタル，I/O を監視しない BitVisor の動作時，I/O を監視する BitVisor の動作時の 3 つの場合についておこなった．グラフには，UnixBench のうち，総合的な性能を示す System Benchmarks Index Score の値を示している．値は index と呼ばれるものである．これは，基準となる計算機に比べて何倍の性能があるかを示す指標であり，値が大きいほど高い性能であることを示す．グラフは，それぞれ 3 回の測定結果の平均値を示している．

結果から，I/O を監視しない BitVisor を動作させると，性能はベアメタルに比べて 27%ほど低下する．また，I/O を監視する BitVisor を動作させると，性能はベアメタルに比べて 30%ほど低下する．また，BitVisor が I/O を監視しない場合と監視する場合を比べると，I/O を監視することによって，監視しないと比べて 5%ほどの性能が低下している．

このことから，プロトタイプの実装における性能劣化の原因は，BitVisor を動作させるためのものがほとんどであり，I/O 監視の有無が性能に与える影響は小さいといえる．

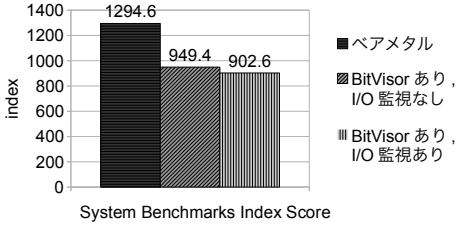

図 6 UnixBench の結果 (3 回の測定の平均値)

6.3 マイグレーションに要する時間

マイグレーションに要する時間とその内訳を表 2 に示し，以下で説明する．今回の実装では，メモリの転送を全て stop-and-copy でおこなっているため，マイグレーションに要する時間と OS のダウンタイムは等しい．表ではマイグレーションに要する時間のうち，メモリ，CPU，デバイスの状態それぞれをコピーする時間を内訳として示している．それぞれの内訳について，測定開始は移動元マシンで状態の取得を開始した時点，終了は移動先マシンで状態の復元が完了した時点としている．また，表のその他の行には移動元と移動先の間での接続の確立や，測定のために必要な処理の時間が含まれている．表中の値は 4 回の測定

結果の平均値を示している．

表に示す結果から，マイグレーションに要する時間のほとんどがメモリ状態のマイグレーション時間であることがわかる．また，デバイス状態のマイグレーションを含め，メモリのマイグレーション以外は合計しても数十ミリ秒オーダであることがわかる．

このことから，本手法でも Pre-copy や Post-copy によるメモリマイグレーションを実装し，ダウンタイム時間を減らすことによって，ダウンタイムが非常に短いライブマイグレーションを実現することが可能であると言える．

表 2 マイグレーションに要する時間の内訳 (4 回の測定の平均値)

内訳	時間 (ms)
メモリコピー	251835.6
CPU 状態のコピー	12.4
デバイス 状態のコピー	11.2
その他	42.7
合計	251878.2

7. 関連研究

以下では，物理マシン上の OS ライブマイグレーションに関する研究を示す．

CompSC[3] は OS からデバイスへ直接アクセスする環境での OS ライブマイグレーションをおこなう研究である．CompSC も，本研究と同様に，I/O を監視することで物理デバイスの状態を把握している．また，デバイス特有の情報はハイパーバイザに持たせず，全て OS のドライバに持たせる方針をとっている．このために，デバイス状態の復元機構を OS のドライバに改変を加えて実現している．本研究は，ゲスト OS に依存しない手法とするため，マイグレーションに必要な機構はすべて仮想マシンの中で実現する点で異なる．

Kadav らの研究 [4] では，shadow driver と呼ぶドライバをゲスト OS 内に組み込むことで，直接アクセスするデバイスのマイグレーションを可能にする手法を提案している．この研究では，異なるハードウェア間のマイグレーションも可能であるものの，OS に shadow driver を組み込むため OS へ依存する手法となっている．本研究では，異なる物理デバイス間のマイグレーションには対応していないものの，OS に依存しない手法である点が異なる．

Jeffrey らの研究 [5] では Network Interface Card (NIC) のファームウェアを改変することによって，クラウド基盤を構築する手法を提案している．この手法は，NIC のファームウェアに依存しており，クラウド事業者が利用可能な NIC を制限してしまう．一方，我々の手法は，ファームウェアの改変は不要のため，クラウド事業者に対して利用可能な NIC を制限することはない．

8. 今後の課題

以下では，本研究の今後の課題について述べる．

(1) PIC，PIT 以外のデバイスへの対応とその評価
今回の実装では，PIC や PIT といった比較的単純なデバイスに対して，本手法を適応した．しかし，実際のクラウド上では，PIC や PIT 以外のデバイスも用いる．特に，NIC はクラウドにおいて必要不可欠なものである．そこで，NIC をはじめとする，様々なデバイスへ本手法を適用し，その影響を評価することが課題となる．
様々なデバイスへ本手法を適用する際，割り込みなどを考慮し，マイグレーション開始のタイミングを慎重に決める必要がものと考えられる．

(2) 記録する I/O の選別
現在，I/O の監視は OS 起動時からおこなっている．また，監視対象の I/O を記録すると，その記録は OS が終了するまで破棄されることはない．このため，起動時から対象となる I/O をすべて記録すると，I/O に関する情報が単調増加する．これは，マイグレーションに要する時間の増加や，データ領域の圧迫を招く．また，I/O によって，最新の I/O のみが必要で，古い I/O の情報は不要なものもある．このことについて検討し，記録しておく I/O の情報を削減することが課題となる．

(3) マイグレーション時のダウンタイムの削減
今回の実装では，メモリのマイグレーションは全て stop-and-copy となっている．このため，大きなダウンタイムが発生した．既存のライブマイグレーション手法である Pre-copy[1] や Post-copy[8] [9] は設計上，本手法でも用いることができる．これらのアルゴリズムを用いて，ダウンタイムを削減することは実際にクラウド上で本手法を用いる上での課題となる．

(4) マルチコア環境への対応と評価
今回のおこなった実装は，マルチコア環境に対応していない．しかし，マルチコア環境は広く普及しており，クラウドにおいてもマルチコアマシンを用いてサービスを提供している．このため，本手法をクラウド上で利用するためには，マルチコア環境への対応と評価をおこなうことが課題となる．対応にあたり，移動元のコアと移動先のコアの ID を合わせる処理をおこなう必要がある．

(5) 脱仮想化と再仮想化の利用
本研究では，準パススルー型仮想マシンモニタを用いて物理マシン環境上の OS のマイグレーションを実現した．OS が動作している間，常に仮想マシンモニタが動作しており，これによるオーバヘッドが生じる．これを排除するため，通常時は仮想マシンモニタを動

作させず，マイグレーション時のみ仮想マシンモニタを動作させる必要がある．これを実現するためには，マイグレーション前に再仮想化し，マイグレーション後に脱仮想化することが課題である．

9. まとめ

本論文では，物理マシン上での OS ライブマイグレーションを実現する手法を提案した．準パススルー型仮想マシンモニタを用い，OS やファームウェアに依存せずにライブマイグレーションを実現する手法とした．また，OS が発行する I/O を監視，記録することで仮想化していないデバイスの状態を取得し，移動先のマシンで実際に I/O を発行することで，デバイス状態の復元した．実際に単純なデバイスに対して本手法を実装し，実験をおこない結果を示した．

参考文献

[1] Clark, Christopher., Fraser, Keir., Hand, Steven., Hansen, Jacob Gorm., Jul, Eric., Limpach, Christian., Pratt, Ian., Warfield, Andrew.: Live migration of virtual machines, *Proc. 2nd conference on Symposium on Networked Systems Design and Implementation*(*NSDI 2005*), pp. 273–286 (2005)

[2] Huang, Wei., Liu, Jiuxing., Abali, Bulent., Panda, Dhabaleswar K.: A case for high performance computing with virtual machines, *Proc. 20th annual international conference on Supercomputing*(*ICS 2006*), pp.125–134 , (2006).

[3] Pan, Zhenhao., Dong, Yaozu., Chen, Yu., Zhang, Lei., Zhang, Zhijiao.: CompSC: live migration with pass-through devices, *SIGPLAN Not.*, Vol.47, pp. 109–120 (2012).

[4] Kadav, Asim., Swift, Michael M.: Live migration of direct-access devices, *SIGOPS Oper. Syst. Rev.*, Vol.43, pp. 95–104 (2012).

[5] Jeffrey, C., Mogul., Jayaram, Mudigonda., Jose, Renato, Santos., Yoshio, Turner.: The NIC Is the Hypervisor: Bare-Metal Guests in IaaS Clouds, (*HotOS 2013*), (2013)

[6] Barham, Paul., Dragovic, Boris., Fraser, Keir., Hand, Steven., Harris, Tim., Ho, Alex., Neugebauer, Rolf., Pratt, Ian., Warfield, Andrew.: Xen and the art of virtualization, *Proc. nineteenth ACM symposium on Operating systems principles*(*SOSP 2003*), pp. 164–177 , (2003)

[7] Liu, Jiuxing., Huang, Wei., Abali, Bulent., Panda, Dhabaleswar K.: High performance VMM-bypass I/O in virtual machines, *Proc. annual conference on USENIX '06 Annual Technical Conference*(*ATEC 2006*), pp.3–3 , (2006)

[8] Hines, Michael R., Deshpande, Umesh., Gopalan, Kartik.: Post-copy live migration of virtual machines, *SIGOPS Oper. Syst. Rev.*, Vol43, No3, pp. 14–26 (2009)

[9] Hines, Michael R., Gopalan, Kartik.: Post-copy based live virtual machine migration using adaptive pre-paging and dynamic self-ballooning, *Porc. 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*(*VEE 2009*), pp. 51–60, (2009)

[10] Shinagawa, T., Eiraku, H., Tanimoto, K., Omote, K., Hasegawa, S., Horie, T., Hirano, M., Kourai, K., Oyama, Y., Kawai, E., Kono, K., Chiba, S., Shinjo, Y., Kato, K.: BitVisor: A Thin Hypervisor for Enforcing I/O Device Security, *Proc. 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments* (*VEE 2009*), pp. 121–130 (2009).

[11] byte-unixbench - A Unix benchmark suite - Google Project Hosting(online), available from 〈https://code.google.com/p/byte-unixbench/〉 (accessed 2013-11-07)