RC-008

HPC クラウドにおける仮想計算機の割込み通知機構の改良

Improvement of Interrupt Handler Mechanism for Virtual Machines on HPC Cloud

本庄 賢光†‡　窪田 昌史†　北村 俊明†
Masamitsu Honjo†‡　Atsushi Kubota†　Toshiaki Kitamura†

1. はじめに

近年，ハードウェアやソフトウェアなどの計算資源をインターネット上のサービスとして提供するクラウドコンピューティングが注目されており，Amazon EC2[11]，北海道大学アカデミッククラウド[12]など，HPC(High Performance Computing)用の実行環境を提供するサービス例も現れている．このような HPC クラウドを実現するには，計算資源の増減と計算資源の移動の要件を満たさなければならない．

計算資源の増減　ユーザが実行するジョブに対して，割当て可能な CPU や記憶容量などの計算資源が豊富にあり，ユーザの要求や実行するジョブのピークに合わせて，計算資源の増減を短時間かつ容易に行える．

障害対策　データセンターでは，大量の計算機（ノード）などの機器を設置し，ノードの管理システムによってクラウドのシステムを実現させている．大量の計算機を管理しているデータセンター全体で障害が発生した場合，そのデータセンター内では稼働中のジョブを継続させることはできない．実行中のジョブのチェックポイントを別の場所などから定期的に取得し，障害発生時には，他の場所にあるデータセンターで取得したチェックポイントから復元し，ジョブを再開する．

消費電力の削減　負荷の少ないジョブが複数台で稼動している場合，1 台もしくは現在よりも少ない数の計算機にジョブを集約し，ジョブが稼動していない計算機の電源を切る，もしくはサスペンド状態にし，システム全体の消費電力を削減する．

高速なネットワーク　HPC アプリケーションでは，複数のノードを用いて並列処理が行われるため，ノード間で大量のデータの送受信が必要となる．そのため Gibabit Ethernet に限らず，10Gigabit Ethernet や InfiniBand といった高速なネットワークも用いられる．

計算資源の増減への対応，故障対応，消費電力の削減といった要件は，一般的なクラウドでも必要とされており，これらの要件を達成するために仮想計算機(VM; Virtual Machine)が広く利用されている．

ジョブを VM 上で実行し，必要に応じてジョブを VM とともに実計算機間で移送を可能にする．利用可能な VM 数を増減することにより，クラウドが提供する計算資源も容易に増減させることができる．計算機の障害時に処理中のジョブを他の計算機に移動して処理を継続させたり，障害を予測できる場合にはあらかじめジョブを他の計算機に移動しておいたりすることができる．また，計算機の OS のメンテナンスやアップデート，地震や火災などの予期できない災害のために処理が急遽継続できない場合にも，他の計算機へジョブを移動することができる．

一方， HPC アプリケーションでは，複数の実計算機を用いて並列処理が行われ，計算機間で大量のデータの送受信が必要となることが多いため，HPC クラウドでは VM 間で大量のデータ通信が発生する． しかし，VM 間のデータ通信は，実計算機に比べ，性能が大幅に低下することが指摘されている[13]．解決方法として，VM のゲスト OS から実計算機の NIC(Network Interface Card)などのネットワークデバイスに対し，直接データの入出力処理を実行する，PCI パススルーと呼ばれる性能改善手法や，各 VM に割り当てられる CPU 時間を調節することで通信待ちのプロセスへ優先的に CPU 資源を割り当てる手法[7]が提案されている．

本稿では，仮想計算機モニタ Xen[1]のデータ通信時のネットワークデバイスから VM に対する割込み通知処理に着目し，既存の PCI パススルーをさらに改良する手法であるネットワークデバイスのための割込み通知機構の改良を提案する．

また，提案手法を Xen へ実装し， 6 core プロセッサを 2 基搭載した Intel Xeon X5660 2.80 GHz，メモリ 24GB のノードからなる PC クラスタで評価を行った．NIC である Intel 82574L Gigabit Network Connection に PCI パススルーを用いた VM を稼働し，NAS Parallel Benchmarks 3.3.1 クラス B のアプリケーションの並列 MPI プログラムを実行した．その結果，8 ノード 64 プロセスを実行したところ，割込み通知機構の改良により，既存の PCI パススルーの手法に比べ，約 1〜6%の高速化を確認した．

以下，2 章では仮想計算機モニタ Xen の概要，VM 上の MPI プログラム実行時に生じる問題点について述べる．3 章では，VM における I/O アーキテクチャ，PCI パススルーとその問題点について述べる．4 章では，本研究で改良を行ったネットワークデバイスのための割込み通知機構について述べる．5 章では，NAS Parallel Benchmarks(NPB)を用いて，提案方式と既存の PCI パススルー方式，VM への CPU スケジューリングを変更する手法とを比較し，提案手法の有効性の評価を行う． 6 章では，まとめと今後の課題について述べる．

2. 仮想計算機モニタ Xen

仮想計算機モニタ Xen は図 1 に示すように，ハードウェア(CPU，メモリ)を直接管理し，必要最低限度の制御メカニズムを備えたハイパーバイザを実装している．ハイパーバイザ上で動作するホスト OS は，Xen の動作に必要不可欠の VM で，Domain0 とも呼ばれる．この VM の上では，実デバイスドライバや，ハイパーバイザと連携した仮想化の制御を行う．ゲスト OS は，ユーザがプログラムを実行させる VM で,DomainU とも呼ばれる．

†広島市立大学, Hiroshima City University
‡現在　株式会社インターネットイニシアティブ,
Currently with Internet Initiative Japan, Inc.

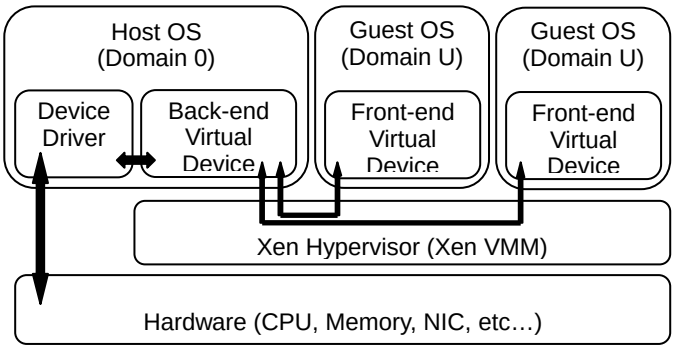

図1　Xen の準仮想化

Xen では完全仮想化と準仮想化の 2 通りの仮想化方法がある．ゲスト OS を変更することなく動作させ，計算機全体を仮想化する方法が完全仮想化である．逆に，ゲスト OS に変更を加え，ハイパーバイザに適合するように最適化する方法が準仮想化である．

また，Xen では図 1 のように，VM 上でのみで動作する仮想デバイスドライバが存在する．ゲスト OS 上で動作するフロントエンド仮想ドライバとホスト OS 上でバックエンド仮想ドライバとを連携させることにより I/O 性能を向上させる．

本研究では，HPC クラウドを実現するにあたり，仮想化ソフトウェアを用いて実計算機の計算資源を仮想化し，仮想計算機(VM:Virtual Machine)上でジョブを実行することを想定している．4 章で述べる提案手法の実現にあたって，仮想化ソフトウェアのソースコードの一部に変更を加えるため，オープンソースの仮想化ソフトウェアを使用する必要がある．また，仮想化によるオーバヘッドが極力少ない仮想化手法を選択する必要がある．よって，仮想化のオーバヘッドが少ない仮想化手法である準仮想化を利用できる仮想計算機モニタ Xen を本研究で用いた．

2.1 マイグレーション

1 章で述べたように，HPC クラウドで要求される計算資源の移動を実現するにあたり，VM の移送（マイグレーション）が必要となる．

VM のマイグレーションは，あるホストの上で稼働している VM を，ゲスト OS のイメージ丸ごと別のホストへ移送させることである．Xen では，マイグレーションとライブマイグレーションの 2 種類のマイグレーションを実装している．マイグレーションは，VM を 2〜3 秒停止し，停止している間に VM のメモリの内容や OS イメージを他のホストへ移送する．しかし，ゲスト OS に割り当てられたメモリサイズが大きい場合，移送先への転送時間も長くなり，VM の停止時間も長くなる．一方，ライブマイグレーションは，移送先に対し、稼働中の VM のメモリの内容や OS イメージを転送する．転送元と転送先とで内容が異なる部分は何度も差分転送を行う．移送先と移送元との整合性を取るために VM を一瞬停止させる[10].

2.2 VM 上で MPI プログラムを実行させた場合に生じる問題

本研究では，MPI(Message Passing Interface)を用いて記述された並列プログラムを HPC クラウド上で実行することを想定する．1 章で述べたように実行中の並列ジョブを実計算機間で移動することが必要となる場合には，MPI の並列プロセスを VM 上で稼働させたままマイグレーションさせることになる．

VM 上で MPI プログラムを実行させた場合，MPI の送信/受信の処理が極端に遅くなり，MPI プログラム全体の実行時間が増大する．本節では，2.2.1 節でその実行時間増大の理由について考察する．また，この問題に対する対処法を 2.2.2 節から 2.2.4 節で取り上げ，それらの得失について考察する．

2.2.1 問題点

HPC アプリケーションでは MPI の通信関数による大量のデータ通信が発生する．MPI の関数内で通信完了待ちになる場合，MPI の多くの実装では MPI プロセスがビジーウェイトになり CPU を消費してしまう．VM 上で MPI プログラムを実行させた場合，VM に割り当てられた CPU 時間を消費しきるまで，他の VM に CPU 時間を割り当てられない．Xen では通信などの入出力を管理する Domain0 に CPU が割り当てられず，これは，大きなオーバヘッドを生じることにつながる．

2.2.2 Xen のクレジットスケジューラの改良

この問題に対し，Xen のハイパーバイザがホスト OS やゲスト OS に割り当てる CPU 時間を調整する手法が提案されている[10].

問題となっているのは，MPI プロセスを実行する VM に割当てられるタイムスライスのデフォルト値が大きすぎるために，CPU 時間を割り当てられたまま，通信待ちを行っている VM が存在していることである．この手法では，VM に適切なタイムスライスを割当てるため，Xen のハイパーバイザのスケジューラであるクレジットスケジューラのパラメータを調整している．調整対象のパラメータは，weight, cap, timeslice, tick である．本手法では，アプリケーションごとにパラメータの最適値を求めることが困難であることが問題点となっている．

2.2.3 VM スケジューラとゲスト OS のプロセススケジューラの協調動作

2.2.1 節で指摘した問題に対する別の解決策として，Xen の VM スケジューラであるクレジットスケジューラとゲスト OS のプロセススケジューラとの協調動作を可能にする手法が考えられる．ただし，我々は，以下に述べる理由により，この手法の実現は困難であると考えている．

この手法では，ゲスト OS と Xen のハイパーバイザを協調動作させるために，ゲスト OS 用のプロセススケジューラと VM スケジューラの両方に修正が必要であり，その実装は困難である．また，ゲストの VM とハイパーバイザは独立して稼働しているにもかかわらず，スケジューラを協調させるために両方の状況や動作を監視し，そこから得た情報を集中して管理することは，セキュリティの観点においても問題がある．さらに，ゲスト OS とハイパーバイザを協調動作させることで，ゲスト OS に不具合が生じた場合，計算機のシステム全体にも被害を及ぼす可能性がある．

2.2.4 PCI パススルー

2.2.1 節の問題のもう 1 つの解決方法として，PCI パススルーを用いることが挙げられる．詳細は 3 章で述べるが，ネットワークデバイスに PCI パススルーを用いた場合，仮想計算機のゲスト OS から実計算機のネットワークデバイスに直接データ通信を行えるため，実計算機並の I/O 性能を得ることができ，通信完了待ちのオーバヘッド削減につながる．PCI パススルーによって VM のノード間の移送が困難になるという問題も存在するが，これも 3.2 節で説明するようにデバイスのハードウェアを変更することで対応可能である．

以上の議論により，我々は 2.2.1 節で指摘した問題を解決する方法として PCI パススルーが有望であると判断した．次章では，PCI パススルーについて詳細を述べる．

3. PCI パススルー

HPC アプリケーションでは大量のデータ通信を必要とすることから，HPC クラウドでは，VM 間で大量のデータ通信が発生することが想定される．しかし，前章で述べたように VM 間でのデータ通信は実計算機間に比べ，大幅に性能が低下する．

本章では，VM の通信を含む I/O アーキテクチャの概要について説明し，その性能改善手法である PCI パススルーについて述べる．さらに Xen で PCI パススルーを用いた場合の問題点について述べる．

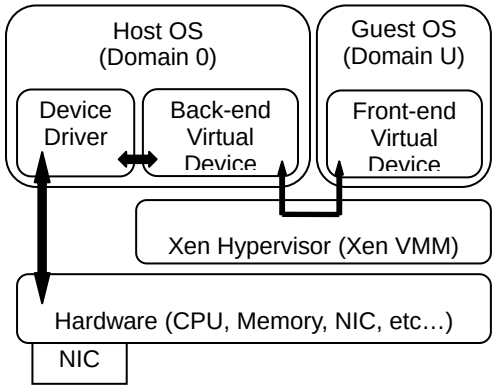

図 2　仮想化 I/O 方式

3.1 仮想 I/O 方式と直接 I/O 方式

Xen における VM の I/O（入出力）アーキテクチャを図 2,3 に示す．ゲスト OS がネットワークデバイスにアクセスし，データ通信を行う方式は，仮想 I/O 方式(図 2)と直接 I/O 方式(図 3)に分類される．以降，ネットワークデバイスにおける各 I/O アーキテクチャについて説明する．

3.1.1 仮想 I/O 方式

仮想 I/O 方式(Virtualized I/O)は，ホスト OS 側のデバイスドライバを，抽象化かつ多重化された仮想デバイスドライバとして提供し，ゲスト OS 専用の仮想デバイスドライバと連携し，データの授受を行う方式のことである．

Xen では，スプリットデバイスドライバ構造を採用しており，ホスト OS にはバックエンド，ゲスト OS にはフロントエンドの仮想デバイスドライバを利用し，両方のデバイスドライバはハイパーバイザを介して，通信を行う．

仮想 I/O 方式の場合，複数の VM から 1 つの物理デバイスを共有することに関しては論理的な制限がなく，ハードウェアによる仮想化支援機構によるサポートも不要である．しかし，物理デバイスを複数の VM で共有することになるため，ホスト OS または仮想化ソフトウェア(VMM)側の仮想スイッチやデバイスの多重化によるオーバヘッドが発生する．

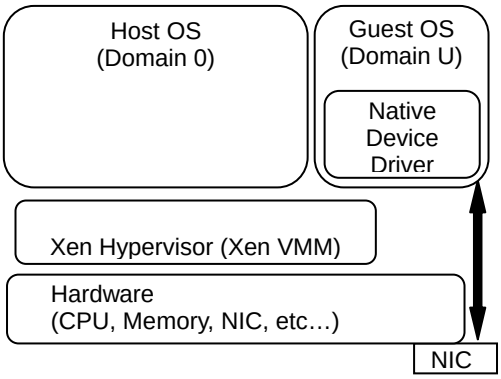

図 3　直接 I/O 方式

3.1.2 直接 I/O 方式

直接 I/O 方式(Direct I/O)は，図 3 に示すように，VM のゲスト OS がネイティブのネットワークデバイスドライバを用いて，VMM(仮想計算機モニタ)をバイパスし，ネットワークデバイスに直接アクセスする．それにより，実性能に相当するほどの I/O 性能を得ることができる．

直接 I/O 方式を利用するにあたり，CPU やチップセットなどのハードウェアが，Intel Virtualization Technology for Directed I/O (Intel VT-d) 等の仮想化支援機構に対応している必要がある[4,5]．直接 I/O 方式では，PCI パススルー，SR-IOV (Single Root I/O Virtualization) 等の方式が存在する．Xen では PCI パススルー，一部のデバイスに限り SR-IOV に対応し，利用することができる．

仮想化 I/O 方式はスケーラビリティ，直接 I/O 方式は性能の点で有利であり，それぞれトレードオフの関係にある．

PCI パススルーを適用する場合，特定の VM がデバイスを占有することになり，VM の移送が困難になる．そのため，Zhai ら[8]や Kadav ら[9]によって VM の移送手法が提案されている． これらの手法と，SR-IOV 対応デバイスを併用することによって VM の移送が容易に実現できると考えられる．

しかし，本研究の実験で用いるデバイスが SR-IOV 非対応である．SR-IOV 対応のデバイスを用いた場合には，VM の移送を実現しつつ，I/O 性能を向上させることが期待できる．今回は，VM の移動の評価は行わず，ネットワークデバイスに PCI パススルーを用いた場合の HPC アプリケーションの性能面に注目する．

3.2 割込みインジェクション

PCI パススルーは，データ転送に関しては，デバイスからゲストの VM に対して直接 DMA (Direct Memory Access) 転送を行うため，仮想計算機モニタ (VMM: Virtual Machine Monitor) を介在させずに処理させることができる．

割込みに関しては複数の VM 上で IRQ が共有されたり，

割込みを受け取る仮想 CPU が実行可能状態にない可能性もあるため，まず Xen のイベントチャネルが割込みを一度受け取り，その割込みを VM に転送する実装となっている．この処理を割込みインジェクションと呼び，割込みインジェクションの実装方法は仮想化ソフトウェアごとで異なる．Xen の場合，割込みはイベントとして抽象化され，イベントチャネルを経由してゲスト OS に通知する仕組みになっている．

3.3 Xen における割込みの伝送方式

Xen で PCI パススルーを用いたネットワークデバイスの場合のデータの送信，および，受信時の割込みの転送方法について詳細に述べる．

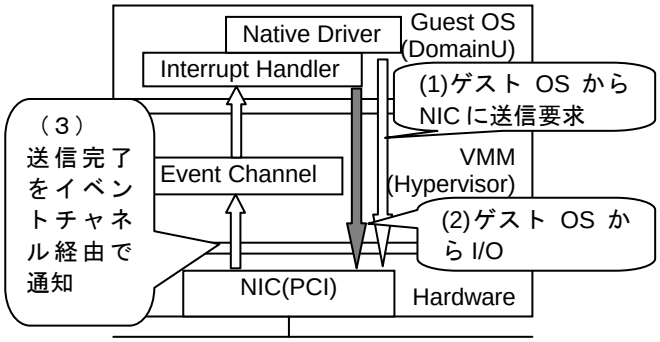

図 4　データ送信時の操作

3.3.1 データの送信

データの送信時の挙動を図 4 に示す．

ゲストの VM からデータを送信する場合は，まず，ゲスト OS から NIC に対し，送信を行うための要求を送る．

この時，NIC が要求を受理できない（ビジー状態）場合は，そのデータを送信可能になるまで送信処理を保留にする．送信可能な場合，ゲスト OS 上のバッファから PCI デバイスにデータ転送を行う．転送が完了するとゲストに転送完了済みのハードウェア割込みを伝送する．この時，イベントチャネルを経由して，ゲスト OS に割込みが通知され，データ転送が完了したバッファ領域をデバイスから解放する．

3.3.2 データの受信

データの受信時の挙動を図 5 に示す．

PCI デバイスにデータが到着した際，ゲスト OS に対しハードウェア割込みを伝送する．送信時の割込みの転送と同様，イベントチャネルを経由してゲスト OS に通知される．ゲスト OS が割込みを受け取った後に，デバイスからゲスト OS 上のメモリに直接 DMA 転送される．

3.4 PCI パススルーの問題点

Xen では PCI パススルーしたネットワークデバイスからの割込みはイベントチャネルを必ず経由してゲスト OS に通知される．ハードディスクやキーボード等といった他の各種デバイスからの割込みやタイマ割込みなども同様に，イベントチャネルを経由して処理される．イベントチャネルを経由する割込みは，割込み発生元デバイスの種類に関わらず，全て平等に扱い，直列で順番に処理される．イベントチャネルでは，PCI パススルーを用いたネットワークデバイスの割込みだけを優先して処理することは行わないため，ゲスト OS に対して割込み通知がすぐに処理されないことによる，ネットワークデバイスの I/O の性能低下につながる[3]．

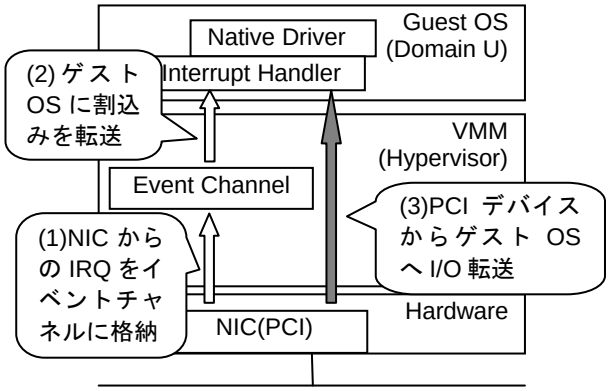

図 5　データ受信時の動作

4. 割込み通知機構の改良

仮想計算機モニタ Xen では，ゲスト OS に対する各種デバイスの割込みは，全てイベントチャネルを経由して処理される．前章では，PCI パススルーが適用された PCI デバイスからの割込みも同等に扱われることによる PCI デバイスにおける I/O 性能の低下について指摘した．本章では，渡辺ら[2]の方式と同様に，PCI パススルーを用いたネットワークデバイスからの割込みを，他のデバイスからの割込みとは分離し，優先して処理を行う割込み通知機構の改良を提案し，その実装方法について述べる．

4.1 PCI パススルーを用いたネットワークデバイスのための割込み通知機構

PCI パススルーを用いたネットワークデバイスからゲスト OS に対する割込みを，他のデバイスからの割込みと分離することにより，他の割込みの順次処理による影響を受けなくなる．

本研究にて実装を行った，PCI パススルーを用いたネットワークデバイスのための割込み通知機構の提案方式について述べる．

- 割込み通知機構の対象となる割込みは，PCI パススルーを用いたネットワークデバイスから発生するハードウェアの割込みである．この割込みをハイパーバイザ内で扱われる全てのデバイスからの割込みと区別し，割込み通知機構を通じて，優先的にゲスト OS に通知する．
- PCI パススルーを用いたネットワークデバイスからの割込みの識別には，ホスト OS が割当てたデバイスの IRQ 番号を用いる．Xen を起動させたときに，一度，全てのデバイスに IRQ 番号が割振られるが，PCI パススルーを用いるネットワークデバイスには，VM の起動時に，再度，別の IRQ 番号が割当てられるため，新しく割振られた IRQ 番号をもとに，割込みを区別する．
- Xen のハイパーバイザ内に到着した割込みは，イベントチャネルに格納する前に，IRQ 番号を確認し，対象と

なる割込みかどうかの区別を行う．これにより，全ての割込みが平等に扱われることによる割込みの直列化を回避させる．

- 複数の仮想 CPU が割当てられている VM が稼動している場合，ネットワークデバイスからの割込みを個々の仮想 CPU に対して転送する．

以上，上記の割込み通知機構の改良手法を仮想計算機モニタ Xen に実装した．

4.2 Xen への実装

PCI パススルーを用いたネットワークデバイスからゲスト OS に対するハードウェア割込みを他のデバイスからの割込みと分離し，専用の割込み通知機構で処理を行うように改良を加えた．Xen-3.4.3 のイベントチャネルのソースコードに改良を加えたソースコードの一部を図 6 に示す．

今回は VM の起動時に割り当てられる特定の PCI デバイス(ネットワークデバイス)の IRQ 番号を事前に調査し，割り当てられる IRQ 番号をソースコード内に記述した．各種デバイスからゲスト OS に対するハードウェア割込みがハイパーバイザが到着した場合，send_guest_pirq 関数が実行される．まず，PCI パススルーを行なった特定のデバイスからの割込みかどうかを判別する．特定のデバイスからの割込みだった場合，次に，ゲスト OS 側でハイパーバイザからの割込みを受信できるようにポートを準備する．そして，専用の割込み通知機構を通じてゲスト OS に割込みの伝送を行う．特定のデバイス以外の割込みだった場合は，通常の割込み通知機構を用いて直列に順次処理されるようになっている．

現段階の実装では，特定の計算機構成(本研究で用いる計算機構成)のみに有効な方法である．その構成以外の計算機や新たに IRQ が割振られるデバイスが追加されることで，対象とするデバイスに割り振られる IRQ 番号が異なってくる．そのため，本研究とは異なる環境の場合などは，割込み通知機構を適用する前に割当てられる IRQ 番号を調査し，ソースコード内の IRQ 番号を変更する必要がある．

4.3 関連研究

ネットワークデバイスに PCI パススルーを用いた際の性能改善に関する研究について述べる．渡邉ら[2]は，組み込みシステム用の RTOS(リアルタイム OS)のリアルタイム性を保証するために，デバイスの I/O と割込みをゲスト側に占有させ，デバイス毎に対し排他的な割込み通知機構を実装させている．割込み通知機構を実装したという点では共通であるが，渡邉らの割込み通知機構がゲストの VM に割当てる仮想 CPU 数を 1 個に限定し，その仮想 CPU に対してデバイスからの割込みを転送させている．しかし，複数の仮想 CPU が割り当てられたゲストの VM の場合，個々の仮想 CPU への割込みの転送には対応していないという点が，本研究とは異なる．

```
int send_guest_pirq(struct domain *d,
int pirq)
{
   int port;
   struct evtchn *chn;

  //ホスト OS 側で再割り当てされる PCI
  //デバイスの IRQ 番号(ここでは N)を指定
  if( pirq == N ){

     //ゲスト OS 用の IRQ 番号に変換
     port = d->pirq_to_evtchn[pirq];

    //ゲスト OS 側で割込みを受信する
    //ためのポートを準備
     chn = evtchn_from_port(d, port);

     //特定の PCI デバイスからの割込みを
     //専用の割込み通知機構を用いて処理
     return
       evtchn_set_pending_for_ppt_pirq
      (d->vcpu[chn->notify_vcpu_id], port);
  }
  …
}
```

図 6　改良を加えた event_channel.c の一部

5. 評価

本章では，4 章で提案した割込み通知機構の改良手法の有効性を評価する．実験では，PC クラスタ上で NPB(NAS Parallel Benchmarks)[6]の並列プログラムを実行する．実行には，以下の 4 通りの方式を用いる．

(1) BMM(Bare Metal Machine)
(2) ハイパーバイザのスケジューラのパラメータを調整したした VM
(3) PCI パススルーを適用した VM
(4) 提案手法: PCI パススルーを適用し，割込み通知機構を改良した VM

(1)が VM を用いない通常の実行方式であり，(2)-(4)が VM 上での実行となる．

提案手法の有効性は，以下の観点で評価する．

- 従来手法である(2)や(3)に比べ，提案手法の(4)によって性能改善されているか．
- VM を用いない(1)に対して，提案手法(4)では VM を用いるオーバヘッドが小さく抑えられているか．

以下，5.1 節で上記の(1)-(4)の実験環境について述べ，5.2 節と 5.3 節でそれぞれ flat-MPI 方式の実行結果と MPI と OpenMP のハイブリッド実行の結果を示し，その結果を考察する．

5.1 実験環境

実験には表 1 および図 7 に示す PC クラスタである CRAY CX1 を使用した．CX1 の筐体は 8 台のノード，Infiniband，Gigabit Ethernet のネットワークスイッチ，電源で構成されている．各ノードは 6 core の Intel Xeon Processor X5660(2.80GHz)を 2 基搭載し，各プロセッサあたり 12GB，

合計 24GB のメモリが接続されている．HyperThreading は無効にしている．各ノードと NFS サーバとの接続には Gigabit Ethernet を用いた．VM の OS イメージは，CX1 の外部に接続した NFS サーバからネットワーク経由で取得するようにした．

CRAY CX1 には高速通信可能なネットワークデバイスの Infiniband が備わっているが，Mellanox 社が現在提供しているデバイスのファームウェア及びドライバでは，PCI パススルーを用いることができなかったため，本研究では Infiniband は使用しなかった．

表 1　実行環境の内部構成

Domain 0 (Host)	
CPU	6 core Intel Xeon X5660 2.80GHz ×2
Memory	24GB
HDD	500GB
Ethernet	Intel 82574L Gigabit Network Connection ×2
Linux Distribution	CentOS 5.7 x86_64
Linux Kernel (Domain 0)	2.6.18.8-xen
Linux Kernel (BMM)	2.6.18-238.19.1.el5
Xen	3.4.3
Domain U (Guest)	
CPU	1 または 12 VCPU
Memory	1GB または 16GB
HDD	8GB または 16GB
Linux Distribution	CentOS 5.7 x86_64
Linux Kernel	2.6.18-274.3.1.el5xen
Network Switch	
Ethernet Switch	16 RJ-45 auto-sensing UTP ports (within CRAY CX1)
Ethernet Switch	Buffalo LSW3-GT-16NSR(16ports)
Software	
MPI	OpenMPI 1.4.3
Compiler	PGI 11.8 (最適化 –fastsse)

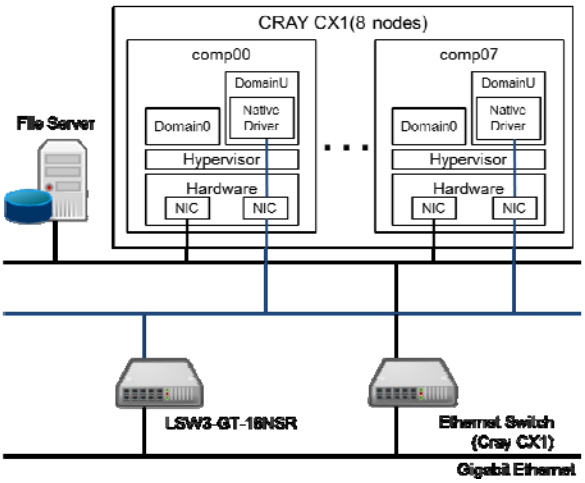

図 7　評価に用いた PC クラスタの機器構成

本研究の実験で実行するアプリケーションは，使用したクラスタが，1 ノードに 12 コアを搭載していることを考慮し，その性能を引き出す実行方式としてフラット MPI と MPI+OpenMP のハイブリッド並列化の 2 通りを採用した．

flat-MPI での実行には NPB 3.3.1-MPI の 6 種類のプログラム(BT，CG，FT，LU，MG，SP)を用い，ハイブリッド並列実行には NPB MZ(Multi-Zone) 3.3.1-MPI の BT-MZ と SP-MZ を用いた．プログラムサイズは NPB 3.3.1-MPI ではクラス B および C，NPB MZ 3.3.1-MPI ではクラス C を用いた．クラス B より C の方がサイズは大きくなっている．

実計算機(BMM)での実行方式(1)では，通常の Linux カーネルを稼働させ，その上で flat-MPI プログラムを実行した．

実行方式(2)のハイパーバイザのスケジューラのパラメータを調整する方式では仮想デバイスドライバを用いた．各ノード上では管理用の VM である Domain0 を 1 台，MPI プログラムを実行するための VM である DomainU を 8 台起動した． DomainU を起動した個数を 1 ノード上のコア数の 12 でなく 8 としたのは，アプリケーションである NPB の制約に合わせたためである．各 DomainU に割当てる仮想 CPU(VCPU)数は 1 とし，VM 上で動作する MPI プロセスは 1 個として，flat-MPI のプログラムを実行した．クレジットスケジューラのパラメータであるタイムスライスとティックはともに 1ms とし，Domain0 の優先度を 1024 に変更した．

PCI パススルーを適用した従来手法(3)と提案手法(4)では，各ノード上で，管理用の VM である Domain0 と MPI プログラム用の VM である仮想計算機 DomainU を 1 台ずつ稼働させる．1 台の計算上のコア数である 12 個に対応して，12 個の VCPU を割当てる．flat-MPI 方式の場合は 1 台の DomainU 上で複数の MPI プロセスを稼働させ，MPI+OpenMP のハイブリッド実行の場合は各ノードの DomainU で 1 つの MPI プロセスを稼働させた．3.2 節で述べたように，使用した NIC の制約により各ノード上でのゲスト OS(VM)は 1 台としている．

今回，使用したネットワークデバイスが SR-IOV に対応していないため，VM の移動を行った場合の評価は行っていない．

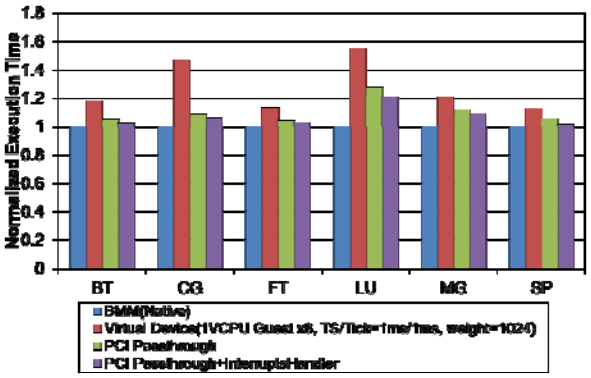

図 8　NPB3.3.1-MPI クラス B 8 ノード 64 プロセスの実行結果

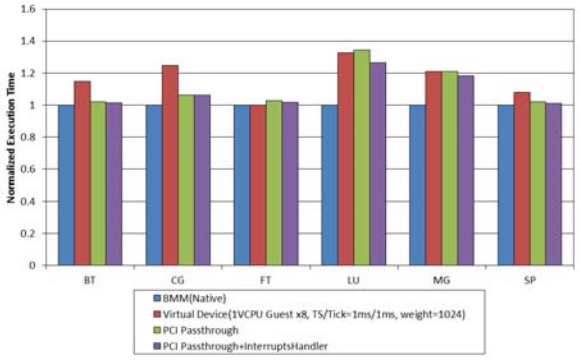

図 9　NPB3.3.1-MPI クラス C 8 ノード 64 プロセスの実行結果

5.2 提案方式の割込み通知機構の性能評価

本研究で提案方式であるネットワークデバイスのための割込み通知機構を実装し評価を行なった．その実行結果を図 8 と図 9 に示す．縦軸は VM を用いない通常の実行方式である(1)BMM の実行時間を 1 とした相対実行時間である．

実行結果から，クラス C の FT と CG を除いたアプリケーションにおいて，本研究の提案方式(4)が，従来方式の(2)クレジットスケジューラのパラメータ調整，(3)既存の PCI パススルーの性能を上回っていることがわかる．

提案手法の(4)割込み通知機構により，従来の(3)PCI パススルーのみ用いた場合に比べ，約 1〜6%高速化された．これは，ネットワークデバイスに PCI パススルーを用いた際に生じる，割込みインジェクションによるオーバヘッドが削減されたことによるものである．

実計算機の(1)BMM との比較では，提案手法の(4)割込み通知機構を実装した PCI パススルーでは，実行時間はクラス B で 1%から 20%，クラス C で 1%から 26%の増大となる．これに対して提案手法を適用しない(3)PCI パススルーのみの場合，実行時間はクラス B で 5%から 27%，クラス C で 2%から 34%の増大となる．特に BT や SP のベンチマークアプリケーションでは，(1)BMM の実行時間に迫る結果となった．

既存の PCI パススルーでは，クレジットスケジューラのパラメータ変更する従来方式よりも，実行時間の増大を抑えることができたが，ノンブロキング通信時に割込みを，ただちにゲスト OS に転送出来なかったことによるオーバヘッドが生じた．提案方式では，既存手法の PCI パススルーで生じた割込みインジェクションのオーバヘッドを削減し，実計算機の実行性能に迫ることができた．

5.3 NPB MZ 3.3.1-MPI による提案方式の評価

今回使用したクラスタのノードには 12 コアが搭載されているため，MPI と OpenMP によるハイブリッド実行が可能である．そこで，PCI パススルーを適用している場合の，割込み通知機構の改良による有効性の評価を，ハイブリッド実行時についても行った．本研究での提案方式と従来方式の PCI パススルーでの NPB MZ 3.3.1-MPI クラス C の実行結果を図 10 と図 11 に示す．縦軸は性能(MFlops)であり，数値が大きい方が性能が高い．

BT と SP の両方のアプリケーションにおいて，合計スレッド数が等しくノード数とノードあたりのスレッド数を変更させて実行させた場合，ノードあたりのスレッド数を少なくし，ノード数が多いほうが高い性能が得られた．

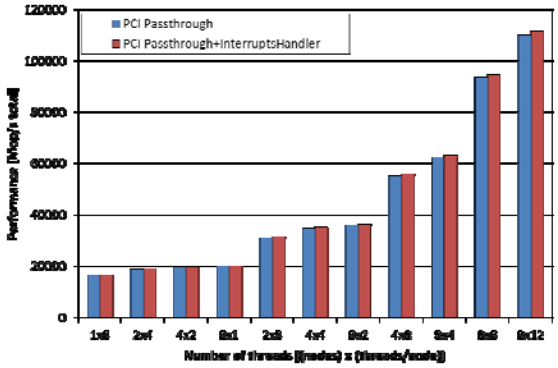

図 10　NPB MZ 3.3.1-MPI MZ-BT クラス C

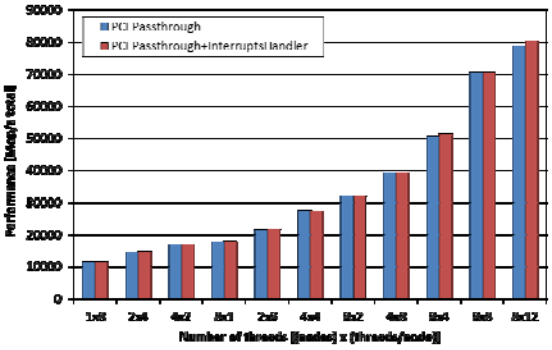

図 11　NPB MZ 3.3.1-MPI MZ-SP クラス C

従来方式と本研究での提案方式との実行結果を比較した場合，提案方式が従来方式より，BT と SP それぞれのアプリケーションで 0.1〜2.0%程度高速化された．特に，ノード数が増えていくにつれて，高速化の割合が高くなっていった．MZ-BT と MZ-SP はノード内通信は OpenMP のスレッド間で通信を行い，ノード間の通信では MPI を利用してネットワークデバイスを介して通信を行う．NPB 3.3.1-MPI と同様の高速化が得られなかった理由は，ノード間をまたが

るプロセス間通信の回数が NPB MZ 3.3.1-MPI の方が少なく，ネットワークデバイスに対する割込みの量が減ったため，本研究での提案方式の特徴を生かしきれなかったためだと考えられる．

この結果から，ノード間をまたがるプロセス間通信が頻繁に行われるアプリケーションの場合に限り，本研究の提案方式による高速化が望めることがわかった．

6. おわりに

HPC クラウドでは，VM 間で大量のデータ通信が行われることから，VM 間のデータ通信で生じる性能低下を改善するために PCI パススルーを用いた．ネットワークデバイスに PCI パススルーを用いた際に生じる，通信時の割込み処理に着目し，仮想計算機モニタ Xen のハイパーバイザ内の割込み通知機構の改良方法を提案し，実装を行った．

Nas Parallel Benchmarks 3.3.1-MPI のアプリケーションをクラス B と C で実行して，提案手法の有効性を評価した．実計算機上での実行時間を基準にすると，割込み通知機構を改良前では，クラス B で約 5〜27%程度，クラス C で約 2〜34%程度の実行時間が増大していたが，改良後では，クラス B で約 1〜20%，クラス C で約 1〜26%にまで実行時間の増大を抑制した．これは提案方式により，改良前の割込み通知機構に比べ，約 1〜6%の高速化が図られていることになる．特に，通信を頻繁に行うアプリケーションでは，実計算機上での実行時間に迫る結果となった．

今回は PCI パススルーを用いたネットワークデバイスの制約により，VM の移送ができなかったが，SR-IOV (Single Root I/O Virtualization)に対応したネットワークデバイスを用いることで VM の移送を実現させ，その際の性能評価を行うことを今後の課題とする．

参考文献

[1] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield, “Xen and the Art of Virtualization”, in Proceedings of the Symposium on Operating Systems Principles, pp.164-177 (2003).

[2] 渡邉 和樹, 片山 吉章, 松本 利夫, 瀧本 栄二, 樫山 武浩, 毛利 公一, “仮想計算機モニタ Xen における RTOS 向け割込み通知機構”, コンピュータシステム・シンポジウム ComSys2011 論文集, pp.23-31 (2011).

[3] 高野 了成, 池上 務, 広渕 崇宏, 田中 良夫, “InfiniBand を PCI パススルーで用いる仮想化 HPC クラスタの性能評価”, 先進的計算基盤システムシンポジウム SACSIS2011 論文集, pp.109-116 (2011).

[4] AMD, Inc., “AMD I/O Virtualization Technology (IOMMU) Specification”, http://support.amd.com/us/Embedded_TechDocs/34434-IOMMU-Rev_1.26_2-11-09.pdf (2009, last access: 2013.4.15).

[5] Intel, Inc., “Intel® Virtualization Technology for Directed I/O Architecture Specification”, http://www.intel.com/content/dam/www/public/us/en/documents/product-specifications/vt-directed-io-spec.pdf (2011, last access: 2013.4.15).

[6] NASA Advanced Supercomputing (NAS) Division, “NAS Parallel Benchmarks”, http://www.nas.nasa.gov/publications/npb.html (last access: 2013.4.15).

[7] 本庄 賢光, 窪田 昌史, 北村 俊明, “VM 上の MPI プログラムの通信オーバヘッドの性能評価”, コンピュータシステム・シンポジウム ComSys2010 論文集, pp.91-100 (2010).

[8] Edwin Zhai, Gregory D. Cummings, and Yaozu Dong, “Live Migration with Passthrough Device for Linux VM” in Proceedings of the Linux Symposium, Vol.2, pp.261-267 (2008).

[9] Asim Kadav and Michael M. Swift, “Live Migration of Direct-Access Devices”, Workshop on I/O Virtualization (WIOV’08), pp.95-104 (2008).

[10] Christopher Clark, Keir Fraser, Steven Hand, Jacob G. Hansen, Eric Jul, Christian Limpach, Ian Pratt and Andrew Warfield, “Live Migration of Virtual Machines”, NSDI’05, pp.273-286 (2005).

[11] Amazon EC2, http://aws.amazon.com/jp/ec2/ (last access: 2013.4.15).

[12] 北海道大学情報基盤センター, http://www.iic.hokudai.ac.jp (last access: 2013.4.15).

[13] Abel Gordon, Nadav Amit, Nadav Har’El, Muli Ben-Yehuda, Alex Landau, Assaf Shuster, and Dan Tsafir, “ELI: Bare-Metal Performance for I/O Virtualization”, in Proceedings of ASPLOS’12, pp.411-422 (2012).