Vol.2010-ARC-190 No.19
2010/8/4

直接 I/O 環境下の仮想マシン移動を実現する PCI Express スイッチ

佐藤充[†] 三吉貴史[†] 岩松昇[†]
稲垣淳一[††] 矢崎昌朋[†] 堀江健志[†]

クラウドシステムの普及により，仮想化の重要性が増してきている．現在の仮想化の課題として I/O 性能および多種 I/O サポートがあり，仮想化環境で I/O 高速化・多種サポートを実現できる直接 I/O と呼ばれる技術が着目されてきている．本論文では，直接 I/O を実現し，I/O をハードウェアレベルで仮想化する技術として PCI Express (PCIe)をベースとした I/O 仮想化技術を取り上げ，仮想化環境で必要とされる仮想マシン移動をサポートするための技術について議論する．
我々は仮想マシン移動をサポートするための技術としていくつかの可能性を検討している．それらは I/O MMU を拡張する方式，PCIe スイッチ内に書き込み記録テーブルを保持する方式，PCIe スイッチで書き込みをマルチキャストする方式である．
今回我々は，PCIe スイッチ内に書き込み記録テーブルを保持する方式を実装し，効果の見積もりを行った．その結果，直接 I/O を用いた仮想マシンでも，見た目の停止時間がわずかに増加（＋5ms）するだけで移動できることを確認した．

Live Migration for VMs with Direct I/O using PCI Express Switch

Mitsuru Sato[†], Takashi Miyoshi[†], Noboru Iwamatsu[†],
Junichi Inagaki[††], Masatomo Yasaki[†], and Takeshi Horie[†]

In recent years, cloud computing becomes popular and importance of virtual machine (VM) system increases. Current VM system has problems in the performance of I/O. So the “Direct I/O” mechanism will become key technology of future virtual machine systems to improve I/O performance of virtual machines.
In this paper, we propose some methods of live migration for VMs with Direct I/O. In Direct I/O environment, there are some difficulties to move VMs. To solve such difficulties, we use PCI Express (PCIe) switch and hardware mechanism in PCIe switch.
We propose three mechanisms for migrate VMs: extension of I/O MMU, page listing in PCIe switch, and multicasting in PCIe switch.
We implemented the second method for testing and estimated additional stop-period of migrated VM at 5ms.

1. はじめに

近年，クラウドコンピューティングが注目を集めている．クラウドコンピューティングでは，計算リソースをユーザの手元からクラウド環境に移し，ユーザに対し，使用できる計算リソース，利用できる場所などに関する高い柔軟性を与えることが可能となる．このようなクラウド環境を支える計算システムインフラストラクチャでは，継続的なサービス提供，柔軟な資源割り当て，効率的な運用などの機能が必要とされてくる．

これらの機能を実現するため，クラウド環境に対応した計算機センターでは仮想化システムが広く用いられている．仮想化システムは，物理的な計算機内部に仮想的な計算機イメージを用意し，アプリケーション（サービス）を仮想マシン上で動作させる仕組みである．このような仮想化システムを用いることで，先に挙げたクラウド環境に要求される機能を実現することができる．

さらに最近では，これまではクラウド環境には不向きであるとされていた領域，たとえば高い I/O 性能を必要とするデータベースアプリケーション，GPU によるアクセラレーションを必要とするメディアアプリケーションなどもクラウド環境へ移行し，その高い柔軟性を活用しようという動きが出てきている．

従来，これら I/O に対する要求が厳しいシステムは，仮想化環境では取り扱いの難しいものとされてきた．それは，仮想マシンの I/O 性能が高くない，あるいは GPU などの I/O デバイスは仮想化環境では取り扱いが困難という理由からである．

本論文では，それら I/O 要求が厳しいシステムも仮想化環境に取り込み，クラウド環境での利用を可能とする仕組みを提案する．本論文で提案する仕組みを用いると，高い I/O 性能，GPU などの I/O デバイスを必要とするシステムでも，従来と同様の仮想化環境の利点を得ることができ，クラウドサービスの利用領域を拡大することが可能となる．

2. 仮想マシンと直接 I/O

2.1 仮想マシンの構成

仮想マシンシステムは，仮想的な計算機イメージである仮想マシンと，それらを管理するハイパーバイザから構成される．仮想マシンシステムを用いると，物理的には単一の計算機上に，複数の仮想マシンを同時に構成することができる．ハイパーバイザは物理的なリソース（CPU，メモリ，I/O）を時間・空間的に分割し，各仮想マシン

[†] 株式会社富士通研究所
Fujitsu Laboratories Limited
[††] 富士通株式会社
Fujitsu Limited

に提供するという役割を担う．

2.2 エミュレーション I/O

従来，仮想マシンシステムにおける I/O は，ハイパーバイザによるエミュレーションが主に用いられてきた．ハイパーバイザは仮想マシンに対してソフトウェアで構成されたエミュレーション I/O を提供する．仮想マシンからは，このエミュレーション I/O があたかも物理的な I/O デバイスであるかのように見える．ハイパーバイザは，各仮想マシンからのアクセスをソフトウェアで制御し，物理的な I/O デバイスとのやりとりを行う．たとえば，複数の仮想マシンが一つの物理 I/O を共有する場合には，ハイパーバイザが各仮想マシンからの I/O 要求をいったん受け取り，スケジューリングした後物理的な I/O デバイスに要求を発行する．これにより，柔軟な I/O 資源割り当てが可能となっている．

一方で，エミュレーション I/O には，以下に述べる課題がある．

(1) I/O 性能

エミュレーション I/O では，仮想マシンからの I/O 要求を一度ハイパーバイザが受け取り，その後に物理 I/O デバイスに I/O 要求を発行する．そのため，I/O アクセスのたびにハイパーバイザでのソフトウェア処理によるオーバーヘッドが発生する．たとえば，仮想マシンのメモリ空間から I/O に対してデータ転送を行おうとする場合，一度データをハイパーバイザ空間へコピーし，その後にデータを I/O デバイスへ転送することとなる．そのため常にデータのコピーオーバーヘッドが生じ，I/O 性能が十分発揮できない．

(2) 特殊デバイスへの対応

エミュレーション I/O では，ハイパーバイザ空間に仮想的なデバイスをソフトウェア的に実現することが必要である．ところが，すべての I/O デバイスに対してエミュレーションモデルを構築することは困難である．特に近年では様々な高機能デバイス，たとえばアクセラレータとしての GPU や高速記憶装置としての PCIe 接続 Flash ROM などが用いられているが，これら高機能デバイスへの対応が難しい．

これらの課題を解決する手段として，近年，直接 I/O と呼ばれる手段が注目されている．以下では直接 I/O を解説し，直接 I/O を用いた仮想マシンシステムで解決すべき課題について議論する．

2.3 直接 I/O

直接 I/O は，仮想マシンから直接物理的な I/O デバイスを操作する手段である．エミュレーション I/O でも，デバイスと仮想マシンを一対一で対応づけることにより物理 I/O デバイスを仮想マシンに占有させる手段は可能であったが，直接 I/O はさらにハイパーバイザの関与をなくすことで仮想マシンが直接 I/O デバイスを操作する手段を提供する．

エミュレーション I/O と直接 I/O の動作比較を図 1 に示す．

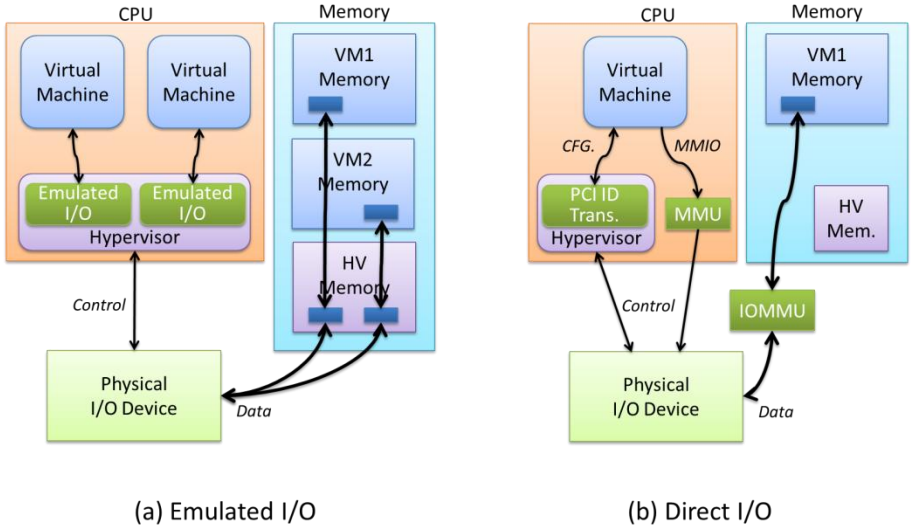

図 1 エミュレーション I/O と直接 I/O

ハイパーバイザは初期化時に，仮想マシンと物理 I/O デバイスとの対応をとり，I/O MMU を設定する．I/O MMU は I/O デバイスからのメモリアクセスに関してアドレス変換を行うユニットで，たとえば Intel では VT-d[1]，AMD では IOMMU[2]という形で実装されている．

直接 I/O 環境下では，CPU と I/O デバイスとの間でのデータ転送は，以下の 3 つのアドレス変換手段を用いる．

1. ハイパーバイザによるソフトウェア変換
   CPU から I/O デバイスのコンフィグレーション空間へのアクセスは，ハイパーバイザによって変換され，対象 I/O デバイスへのアクセスとなる．このアクセスはエミュレーション I/O の場合と変わらない．
2. CPU のアドレス変換
   CPU から MMIO (Memory Mapped I/O)空間へのアクセスは，CPU のメモリアドレス変換機構を使って変換され，直接 I/O デバイスの MMIO 空間へ書き込まれる．
3. I/O MMU のアドレス変換
   直接 I/O を最も特徴付けているのがこのアドレス変換である．I/O デバイスとのやりとりは，CPU から発行するものだけでなく，I/O デバイスからの DMA アクセスがある．この DMA アクセスは，一般には物理マシンの物理メモリアドレ

スをベースとして行われるため，そのままでは仮想マシンから直接扱うことはできない．これを仮想マシンから扱えるようにするのが I/O MMU によるアドレス変換である．I/O MMU では，イニシエーターとなるデバイス ID および対象となるアドレスをキーとして，物理アドレスを算出する．これにより，I/O デバイスは仮想マシン上のアドレスを利用しつつ，ハイパーバイザを経由せず直接物理メモリへアクセスすることが可能となる．

これらアドレス変換の技術を用いることで，通常の I/O アクセスはハイパーバイザを経由せず行うことが可能となる．

直接 I/O を用いると，エミュレーション I/O で問題となっていた 2 つの課題，すなわち I/O 性能および特殊デバイスへの対応が解決される．直接 I/O では，エミュレーション I/O とは違いハイパーバイザでのデータコピーが発生せず，ソフトウェアのオーバーヘッドを大幅に削減することができる．今日では，高い I/O 帯域を必要とするサービス，たとえばデータベースサーバやネットワークサーバなどが存在しており，これらサービスを仮想マシン環境で実行する際に，エミュレーション I/O では大きな性能劣化が発生する．直接 I/O はこれら高い I/O 帯域を必要とするサービスに対しても，仮想マシンでの運用を可能とする．

また，エミュレーション I/O 用のモデルを作成困難な特殊デバイス，たとえば GPU や Flash ROM，レガシーPCI デバイスなども，仮想マシンから直接利用することができ，仮想マシンの応用が広がることが期待される．

一方で，直接 I/O では仮想マシンが I/O デバイスを占有してしまうため，エミュレーション I/O で可能となっていた I/O デバイスの共有はできなくなる．しかし直接 I/O で I/O デバイスの共有を実現するための SR-IOV (Single-Root I/O Virtualization)[3]と呼ばれる規格が PCI-SIG で策定されており，これに準拠した I/O デバイスであれば直接 I/O を用いつつ I/O デバイスを共有することも可能となる．

3. 直接 I/O と VM 移動

3.1 直接 I/O の課題

直接 I/O では，I/O MMU を用いて，I/O デバイスから直接仮想マシンの持つメモリ空間へデータを転送することが可能となっている．これにより，ハイパーバイザを経由せず仮想マシンと I/O デバイスがデータのやりとりをする．

この仕組みは仮想マシンの I/O 性能を高めるのに寄与するが，逆にハイパーバイザが仮想マシンと I/O デバイスとのやりとりに関与できないことにより，Live Migration ができなくなるという問題が生じる．以下では Live Migration がどのように実現されているかを説明し，それが直接 I/O 環境ではなぜ困難なのかを説明する．

3.2 Live Migration の仕組み

Live Migration は仮想マシンが稼働したままで物理ノード間の移動を可能にする技術である．Live Migration を用いると，仮想マシンシステムの大きな特徴であるメンテナンス性の向上や，柔軟な動的資源割り当て，仮想マシンの動的負荷分散などが可能となる．

一般的な Live Migration の手順を図 2 に示す．

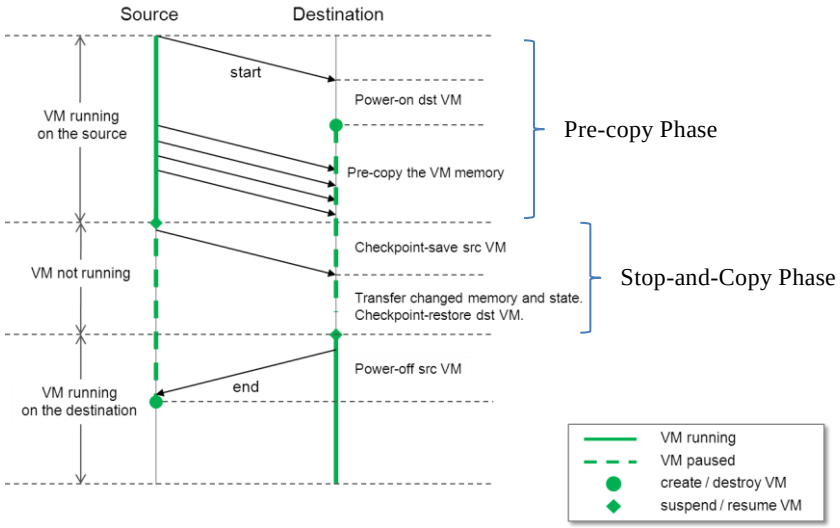

図 2　Live Migration タイミングチャート

Live Migration では，転送先に空の仮想マシンを準備し，転送元で仮想マシンを稼働させたまま仮想マシンのメモリ空間をコピーする（Pre-copy フェーズ）．仮想マシンは稼働し続けるので，転送元の仮想マシンのメモリ空間は，転送中にもアップデートされ続ける．そのため，転送先へメモリコピーが終了したときには，メモリ内容は一致していない．そこで次に，転送中にアップデートされたメモリをもう一度コピーする．転送中にアップデートされたかどうかの判断は，ページテーブルの Modified ビットを用いる．これにより，転送元と転送先のメモリ一致をはかる．さらにこの転送中でアップデートされた部分があれば，再度アップデートされた分の転送を行い，何度かこれを繰り返した後，Stop-and-Copy のフェーズに入る．

Stop-and-Copy フェーズでは，転送元の仮想マシンを一度停止させ，最終的なメモリ内容，CPU 状態，I/O 状態などを転送先にコピーする．この際，これまでの Pre-copy フェーズでほとんどのメモリはコピーされているので，Stop-and-Copy フェーズで転送

すべきメモリ量は少量ですむ．従って，Stop-and-Copy フェーズにかかる時間はごくわずかである．

Stop-and-Copy フェーズが終了すると，転送先で新しい仮想マシンを起動する．仮想マシンが停止し，仮想マシン上のサービスが中断されるのはこの Stop-and-Copy フェーズのみであり，非常に小さな値（一般には数 10～数 100ms）である．

3.3 直接 I/O 環境下での Live Migration

2.3 節で述べたように，直接 I/O 環境下では仮想マシンと I/O デバイスは直接データのやりとりを行う．そのため，以下に述べる問題が発生する．

1. 転送元と転送先で I/O デバイスが一致しない
   Live Migration では，仮想マシンが別の物理マシンへ移動する．移動元の物理マシンと移動先の物理マシンが同一構成である保証はなく，移動元で利用していた I/O デバイスが移動先では存在しない可能性がある．さらに，移動元と移動先で同じ種類の I/O デバイスがあったとしても，内部状態は一致しておらず，Live Migration 後継続して使うことはできない．
2. I/O デバイスから書き込まれたメモリ領域を特定することができない
   Live Migration では仮想マシン動作中にメモリコピーを行い，後からアップデートされた領域のみを転送することで，見かけの停止時間を短くするという技術を用いている．しかし直接 I/O では，I/O デバイスからの書き込みは I/O MMU を介してハードウェアにより行われる．そのため，CPU からどのメモリ領域が I/O デバイスによって書き換えられたのか検出することができない．
3. ハイパーバイザから I/O デバイスの停止操作ができない
   I/O デバイスの制御はすべて仮想マシン自身が行っているため，ハイパーバイザから I/O デバイスを停止したり再開したりすることができない．すなわち，上記 Live Migration の Stop-and-Copy のフェーズでも I/O デバイスは動き続けてしまい，デバイス状態やメモリ内容が転送元と転送先で一致しない場合がある．

これら問題点のため，現状では直接 I/O を用いた仮想マシンは Live Migration ができないという制限が生じている．

3.4 直接 I/O と仮想マシン移動を実現する手法

これら問題を解決し，直接 I/O 環境下の Live Migration を実現する手法は，これまでいくつか提案されてきている．それら手法は，大きく分けて以下の 3 つに分けられる．

1. エミュレーション I/O との組み合わせによる解決手法
   直接 I/O とエミュレーション I/O の両方を利用し，Live Migration の期間はエミュレーション I/O で動作させるという手法が提案されている（文献 5)）．ネットワークデバイスなど，ボンディングが可能な一部のデバイスには有効だが，一般のデバイスには利用できない．
2. ソフトウェアによる解決手法
   直接 I/O とエミュレーション I/O の中間的な手法を用いて，性能を保ったまま仮想マシン移動を可能にする手法が考えられている．たとえばネットワークコントローラとして，汎用かつデータを直接 I/O デバイスとやりとりできるインタフェースを定義する．このインタフェースを用いて，物理 I/O デバイスの上に wrapper をかぶせることで仮想マシン移動に対応しつつ，直接データ転送を可能にすることができる．しかしこの手法では，デバイス内部に状態を持つ GPU や Flash ROM などのデバイスには対応できない．
3. ハードウェアによる解決手法
   専用のハードウェアを用いて解決する手段はいくつか提案されている．たとえば，PCI-SIG で定義されている MR-IOV (Multi-Root I/O Virtualization)[4]に準拠した I/O デバイスを用いると，I/O デバイスの停止・起動がハイパーバイザから可能となる．

これら手法のうち，我々は 3 のハードウェアによる解決法に属する手段を用いて，直接 I/O と仮想マシン移動を実現する手段を提案する．

4. PCI Express スイッチによる I/O 接続

4.1 Multi-Root PCI Express スイッチ

PCI Express (PCIe)スイッチは，多数の PCIe 間を接続するためのスイッチである．PCIe は PCI と同様，単純な木構造をとる．PCIe の木構造では，ルートは一つ（Root Complex と呼ばれる）であり，一般の PC では CPU（あるいはチップセット）がここに相当する．

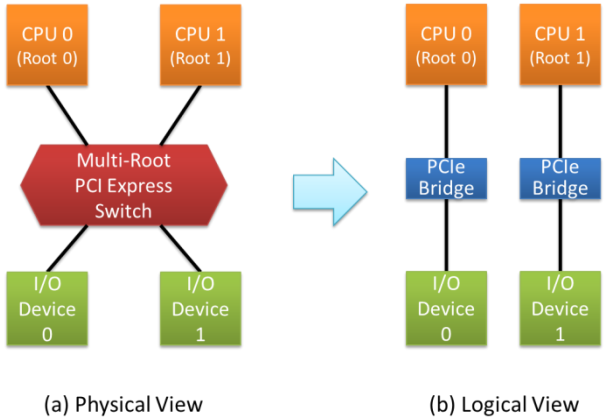

図 3 Multi-Root PCIe Switch

Multi-Root PCIe スイッチは，このルートを複数持てるようにした PCIe スイッチである．Multi-Root PCIe スイッチを用いた I/O 構造を図 3 に示す．

Multi-Root PCIe スイッチは，内部に複数の木構造を持ち，ルートに相当するものが複数存在することを許す．従って，たとえば図 3 のように CPU が 2 つ，I/O デバイスが 2 つという構成も可能となる．図 3 の場合は CPU 0 に相当するルート配下に I/O デバイス 0 が所属し，これで一つの木構造を構成している．同様に CPU 1 に相当するルートと I/O デバイス 1 がもう一つの木構造を構成する．Multi-Root PCIe スイッチは，単一スイッチ内部にこのような異なる木構造を同居させることができるものである．

このような仕組みは，PCI-SIG において I/O 仮想化の一部，MR-IOV (Multi-Root I/O Virtualization)[4]として標準化されている．しかし MR-IOV はここで述べた Multi-Root PCIe スイッチを包含したさらに上位の標準規格であり，ここで述べることは必ずしも MR-IOV 全体を必要としないことには注意が必要である．

4.2 PCIe スイッチを用いた I/O デバイスの共有

Multi-Root PCIe スイッチを用いると，図 4 のように複数の CPU 間で一つの I/O デバイスを共有することが可能となる．

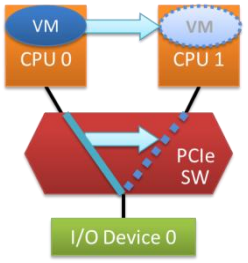

図 4 I/O デバイスの共有

今，図 4 において CPU 0 上で動作していた仮想マシンが CPU 1 へ移動することを考える．この仮想マシンが I/O デバイス 0 を直接 I/O を用いてアクセスしていたとすると，仮想マシンが CPU 1 に移動した後は異なる PCI バスツリーに移動してしまうこととなるので，I/O デバイス 0 はアクセスできなくなる．しかし，ここで PCIe スイッチが I/O デバイスの切り替えを行い，I/O デバイスを CPU1 の属する PCI バスツリーへ移動することができれば，仮想マシンは CPU 1 へ移動後も同じ I/O デバイス 0 を利用し続けることが可能となる．

このように，Multi-Root PCIe スイッチを用いることで，同じ PCIe スイッチに接続している CPU 間を仮想マシンが移動しても，I/O デバイスを利用し続けることが可能となり，3.3 節で挙げた問題点 1 を解決することができる．

4.3 PCIe スイッチを用いた I/O デバイスの一時停止

PCIe スイッチは I/O デバイスとのインタフェースとなっているため，PCIe スイッチ部で I/O デバイスからのトラフィックを一時止めることができる．これは単に PCIe スイッチ内部にパケットバッファを設け，そこに一時的にパケットをバッファリングすることで実現できる．これにより，I/O デバイスが動作中でも I/O デバイスからの書き込みデータを一時的に止め，仮想マシンが移動する間メモリがアップデートされるのを防ぐことが可能となる．

このような機能を PCIe スイッチに備えることで，3.3 節で挙げた問題点 3 を解決することができる．

5. I/O デバイスからの書き込み検出手法

前章では，Multi-Root 対応の PCIe スイッチを用いることで，直接 I/O 環境下の仮想マシンを移動する際に問題となる 3 点のうち 2 点までは解決できることを示した．

本章では残りの 1 点, Pre-copy フェーズ中に I/O デバイスからアップデートされたメモリ領域を検出できないという問題を解決するための 3 つの手段を提示し，それぞれの特徴，課題をまとめる．

5.1 I/O MMU 拡張

エミュレーション I/O では，Pre-copy フェーズ中に CPU からアップデートされた領域はページテーブルの Modified ビットを参照することで検出することができた．これと同様のことを I/O MMU で行えば，同様の検出が可能である．すなわち，I/O デバイスから書き込みアクセスが発生し，I/O MMU でアドレス変換をした際に，I/O MMU のテーブル内に I/O デバイスからの書き込みが行われたフラグを立てるような仕組みができれば，Pre-copy 終了時にフラグをチェックすることで I/O デバイスからアップデートされた領域を知ることが可能となる．

現在，各ベンダが提供している I/O MMU にはこのような機能は存在しないが，これを拡張し Modified ビットあるいはそれ相当のフラグを用意することで，エミュレーション I/O の場合と同様の処理が可能となる．

しかしこの手法は，I/O MMU の実装に依存する方法であり，PCIe の Root Complex あるいはメモリコントローラ, すなわちチップセットや CPU を提供するベンダ次第である．従って一般に利用できる手段とは言い難い．

5.2 Dirty Page List

前節で述べた I/O MMU 拡張と同様のことを，I/O MMU 外部で行う方法である．PCI バスを監視し，I/O デバイスから書き込みアクセスが発生した際，書き込み元の I/O デバイスの ID，ページ番号を記憶しておくテーブルを保持するデバイスを別途用意する．

Live Migration の際，Pre-copy が終了した後，このテーブルを参照することで，どの

ページに対して I/O デバイスから書き込みが発生したかを知ることができる．テーブルは I/O デバイスと Root Complex（メモリコントローラ）の間であればどこに配置してもよく，たとえば PCIe スイッチの内部に配置するということも考えられる．

この手法の問題点は，記憶しておくべきページの量である．Pre-copy の間，I/O デバイスがアップデートするページのリストをすべて記憶するためのテーブル容量が十分でなければ，Pre-copy を最初からやり直さなくてはならず，仮想マシン移動に多大な時間を費やすことになる．

5.3 Packet Multicast

I/O デバイスからのアップデートを検出せず，仮想マシンの移動を完了させる方法も考えられる．マルチキャストを使った手法は，I/O デバイスからの書き込みを，仮想マシンの移動元と移動先の双方に同時に行い，移動元と移動先での I/O デバイスによるアップデートを同期させる手法である．PCIe スイッチ内で I/O デバイスからの書き込みをマルチキャストし，異なる 2 つの Root Complex に同時に送信することで実現できる．

この手法は，スイッチ内部のリソースを必要とせず，ハードウェア的には比較的簡単な構造で実現することができる．しかし，I/O デバイスを移動元と移動先の双方で同時に認識しなくてはならない点や，仮想マシン移動の際，あらかじめ移動してくる仮想マシンのメモリ領域をすべて準備しておかなくてはならない点など，いくつかソフトウェア的にこれまでの枠組みとは違った実装が必要とされる．

6. 実験環境

前章で述べたように，直接 I/O と仮想マシン移動を両立させる手法としてはいくつかの手段が考えられる．我々はこれら手段のうち，1. 現状の CPU には手を加えない，2. なるべく現状のハイパーバイザの仮想マシン移動の仕組みに沿う，という観点から，5.2 節で述べた Dirty Page List 方式を選択し，実装を行った．本章では，我々の実装環境について述べる．

6.1 PCI Express Switch: SIVA

我々は独自の PCIe スイッチとして SIVA (Switch for I/O Virtualization Architecture)を開発している．

SIVA は FPGA で構成された PCIe スイッチで，独自の機能の自由に追加することができる I/O 仮想化試験用のプラットフォームである．SIVA は PCIe 拡張ボードの形式で実装され，PCIe コネクタ，あるいは拡張 PCIe ケーブルという，複数のホスト接続ポートを持っている．また，SIVA ボード自身にも PCIe コネクタを備え，PCIe 拡張カード形式の I/O デバイスを接続できるようになっている．

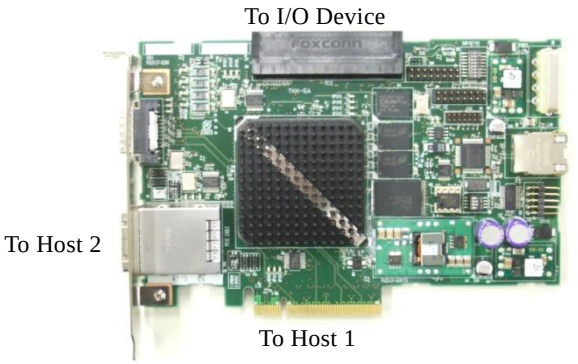

図 5 SIVA: Switch for I/O Virtualization Architecture

SIVA のスイッチ内部は，図 6 に示すように，2 系統のホスト側に接続される Upstream Port (Port 1, 2)，I/O デバイスを接続する Downstream Port (Port 3)，それらを相互接続するスイッチ部から構成される．Downstream Port は，2 系統のホストから共有される．動的な仮想マシン移動のため，以下の機能を実装した．

(1) **Multi-Root 間の動的経路変更**

仮想マシン移動に伴い，移動の対象となる I/O デバイスが接続されている Downstream Port の Multi-Root 間の付け替えが発生する．たとえば，Port 3 配下のアダプタが Port 1 側のホスト(Host 1)から Port 2 側のホスト(Host 2)に移動される場合を想定する．仮想マシンの移動により，当初 Host 1 によって Port 3 の PCI ブリッジ構成が設定されている状態から，Host 2 の PCI ブリッジ構成に切り替える必要がある．このため，あらかじめ Host 1 用の設定と Host 2 用のコンテキストを自由に切り替えられる仮想化機能を搭載した．この機能により Host 1, 2 間の経路変更時間を大幅に短縮可能(～1ms)である

(2) **I/O デバイスの一時停止**

仮想マシン移動に伴う I/O デバイス一時停止には，Downstream Port において Upstream 方向の DMA Write トラフィックを FIFO に格納することで対応を行った．FIFO がフルになるようなケースでは，PCIe Link のクレジットによるバックプレッシャーによりアダプタ側の出力を抑制することが可能である．問題点として，DMA Write を止めてしまうと，PCI のオーダリングルールにより，ホスト側からの PIO Read に対するリプライも停止してしまう．このため，移動対象の仮想マシンからの I/O アクセス停止後に I/O デバイス停止することを保障しなければならない．

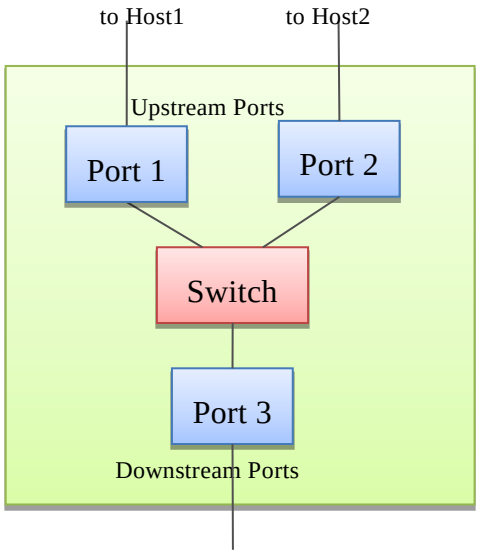

図 6 SIVA スイッチ内部構造

(3) **Dirty Page List の実装**

Pre-copy の間, I/O デバイスからの DMA Write の履歴を保持するための FIFO を SIVA 上に実装した．Downstream Port において，Upstream 方向の DMA Write を検出するとそのアドレスを FIFO に追加する．FIFO の内容は，ハイパーバイザによって参照される．SIVA では FPGA の容量制限のため，ページサイズを 4MB 単位に拡大して保存すべきデータサイズの圧縮を行っている．Dirty Page List では，Pre-copy の間トラフィックのスヌープを行うだけなので，性能劣化の副作用がない．

6.2 Hypervisor: Xen

実験システムのハイパーバイザとしては, オープンソースである Xen を用いた. Live Migration に同期した PCIe スイッチの切り替え, Pre-copy フェーズにおいて I/O デバイスがアップデートした領域の検出などに，ハイパーバイザのソースレベルでの修正が必要不可欠だからである．

Xen の Live Migration の処理に，以下の処理を加えることで直接 I/O に対応した Live Migration ルーチンを作成した．

1. PCI デバイスの情報取得
   Live Migration の対象となる仮想マシンが直接 I/O を使ってアクセスしている PCI デバイス情報を取得する．
2. PCI デバイスの DMA 転送監視
   Live Migration の Pre-copy フェーズにおいて，1 で取得された PCI デバイスから Guest OS 空間に対して DMA Write を行ったページを Dirty Page List に記憶するよう PCIe スイッチに指示を出す．
3. PCIe スイッチからの Dirty Page List 取得
   PCIe スイッチから Dirty Page List を取得し，I/O デバイスによって書き換えられたページのリストを作成する．
4. PCI デバイスの転送停止
   Live Migration の Stop-and-Copy フェーズで, 1 で取得された PCI デバイスからの I/O トラフィックをすべて停止するよう PCIe スイッチに対して指示を出す．
5. PCI デバイスの切り替え
   Live Migration の最終段階で，仮想マシン情報を移動すると同時に PCI デバイスの接続先を切り替えるよう PCIe スイッチに対して指示を出す．
6. PCI デバイスの接続処理
   移動先で切り替えられた PCI デバイスを，再び直接 I/O の対象として利用するようハイパーバイザ内の設定を行う．

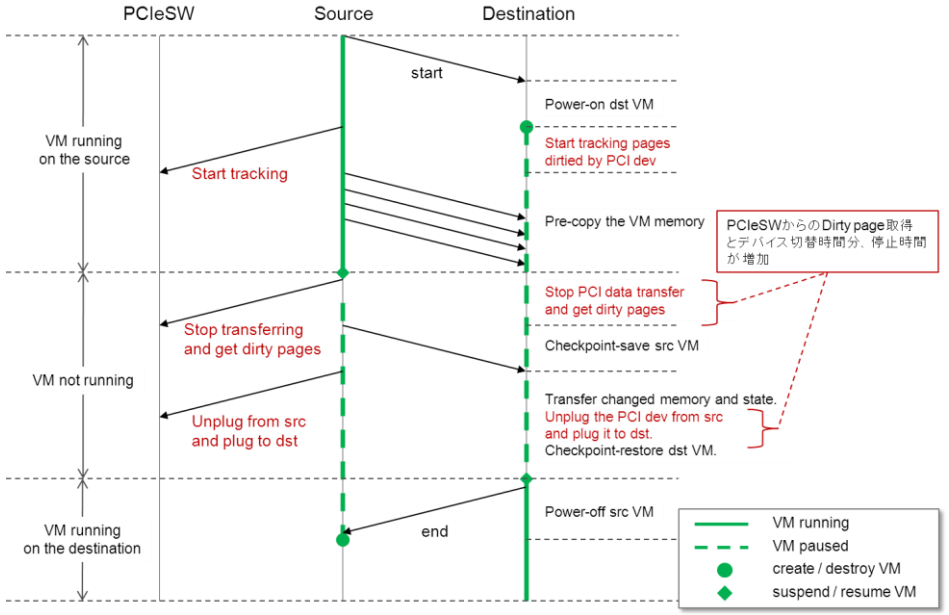

図 7 実験システムの Live Migration タイミングチャート

図 2 で示した一般的な Live Migration のタイミングチャートに，上記処理を追加したものを図 7 に示す．

7. 評価

6 章で述べた実験環境はまだ動作していないため，評価として Live Migration 時の仮想マシン停止時間の見積もりを用いる．

3.2 節で示したように，Live Migration での仮想マシン停止時間は，Stop-and-Copy フェーズの時間に等しい．従って，Stop-and-Copy 時の時間を見積もることで，Live Migration 時の仮想マシン停止時間を見積もることができる．

6.2 節で示したように，本実験環境ではエミュレーション I/O での Live Migration に比べて，6 つの新たな処理が加わっている．これら 6 つの処理のうち，Stop-and-Copy フェーズで処理される部分は 3, 4, 5, 6 である（図 7）．これら処理時間の合計が，エミュレーション I/O に比べた場合の Live Migration 時仮想マシン停止時間の増加分となる．

表 1 に各処理にかかる時間の見積もりを示す．表 1 から，本手法による Live Migration 時の仮想マシン停止時間の増加分は 5ms 程度であり，一般的なエミュレーション I/O 環境での Live Migration 時の仮想マシン停止時間に比べて十分小さな値であることが確認された．

表 1 処理時間の見積もり

処理	時間
3. Dirty Page List 取得	0.4 ms
4. PCI デバイス停止	2 ms
5. デバイス切り替え	1 ms
6. PCI デバイス接続	2 ms
合計	5.4 ms

8. まとめ

本論文では，今後のクラウド環境拡大のために必要とされる，仮想マシンの直接 I/O 技術と，仮想マシンの Live Migration を両立させるための技術について議論し，その解決手段を提案した．

直接 I/O は，仮想マシンから直接 I/O デバイスを操作する技術であり，I/O MMU の機能を使って支援する．この直接 I/O を用いることにより，仮想マシンの I/O 性能の向上，GPU や Flash ROM などのデバイスを仮想マシンからアクセスすることが可能となる．

しかし現状の Live Migration の仕組みでは，1. 移動元と移動先で I/O デバイスが異なる，2. ハイパーバイザから I/O デバイスの動作を制御できない，3. Migration 中に I/O デバイスから書き換えられたメモリ領域をハイパーバイザが認識できない，という 3 つの問題から，直接 I/O を実行中の仮想マシンは Live Migration の対象とできないという課題があった．

そこで本論文では，PCIe スイッチを用いた 3 つの手法を提案し，これら技術を導入することにより，直接 I/O を実行中の仮想マシンでも Live Migration が可能になることを示した．さらに，そのうちの一つを実装し，直接 I/O を実行している仮想マシンでも，エミュレーション I/O を用いている仮想マシンの Live Migration における停止時間（数 10ms～数 100ms）に対し，わずか 5ms 程度の停止時間の増加で Live Migration が可能となることを示した．

今後は，より実用的な環境での評価，クラウド環境への適用などを通じ，本手法の有用性を確認していく予定である．

参考文献

1) “Intel® Virtualization Technology for Directed I/O, Architecture Specification”, Rev 1.2, Intel Corp., Sep. 2008, http://download.intel.com/technology/computing/vptech/Intel(r)_VT_for_Direct_IO.pdf
2) “AMD I/O Virtualization Technology (IOMMU) Specification”, Rev 1.26, Advanced Micro Devices Inc., Feb. 2009, http://support.amd.com/us/Embedded_TechDocs/34434-IOMMU-Rev_1.26_2-11-09.pdf
3) “Single-Root I/O Virtualization and Sharing Specification Revision 1.1”, PCI-SIG, Apr. 2009, http://www.pcisig.com/specifications/iov/single_root/
4) “Multi-Root I/O Virtualization and Sharing Specification Revision 1.0”, PCI-SIG, May 2008, http://www.pcisig.com/specifications/iov/multi-root/
5) A. Kadav and M. M. Swift, “Live Migration of Direct-Access Devices”, First Workshop of I/O Virtualization (WIOV’08), Dec. 2008, http://www.usenix.org/events/wiov08/tech/