データセンター内における 高速 VM 間通信用準仮想化ドライバの設計と実装

上野 幸杜† 植原 啓介‡
†慶應義塾大学 政策・メディア研究科 ‡慶應義塾大学 環境情報学部

現在のデータセンタにおける問題の 1 つは、仮想化技術を用いることで得られる柔軟性とその環境下で動作するホストが達成する入出力性能の両立が難しいことである。仮想化技術を用いることで得られる柔軟性と入出力性能を両立するためには、仮想化技術のオーバーヘッドを軽減する必要がある。本研究ではハイパーバイザ型のサーバ仮想化を使用するデータセンタを想定し、データセンタ内部での VM 間通信における 1VM あたりの入出力性能を改善することを目的とする。そのアプローチとして、ゼロコピーや割り込み軽減等のオーバーヘッド軽減手法を適用した準仮想化ドライバを設計・実装した。また、実装した準仮想化ドライバの評価として、通信時の遅延、パケット処理性能及びハイパーバイザの CPU 負荷を計測し、同様の機能を実現可能な既存実装と比較した。計測の結果、本研究で実装した準仮想化ドライバは既存実装と比較して遅延、パケット処理性能、通信時のハイパーバイザにおける負荷の点で優位性があることを示した。

Design and implementation of para-virtualized driver for faster inter-vm communication in data-center

Yukito Ueno† Keisuke Uehara‡
†Keio University, Graduate School of Media and Governance
‡Keio University, Faculty of Environment and Information Studies

Both flexibility the virtualization technology brings and I/O performance of a VM under virtualized environment are incompatible with each other. The reduction technique for overhead of virtualization technology is necessary. The goal of this study is to improve performance of network I/O of an inter-VM communication in data-center. We proposed a para-virtualized driver that is applied overhead mitigation techniques such as zero-copy and interrupt-coalecsing, and designed and implemented that driver to minimize overheads caused by server and network virtualization. In this study, we measured latency, packet processing rate and CPU load of hypervisor of our driver and compared the results with our driver and existing implementation that can realize similar function with our driver. As a result, we found that the para-virtualized driver implemented in this study has advantages over the existing implementation in terms of latency, packet processing rate and CPU load of hypervisor.

1. 背景

現在のデータセンタにおける問題の 1 つは、仮想化技術を用いることで得られる柔軟性とその環境下で動作するホストが達成する入出力性能の両立が難しいことである。主要なクラウドサービス事業者は、保有するデータセンタについてそれぞれ 1 万から 10 万台規模のサーバを有している [1, 2]。Google 社や Yahoo 社など大規模な事業者は、1 つのデータセンタ内で使用するサーバ数が 100 万台規模にのぼる場合もある。このような規模のデータセンタでは、サーバの管理に要するコストが問題となる。データセンタにおける物理的制約及びサーバの管理に要するコストを軽減する為の技術として、仮想化技術が用いられている。

一方で、データセンタ内で動作する仮想化されたホスト (VM) には入出力性能の点で課題がある。入出力性能は、特にデータセンタ内通信を行う際に重要となる。仮想化されたホストの入出力性能が重要となる例として、データセンタ内に構築された Web サービスが挙げられる。近年では、Web サービスフレームワークを用いて構築されることが多い [3]。このとき、サービスフレームワークのバックエンドとして使用される分散データベースシステムやファイルシステム、サービスフレームワークのコンポーネント間通信に要する時間は、データセンタ内のサーバの入出力性能に強く依存する [4, 5]。さらに、遠隔ストレージやデータベースなどのシステムも構成によっては高い入出力性能を要求する。

しかし、仮想化技術を用いることで得られる柔軟性と入出力性能は、トレードオフの関係にある。仮想化技術はホスト運用の柔軟性を高めることができるが、同時に仮想化に起因するオーバーヘッドが発生するため、特に入出力性能が低下する。一方で、入出力性能を優先すると仮想化技術が使用できず、物理的制約や管理コストなどの問題を解決することが難しくなる。仮想化に起因するオーバーヘッドを軽減する多くの技術が開発され、これらの技術を適用することで柔軟性を確保しつつ入出力性能の低下を軽減するための試みが行われている。

2. 本研究の目的

本研究では、ハイパーバイザ型のサーバ仮想化を使用するデータセンタを想定し、データセンタ内部での VM 間通信における 1VM あたりの入出力性能を改善することを目的とする。そのアプローチとして、仮想化環境における通信時の入出力性能に関わる準仮想化ドライバに着目し、仮想化技術による柔軟性を確保しつつ、通信時の入出力性能を向上する手法を提示する。

3. 仮想化技術

サーバ仮想化においてハードウェアを仮想化する方式として、完全仮想化と準仮想化がある。

完全仮想化とは、仮想環境下で既存のハードウェアを完全に再現する技術である。完全仮想化による仮想環境下では、ゲスト OS は変更を加えることなく仮想化されたハードウェア上で動作する。

準仮想化とは、完全仮想化のように既存のハードウェアを完全に再現する代わりに、仮想環境下において必要な機能のみをゲスト OS に提供する技術である。準仮想化による仮想環境下では、実際のハードウェアとしては存在しないハードウェアをゲスト OS に提供するため、準仮想化のための専用ドライバが必要になる。

完全仮想化と準仮想化には、ゲスト OS に対する透過性とそのオーバーヘッドに違いがある。以下に、透過性及びオーバーヘッドの各特徴について完全仮想化と準仮想化の違いを述べる。また、表 1 に完全仮想化と準仮想化の差異を示す。

表 1 完全仮想化と準仮想化の差異

方式	透過性	オーバーヘッド
完全仮想化	あり	大きい
準仮想化	なし	小さい

3.1 透過性

完全仮想化では、ゲスト OS に対して実際のハードウェアと全く同一の振る舞いを再現する。そのため、ゲスト OS に対しては、完全な透過性が提供される。完全仮想化の持つ透過性により、ドライバを含めゲスト OS に対して一切変更を加えることなく仮想環境下で動作させることができるという利点がある。一方で、ゲスト OS は、動作している環境が仮想環境か実環境かを判別できないため、仮想環境に合わせた最適化等を実施することはできない。

準仮想化は、ゲスト OS に対して透過性を提供しない。そのため、仮想化技術として準仮想化を用いる場合、ドライバの追加等、ゲスト OS に対しての変更が必要となる。

3.2 オーバーヘッド

完全仮想化は、準仮想化と比較して仮想化に要するオーバーヘッドが大きい。完全仮想化では、実際のデバイスの挙動を再現するため、ゲスト OS が外部デバイスに対して行う操作をホスト OS が全てトラップし、それに対するエミュレーションを行う。ホスト OS がゲスト OS の挙動をトラップする際には、大きな処理コストを伴う CPU の動作モード変更やコンテキストスイッチ等を行う。また、これらの処理の回数を減らすような最適化は、ゲスト OS による外部デバイスの操作が仮想環境下であることを想定していないため、行われていない場合が多い。

準仮想化は、完全仮想化と比較して仮想化に要するオーバーヘッドが小さい。準仮想化では、実際のデバイスの挙動の再現はせず、VM に対して入出力に必要な機能のみを提供する。そのため、準仮想化による仮想環境におけるオーバーヘッドは、入出力機能を呼び出す際に必要となるもののみである。ただし、入出力機能を呼び出す際には割り込み等のエミュレーションは必要となる。このとき、入出力機能を呼び出すゲスト OS には準仮想化ドライバが組み込まれているため、呼び出しの回数を削減するなど最適化を実施することができる。

4. 関連研究

関連研究として、Ethernet を対象とした準仮想化フレームワークである virtio[6] がある。virtio は、従来のような完全仮想化ではなく、準仮想化を用いることで VM の入出力性能を向上できることを示し、準仮想化デバイスを実装する際のデファクトスタンダードとなった。一方で、virtio フレームワーク上に実装されている準仮想化 Ethernet ドライバである virtio-net には、入出力性能の向上に効果があるとされる送信時の interrupt coalescing や受信時のゼロコピーは実装されていない。

また、K. Salah らは、一般的な Gigabit Ethernet NIC において、polling と通常の割り込みを組み合わせることにより割り込み処理に起因するオーバーヘッドを軽減できることを示している [7]。この手法は Linux において NAPI として実装されており、上で述べた virtio のような準仮想化ドライバにも採用されている。

さらに、10Gbps 以上の VM 間 Ethernet 通信に対する最適化を進めた研究として、NetVM[8] がある。NetVM では、VM 同士がホスト OS に確保された共通のバッファを使用することで、ゼロコピーによる通信を実現する。この研究が提案する手法では、デバイスが DMA を実行するメモリ上の宛先アドレスを、ホスト OS 上の特定領域に固定できるため、Intel Data Plane Development Kit(DPDK)[9] 等既存のパケット処理フレームワークを用いている。ただし、共通のバッファを使用することにより各々の OS が持つネットワークスタックとの整合性を保つ事ができなくなるため、VM のユーザランドからの使用を想定した独自のライブラリを用いた通信に用途が限定される。

一方で、上で述べた研究はいずれもソフトウェアによる解決手法を示したものであり、ハードウェアが持つ機能と準仮想化ドライバを組み合わせる手法は十分に検討されていない。ハードウェアがゼロコピーや割り込み軽減に関する機能を持つことを前提としてソフトウェアを設計することにより、VM の入出力

性能の向上が期待できる。本研究では、以上の機能を持つハードウェアとして Infiniband HCA に着目し、準仮想化ドライバと組み合わせることにより性能の向上が達成できることを示す。

5. アプローチ

5.1 想定環境

1. 章で述べたように、サービスフレームワーク等をデータセンタ上に構築し、その上でサービスを展開している場合、VM 間通信の通信性能は重要となる。サービスフレームワークのパフォーマンスは VM 間通信の通信性能に影響を受ける。また、遠隔ストレージやデータベースなども VM 間通信の通信性能に影響を受ける。

さらに、2009 年時点での典型的な構成では 1 つのハイパーバイザが担当可能な VM 数はおよそ 32 台前後となっており、それぞれ少なくとも 400Mbps 程度の帯域を必要とする [1]。この場合、ハイパーバイザあたりに必要な帯域は 12.8Gbps となるため、10Gbps Ethernet が 1 ポートでは帯域不足となる。近年では 40Gbps Ethernet や Infiniband QDR 等のより高速なインターコネクト技術が登場しているため、前述の要求から将来的にはそれらへの移行が進むと考えられる。従って、本研究では以下のようなデータセンタを想定する。

a). サーバ及びネットワークが仮想化され 100 万台規模の VM が動作する
b). 40Gbps 前後のインターコネクトにより各ハイパーバイザが接続されている

5.2 機能要件

通信時の入出力性能は主にスループットと遅延によって評価されるが、一方の性能を向上させる最適化手法はもう一方の性能を低下させる場合がある。これは、Interrupt Coalescing や Polling などのオーバーヘッド軽減手法を使用した場合に特に顕著となる。そこで、本研究では遅延を既存実装と同程度に保ちつつスループットを最大化するよう最適化する。

また、準仮想化ドライバの機能として、ピーク時の通信性能だけでなく、通信量に対する負荷が妥当な範囲に収まる事も要求される。オーバーヘッド軽減手法として Polling 等を用いる場合、Polling の間隔を狭め過ぎるとピーク時の通信性能が上昇する代わりにほとんど通信をしていない状態でもハイパーバイザに大きな負荷がかかる。このような設計では単一の物理サーバ上で複数の仮想サーバを動作可能にするサーバ仮想化の利点を活かせないため、通信時のハイパーバイザにおける負荷が十分な規模性を持つことが要求される。本研究では、既存実装と同一条件下で VM 間通信を行うとき、ハイパーバイザの負荷が既存実装と同じかそれ以下であれば十分な規模性があるものと見なす。以上をまとめると、本研究で実装する準仮想化ドライバの機能要件は以下の通りである。

1. 遅延が既存実装と同程度かそれ以下であること
2. 最大スループットが既存実装と比較して大きいこと
3. 複数の VM での使用に耐える十分な規模性を持つこと

6. 設計と動作概要

本研究で実装する準仮想化ドライバの概要を図 1 に示す。この準仮想化ドライバは、ハードウェアの制御及びゲスト OS との入出力を行うホスト OS 上のヘルパ及びゲスト OS 上で動作する仮想 Ethernet ドライバから成る。

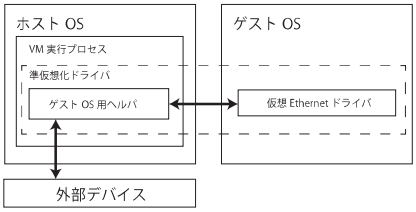

図 1 本研究で実装する準仮想化ドライバの概要

本研究では通信時の入出力性能を改善することを目的としているため、準仮想化ドライバのうちネットワークデバイスとしての動作をするものを設計・実装する。通信時の入出力に関わるオーバーヘッドを軽減するため、以下の手法を適用する。

- Interrupt Coalescing
  Interrupt Coalescing は、ネットワークデバイスの割り込みの回数を抑制し、割り込みの回数を減らす手法である。本来は 1Gbps もしくは 10Gbps の Ethernet NIC が採用した手法だが、virtio-net などの仮想ネットワークデバイスでも用いられている。ネットワークデバイスが設定した閾値を超えるパケット着信に対して一度だけ割り込みを挿入することにより、割り込み回数を軽減する。また、パケット着信数が閾値に満たない場合でもクロック数など一定時間で自動的に割り込みを挿入することで、パケットの着信遅れを防ぐ。
- Polling
  Polling は、ネットワークデバイスによる割り込みを無効化し、受信するホストが定期的にパケットの着信を確認する手法である。割り込みを無効化することにより、割り込みに起因するオーバーヘッドを削減することができる。また、ホストは他の処理を中断せずに任意のタイミングでパケット着信処理を行うことができるため、コンテキストスイッチの回数及びそれに伴うオーバーヘッドを減らすことができる。
- ゼロコピー
  ゼロコピーは、CPU によるメモリ上のデータのコピーを避けて処理を実行させる設計手法である。通信時の入出力においては、外部デバイスが最終的にそのパケットを利用するプロセスの用

意したメモリ空間に対して DMA を行うことにより実現する。ゼロコピーでの処理が可能な場合、処理過程でメモリコピーに要するオーバーヘッドが削減できるため、より高速な通信が実現できる。しかし、ゼロコピー通信の実現可否はゲスト OS 及びホスト OS、外部デバイスのアーキテクチャに大きく依存する。特に、ネットワーク通信における受信側のゼロコピーは、デバイスへのパケット到着順序が予測困難であるため、受信者を単一のプロセスに限定するか、デバイスが複数の DMA キューを用意しパケットの内容に応じてキューを選択するなどの機能を持つ必要がある。

以下では、前述したオーバーヘッド軽減手法を適用した準仮想化ドライバの設計及び動作概要について述べる。準仮想化ドライバの動作は初期化・送信処理・受信処理に大別できるため、以下ではゲスト OS からパケットを送信する動作を送信側、ゲスト OS がパケットを受信する動作を受信側とし、それぞれの処理について述べる。

6.1 初期化処理

初期化時の動作を図 2 に示す。本準仮想化ドライバの初期化手順は大きくわけて 6 つの手順からなる。以下に 6 つの手順を示す。

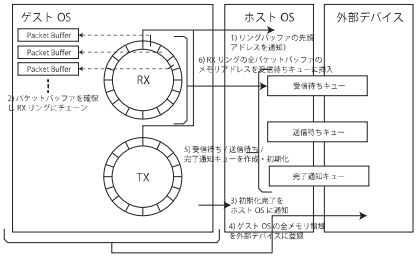

図 2 初期化処理の動作概要

1. リングバッファの先頭アドレスの通知
   ゲスト OS は、起動シーケンスの途中でホスト OS とデータを共有するための固定長リングバッファをパケット送信用と受信用の計 2 つ確保する。このリングバッファは、送受信用パケットバッファのメモリアドレスを共有するために使用する。リングバッファの確保が完了次第、ゲスト OS はリングバッファのメモリアドレスをホスト OS に通知する。このとき、ゲスト OS とホスト OS 間ではメモリアドレス空間が異なるため、ゲスト OS は自身の物理メモリアドレス (Guest Physical Address) を通知し、ホスト OS ではゲスト OS 用に確保したメモリ領域のオフセットと通知された物理メモリアドレスからゲスト OS の物理メモリアドレスをホスト OS 上の仮想メモリアドレス (Host Virtual Address) へ変換する。
2. パケットバッファの確保及びリングバッファの更新
   リングバッファの通知処理完了後、ゲスト OS は受信用リングバッファのエントリ数分のパケットバッファを確保し、全てのパケットバッファをリングバッファに紐づける。
3. 初期化完了をホスト OS に通知
   送受信用リングバッファの確保及び受信用リングバッファの充填が完了すると、ゲスト OS は初期化完了をホスト OS に通知する。
4. ホスト OS による外部デバイスへのメモリ領域登録
   外部デバイスの初期化を行うためには、パケットの送受信に使用するメモリ領域の登録が必要である。ユーザランドへのゼロコピーが可能なアーキテクチャでは、この登録を行うことによりユーザランドのメモリ空間に対する DMA を実現する。ゲスト OS が確保するパケットバッファ領域は一定ではないため、本研究で実装する準仮想化ドライバではホスト OS がゲスト OS の物理メモリとして確保した全領域をパケットバッファのためのメモリ領域として登録する。
5. 送受信キュー及び完了通知キューの作成・初期化
   メモリ領域の登録が完了した後、ホスト OS はパケットの送受信を行うためにゲスト OS 毎に送信待ち/受信待ちキュー及び完了通知キューを用意する。
6. リングバッファの全パケットバッファのメモリアドレスを受信キューに挿入
   上記手順が完了した後、ホスト OS はゲスト OS があらかじめ受信用リングバッファに充填したパケットバッファのメモリアドレスを含むエントリを作成し、外部デバイスの受信待ちキューに挿入する。

6.2 送信処理

送信処理の動作概要を図 3 に示す。本準仮想化ドライバの送信処理は、大きくわけて 6 つの手順からなる。以下に 6 つの手順を示す。

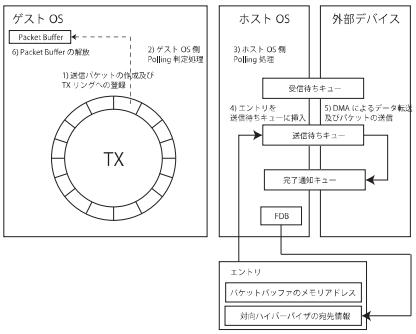

図 3 送信処理の動作概要

1. 送信パケットの作成
   ゲスト OS の仮想 Ethernet ドライバは、送信パケットを作成し自身の TX リングにバッファを登録する。
2. ゲスト OS 側 Polling 判定処理
   本準仮想化ドライバでは、ゲスト OS がホスト

OS の送信処理を要求する際に、オーバーヘッド軽減手法である Polling を用いる。ゲスト OS は、ホスト OS の状態が「通知可能」ならば、ホスト OS に対してパケットの送信を通知する。ホスト OS の状態が「通知不可能」であれば、ホスト OS に対して何もしない。ホスト OS の状態が「通知可能」である場合、ホスト OS 上ではタイマーによる送信側 Polling が動作していない。このため、ゲスト OS からの能動的な通知を必要とする。

3. ホスト OS 側 Polling 処理
   ホスト OS は、ゲスト OS から送信待ちパケットの存在を通知されると、Polling 処理を起動し、ゲスト OS からの通知を不可にした上でパケット送信処理を行う。その後、新たな送信待ちパケットは Polling による定期処理によって確認される。これにより、ゲスト OS からホスト OS へのシグナリングの頻度を軽減することができる。
4. 送信パケットのエントリを送信待ちキューに挿入
   ホスト OS は宛先ハイパーバイザを決定し、ホスト OS は送信パケットバッファのメモリアドレスと宛先ハイパーバイザをまとめ、エントリとして外部デバイスの送信待ちキューに挿入する。
5. DMA によるデータ転送及びパケットの送信
   外部デバイスはエントリが登録され次第エントリ内のメモリアドレスを参照し、DMA によってデータを読み出した後パケットとして送信する。
6. ゲスト OS によるパケットバッファの解放
   ゲスト OS は、ホスト OS からパケットの送信完了通知を受けた後、送信に使用したパケットバッファを解放する。

6.3 受信処理

受信処理の動作概要を図 4 に示す。本準仮想化ドライバの受信処理は、大きくわけて 6 つの手順からなる。以下に 6 つの手順を示す。

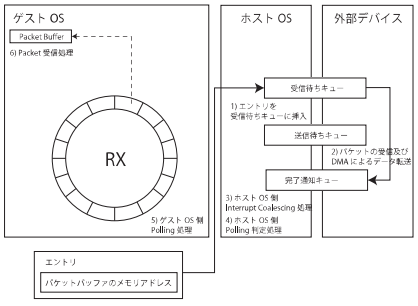

図 4 受信処理の動作概要

1. 受信パケットバッファのエントリを受信キューに挿入
   ホスト OS は、適当なタイミングでゲスト OS とのデータ共有リングバッファを確認し、新しい受信用パケットバッファが登録されていれば外部デバイスの受信待ちキューにパケットバッファのメモリアドレスをエントリとして挿入する。
2. パケットの受信及び DMA によるデータ転送
   パケットが着信すると、外部デバイスが受信キューに挿入されたエントリに従い、DMA によってデータをパケットバッファに転送する。
3. ホスト OS 側 Interrupt Coalescing 処理
   本準仮想化ドライバでは、ホスト OS 側の受信処理において、オーバーヘッド軽減手法である Polling 及び Interrupt Coalesing を適用する。パケット受信後、ホスト OS はゲスト OS に対して割り込みを行う。このとき、Interrupt Coalesing によって割り込みがキャンセルされる場合がある。InterruptCoalesing は時間とパケット着信数によって割り込みの頻度を制御する。パケット着信時点で Interrupt Coalesing が起動していない場合、ホスト OS は Interrupt Coalesing を起動し、着信したパケットは Polling による割り込み判定に移行する。Interrupt Coalesing が起動している場合、次回の割り込みタイミングまで何もしないか、最後の割り込み時から閾値を超えるパケット数が到着した場合は Polling による割り込み判定に移行する。同様に、割り込みがキャンセルされた場合、タイマーにより一定時間で Polling による割り込み判定に移行する。
4. ホスト OS 側 Polling 判定処理
   Interrupt Coalesing による割り込み判定プロセスを経て、ホスト OS は Polling による割り込み判定を行う。この処理は、送信処理と同様である。
5. ゲスト OS 側 Polling 処理
   ゲスト OS は、ホスト OS から受信パケットの存在を通知されると、Polling 処理を起動し、ホスト OS からの通知を不可にした後、パケット受信処理を行う。その後、新たな受信パケットは Polling による定期処理によって確認される。
6. ゲスト OS によるパケット受信処理
   ゲスト OS は、ホスト OS によりパケットの受信を通知されるか、Polling 処理により未受信のパケットを検知すると、通常のネットワークスタックにパケットをインジェクトする。パケットの受信処理はその後ゲスト OS のネットワークスタックでの処理に移行する。

7. 実装

本節では、本研究で実装した準仮想化ドライバの実装の概要を述べる。実装の概要を図 5 に示す。また、実装を行った環境を表 2 に示す。

表 2 本研究の準仮想化ドライバの実装環境

ホスト OS	Scientific Linux 6.4
ゲスト OS	Debian GNU Linux 7.1 wheezy
ホスト OS カーネル	2.6.32-358.14.1
ゲスト OS カーネル	3.10.19
使用言語	C 言語 (GNU C Compiler)
qemu	1.6.0
OFED	2.0-3.0.0

本準仮想化ドライバは、ホスト OS 側のユーザラン

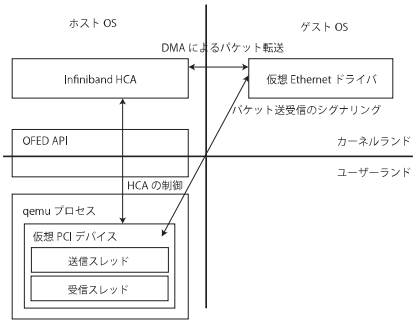

図 5 実装の概要

ドにおいて動作する仮想 PCI デバイス及びゲスト OS 側のカーネルランドにおいて動作する仮想 Ethernet ドライバとして実装されている。ホスト OS 側の仮想 PCI デバイスは、qemu のモジュールとして実装され、送信側処理と受信側処理は、それぞれスレッドとして独立して動作する。また、ゲスト OS 側の仮想 Ethernet ドライバは、Linux カーネルで動作するデバイスドライバとして実装されている。

本準仮想化ドライバの実装には、ハイパーバイザ間の通信に使う物理メディアとして Infiniband を採用した。Infiniband を用いることにより、着信したパケットを直接ゲスト OS の物理メモリ領域に対して直接 DMA 転送することができる。Infiniband HCA の制御はホスト OS 側仮想 PCI デバイスが担当し、制御用 API として OpenFabrics Enterprise Distribution(OFED)[10] を用いる。ホスト OS 側仮想 PCI デバイスドライバは、初期化処理として各種キューの作成、ゲスト OS の物理メモリ領域の登録及び共有リングバッファを介したゲスト OS とのパケット送受信に関するシグナリングを行う。ただし、実際のパケットデータは、共有リングバッファを介して通知されたゲスト OS の物理メモリ上の領域に直接 DMA 転送される。

ゲスト OS 側仮想 Ethernet ドライバは、共有リングバッファの確保を行う。また、通信時は共有リングバッファを用いてホスト OS とのパケット送受信に関するシグナリングを行い、受信したパケットをカーネルのネットワークスタックに渡す。

8. 評価

8.1 評価項目

5.2 節で示した通り、機能要件は 1) 遅延が既存実装と同程度かそれ以下であること 2) 最大スループットが既存実装と比較して大きいこと 3) 複数の VM での使用に耐える十分な規模性を持つこと、の 3 つである。本研究では、実装した準仮想化ドライバが 1)、2)、3) の機能要件を満たしていることを確認するため、以下で述べる点を計測項目とする。

8.1.1 1VM あたりの VM 間通信時の Round Trip Time

本研究で実装した準仮想化ドライバには、Interrupt Coalescing や Polling などを採用しており、最大スループットを大きくすることに主眼を置いている。しかし、これらの最適化手法は遅延を増加させる可能性がある。準仮想化ドライバの性能として遅延とスループットは重要なメトリックであるため、遅延が大きくなりすぎた場合、結果として本研究の目的であるデータセンタ内部での VM 間通信における 1VM あたりの入出力性能を改善することはできない。そのため、本研究で実装したドライバと既存実装の性能を遅延の点で評価する。評価は、2 台のハイパーバイザ上でそれぞれ 1 台ずつ VM を動作させ、VM 間通信を行った際の Round Trip Time を計測することによって行う。

8.1.2 1VM あたりの VM 間通信時のパケット処理性能

パケット処理性能は準仮想化ドライバの性能として重要なメトリックである。本研究ではパケット処理性能を増加させるためのオーバーヘッド軽減手法を複数適用しているため、これらの手法によって VM 間通信における 1VM あたりの入出力性能を改善できているかを評価する。評価は、2 台のハイパーバイザ上でそれぞれ 1 台ずつ VM を動作させ、VM 間通信を行った際の最大スループット (bps) 及び最大パケット処理性能 (pps) を計測することによって行う。

8.1.3 1VM が一定レートで通信する際のハイパーバイザの CPU 負荷

5.2 節で述べたように、準仮想化ドライバを使用する際にハイパーバイザにおける負荷が通信量に応じた妥当なものであることが要求される。そのため、ハイパーバイザ上で VM が一定レートで通信するとき、どの程度ハイパーバイザに CPU 負荷がかかるかを評価し、これを準仮想化ドライバの規模性の指標とする。評価は、2 台のハイパーバイザ上でそれぞれ 1 台ずつ VM を動作させ、一定レートで VM 間通信を行った際のハイパーバイザの CPU 負荷を計測することによって行う。

8.2 実験計画

本節では、8.1 節で挙げた計測項目について実験を行うための計画について述べる。5.1 節で述べた通り、本研究では a) サーバ及びネットワークが仮想化され 100 万台規模の VM が動作する b)40Gbps 前後のインターコネクトにより各ハイパーバイザが接続されている、の 2 点を想定している。本研究の目的は、データセンタ内部での VM 間通信における 1VM あたりの入出力性能を改善することであるため、これらの想定下で 1VM あたりの通信性能を改善できているかを評価する。ただし、a) に関しては、本準仮想化ドライバがハイパーバイザの総数によって性能が変動するような機能を持たないため、VM 間通信時の最小構成であるハイパーバイザ 2 台を再現することによって計測する。また、b) に関しては、40Gbps 前後のインターコネクトが必要となる。

本研究では、同一の環境下において同一の機能を実現可能な既存実装として vhost-net[11]、及び eIPoIB[12] によって構築したシステムを用いる。

vhost-net、eIPoIB 及びその前提となる IPoIB については 8.4.2 節で述べる。他の実装では、同一のハードウェア環境下において動作しないか、または同一の機能を実現できない。ただし、想定環境及び機能要件の違いから本研究の比較対象にはならないものの、通信の多重化を行いつつ VM に外部接続性を提供する手法としては、Ethernet 上で vhost-net のみを用いる手法や、Infiniband HCA を PCI passthrough により VM に提供する手法等がある。

8.3 実験環境

本実験で構築する実験環境のトポロジを図 6 に示す。前述の実験環境に対する要求を満たすため、本研究では x86-64 アーキテクチャのサーバ上で Linux と qemu-kvm によるサーバの仮想化環境を構築する。qemu-kvm はオプションが非常に多様であるため、libvirt[13] のフロントエンドである virsh を用いて VM の起動オプションを制御する。2 台のハイパーバイザはいずれも Intel-VT を搭載したものを使用し、仮想化支援機能を有効化する。さらに、ハイパーバイザのプロセススケジューリングによる計測誤差を避けるため、VM には 1 つの CPU コアのみを割り当て、CPU コアはハイパーバイザ上の特定のコアに固定割り付けとする。使用する機材のスペックを表 3 に、VM に割り当てるリソースを表 4 に示す。また、使用するソフトウェアのバージョン等を表 5 に示す。

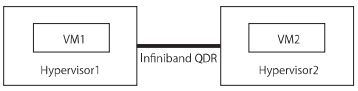

図 6 実験環境のトポロジ

表 3 使用する機材のスペック

	Hypervisor1	Hypervisor2
CPU	Intel Xeon E5607	Intel Corei7 920
コア数	4	4
メモリ	32GB	12GB
HCA	Mellanox MT26428	Mellanox MT26428

表 4 VM に割り当てるリソース

	VM1	VM2
CPU	QEMU Virtual CPU	QEMU Virtual CPU
コア数	1	1
メモリ	1GB	1GB

表 5 使用するソフトウェアのバージョン

	VM1	VM2
OS	Scientific Linux 6.4	Scientific Linux 6.4
Linux kernel	2.6.32	2.6.32
OFED	2.0-3.0.0	2.0-3.0.0
qemu	1.6.0	1.6.0
virsh	1.1.2	1.1.2

また、本実験ではハイパーバイザ間のインターコネクトとして、Infiniband QDR を用いる。Infiniband QDR は 40Gbps のインターコネクト技術であり、8b/10b 符号化方式 [14] を用いるため理論上のデータ転送レートは 32Gbps である。Infiniband QDR の HCA は各ハイパーバイザに PCI Express Gen2.0 5GB/s にて接続する。また、HCA 間はダイレクトアタッチケーブルを用いて直接接続する。

以上の物理実験環境上で、準仮想化ドライバを用いたシステムを構築し、実験対象とする。8.1 節で述べたように、本実験では同一の環境下において同一の機能を実現可能な既存実装として vhost-net + eIPoIB によって構成されるシステムを用いる。現時点では、この組み合わせのみが、安価に 40Gbps 前後のインターコネクトを実現可能な Infiniband を使用してゲスト OS に Ethernet を提供する現実的な手法である。

8.4 実験に使用するシステムの構築

8.4.1 本研究で実装した準仮想化ドライバ

本研究で実装した準仮想化ドライバを用いて構築するシステムの概要を図 7 に示す。ホスト OS 側では、ゲスト OS の起動時に qemu に対してオプションを与えることにより、仮想 PCI デバイスの動作が開始する。一方、ゲスト OS 側では準仮想化ドライバの一部として実装した専用の仮想 Ethernet ドライバを読み込む必要がある。これにより、ゲスト OS は通常の物理インターフェースと同様に準仮想化ドライバを介した Ethernet 通信を利用可能になる。本実験では、このインターフェースを使用して通信を行い、通信性能を計測する。

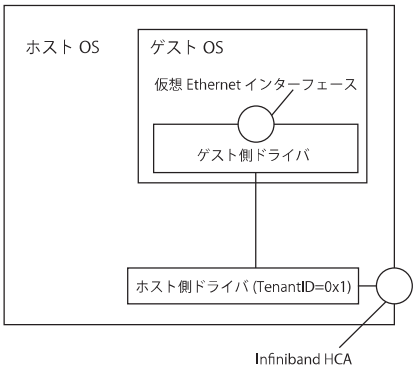

図 7 本研究で実装した準仮想化ドライバを用いて構築するシステムの概要

8.4.2 vhost-net + eIPoIB により構成されるシステム

本節ではまず、構築するシステムにおいて使用する、vhost-net、IPoIB 、及び IPoIB のプロトコルを用いて実現される eIPoIB のそれぞれについて詳細を述べる。

- vhost-net
  vhost-net は、virtio フレームワークを使用した準仮想化ドライバである。vhost-net は Linux カーネルモジュールとして実装されており、qemu が VM を起動後ネットワークインターフェースなどのパラメータをカーネル内の vhost-net に渡すことで動作する。
- IPoIB
  IPoIB[15] は、Infiniband ネットワーク上で仮想

的に IP ネットワークを構築するプロトコルである。OpenFabrics Alliance(OFA) による Linux 上のカーネルモジュールとしての実装がある。IPoIB を有効にすると IP アドレスを付与可能な仮想 PPP インターフェースが作成される。この仮想インターフェースに対してパケットを送受信することで Infiniband ネットワーク上で IP による通信を行う。

- eIPoIB
  eIPoIB は、Infiniband ネットワーク上に構築された IPoIB ネットワークを Ethernet ネットワークとして使用可能にする技術である。eIPoIB は主に Infiniband を準仮想化ドライバなどの仮想化技術と組み合わせて使用する目的で提案・実装された。eIPoIB は、ホスト OS 内に仮想 Ethernet インターフェースを作成する。仮想 Ethernet インターフェースに Ethernet フレームが着信すると、eIPoIB は Ethernet フレームの Ethernet ヘッダを取り除き、取り除いた Ethernet ヘッダを基に作成した IPoIB ヘッダを付加する。外部から IPoIB パケットの着信があった場合、IPoIB ヘッダを取り除き Ethernet ヘッダを付加する。この一連の動作は eIPoIB の提供する Shim レイヤとして実装されている。

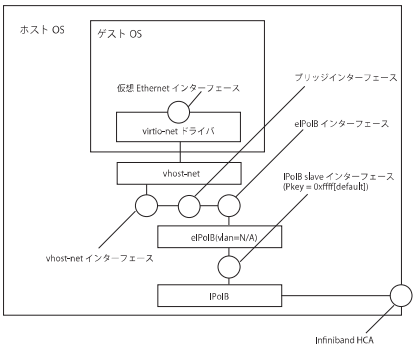

図 8　vhost-net ＋ eIPoIB を用いて構築するシステムの概要

以上の技術を用いて構築するシステムの概要を図 8 に示す。ホスト OS において、vhost-net は仮想 Ethernet インターフェースを作成し、そのインターフェースを通してゲスト OS にパケットを配送する。また、eIPoIB は仮想 Ethernet インターフェースを作成し、そのインターフェースに対してパケットを送受信することで通信を行う。そこで、Linux カーネルが提供するブリッジ機能を使用して vhost-net と eIPoIB を相互接続する。eIPoIB は所属する IPoIB インターフェースとして IPoIB slave インターフェースを作成し、IPoIB slave インターフェースに対して Partition Key を設定する。今回の計測では、IPoIB slave インターフェースの Partition Key としてデフォルトの 0xffff を使用する。

ゲスト OS 側では、vhost-net のゲスト用ドライバが必要となる。本研究で実装した準仮想化ドライバと同様に、ホスト側 vhost-net はゲスト OS にとって仮想 PCI デバイスと同様に振る舞う。そのため、仮想 PCI デバイスドライバを動作させることが必要となる。ゲスト OS 側では、本研究で実装した準仮想化ドライバと同様に仮想 PCI デバイスドライバにより Ethernet のインターフェースが使用可能となる。本実験では、このインターフェースを使用して通信を行い、通信性能を計測する。

8.5　計測

8.5.1　Round Trip Time

Round Trip Time(RTT) は、ゲスト OS 間で ping を用いて計測した。ping を用いて 1 秒に一回のパケット送受信を計 20 回行い、その平均値を結果とした。また、計測はフレームサイズを変え、フレームサイズが 64/128/256/512/1024/1280/1518 バイトのそれぞれの場合について行った。計測の結果を図 9 に示す。

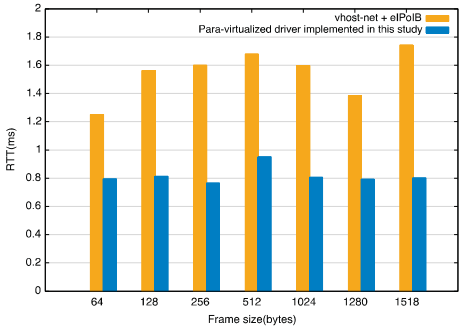

図 9　各システムを使用した際の Round Trip Time

結果から、vhost-net ＋ eIPoIB の場合、フレームサイズによる RTT の変動が大きいことがわかった。フレームサイズが 1024 バイト及び 1280 バイトの時 RTT は減少しているが、おおむねフレームサイズが大きくなるにつれて RTT が増加している。vhost-net ＋ eIPoIB は、ゼロコピーではなく途中経路で複数回フレームのコピーが行われるため、そのオーバーヘッドが反映されていると考えられる。また、その過程でホスト OS のネットワークスタックを通過するため、ホスト OS のスケジューリングなどによって RTT に揺れが生じていると考えられる。

対して、本研究で実装した準仮想化ドライバは、フレームサイズ 64 バイトから 1518 バイトまで RTT が 1ms 以内に収まっている。フレームサイズによる RTT の変動が少ないのは、ゼロコピーを使用しているため、フレームサイズが大きい場合でもメモリコピーによるオーバーヘッドがないためであると考えられる。また、ゼロコピーであるため、シグナルパスではホスト OS のスケジューリングの影響を受けるものの、データパスにおいてはホスト OS のスケジューリングの影響を受けていないと考えられる。比較結果から、オーバーヘッド軽減手法のうち polling や Interrupt Coalesing など遅延を増加させる手法を用いた場合でも、ゼロコピーなど他の手法を組み合わせた場合実用的な範囲に影響を抑えられることがわかった。

8.5.2　パケット処理性能

パケット処理性能は、ゲスト OS のカーネルランドからトラフィックを生成することで計測した。トラフィックの生成は 60 秒間行い、対向のゲスト OS が受信したトラフィック量から 1 秒あたりのパケット数及びトラフィック量を算出し結果とした。計測は MTU 及びそのとき生成するフレームサイズを変え、MTU 及びフレームサイズが 64/128/256/512/1024/1280/1518 バイトのそれぞれの場合について行った。計測の結果のうち pps 単位のものを図 10 に、bps 単位のものを図 11 に示す。

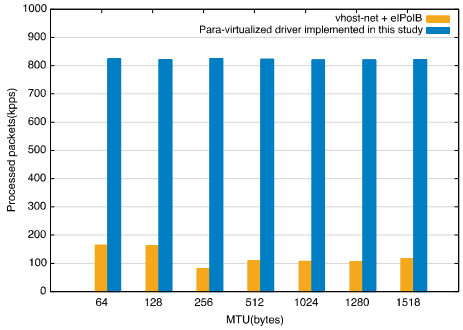

図 10　各システムのパケット処理性能 (pps)

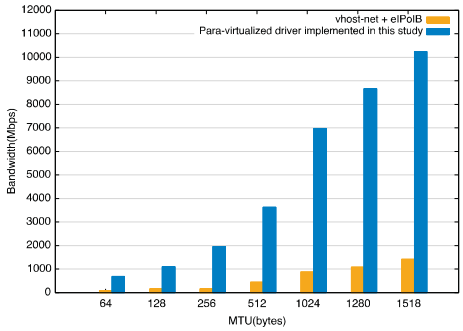

図 11　各システムのパケット処理性能 (bps)

結果から、vhost-net + eIPoIB の場合、MTU が 256 から 1518 バイトの場合、64 バイトから 128 バイトの場合と比較してわずかにパケット処理性能が低いことがわかった。トラフィックの処理過程でメモリコピーが発生するため、フレームサイズに応じてパケット処理性能が低下していると考えられる。

対して、本研究で実装した準仮想化ドライバは、MTU によってほとんど pps が変動しない。トラフィックの処理過程にメモリコピーがないため、フレームサイズに応じてパフォーマンスの変動がないものと考えられる。

8.5.3　ハイパーバイザの負荷

ゲスト OS 間通信の際のハイパーバイザの負荷は、8.5.2 節と同様の方法でゲスト OS 間においてトラフィックを生成し、その際のハイパーバイザにおける CPU 負荷を計測することで行った。トラフィックの生成は 60 秒間行い、そのうち前後 20 秒間を除いた 20 秒間ハイパーバイザにおいて CPU 負荷を取得し、その平均値を結果とした。ハイパーバイザはマルチコアの CPU を搭載しているため、全てのコアの合計を 100% とした。

なお、8.5.2 節の結果よりゲスト OS が生成可能なトラフィックは各システムによって上限に差があるため、各 MTU でのスループット計測の結果から上限の低い値を 50Mbps 単位で切り捨て、その数値を上限としてトラフィックを生成した。送信側ハイパーバイザの結果を図 12 に、受信側ハイパーバイザの結果を図 13 に示す。

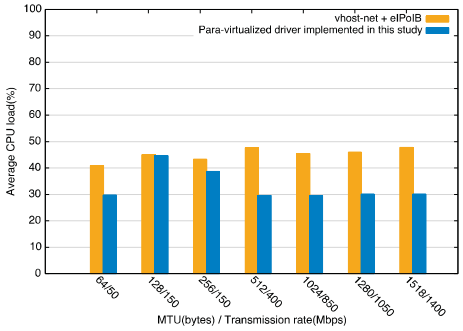

図 12　各システムを使用した際のハイパーバイザの CPU 負荷 (送信側)

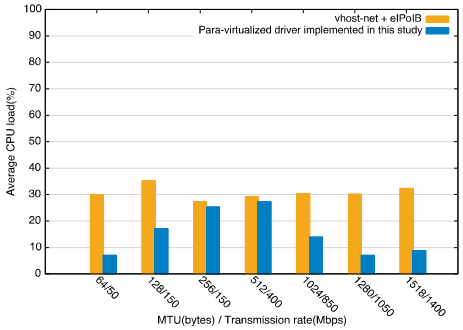

図 13　各システムを使用した際のハイパーバイザの CPU 負荷 (受信側)

結果から、本研究で実装した準仮想化ドライバは vhost-net + eIPoIB のシステムと比較して、送信側及び受信側の両方において一貫してハイパーバイザにおける CPU 負荷が低いことがわかった。8.5.2 節と同様に、ゼロコピーによるオーバヘッド軽減効果があるものと考えられる。

8.6　実験のまとめと考察

Round Trip Time に関する実験の結果から、本研究で実装した準仮想化ドライバは VM 間通信においてフレームサイズに関わらず 1ms 以下での通信を実現できることを示した。また、パケット処理性能に関する実験の結果から、本研究で実装した準仮想化ドライバは MTU に関わらず常に 820kpps 前後のパケット処理性能を出せることを示した。さらにハイパーバイザにおける CPU 負荷の計測結果から、通信時のハイパーバイザにおける CPU 負荷が実用的な範囲に留まることを示した。

また、本研究では既存の手法との比較として vhost-net + eIPoIB で構成したシステムについても同様の

計測を行った。結果から、本研究で実装した準仮想化ドライバは、vhost-net + eIPoIB で構成したシステムと比較して、全ての計測項目において優位性があることがわかった。

9. 結論

本研究では、データセンタ内部での VM 間通信における 1VM あたりの入出力性能を改善することを目的として、準仮想化ドライバを設計・実装した。サーバ仮想化における準仮想化ドライバのオーバーヘッド軽減手法として Interrupt Coalescing、Polling、ゼロコピーを適用し、それを実現するための物理メディアとして Infiniband QDR を用いた。

本研究で実装した準仮想化ドライバにより、データセンタ内部での VM 間通信における 1VM あたりの入出力性能を改善できていることを示すため、Infiniband QDR によって相互接続されたハイパーバイザにより、データセンタ内のサーバ仮想化環境を模擬した環境を構築し評価を行った。評価は、遅延、パケット処理性能、通信時のハイパーバイザにおける負荷を既存手法と比較することによって行い、その結果既存手法と比較して低い RTT 及びハイパーバイザへの負荷を維持しつつ、スループットが 600kpps 以上向上することを示した。本手法をデータセンタ内のハイパーバイザ及び VM に導入することにより、仮想化環境の柔軟性を確保しつつ、入出力性能のオーバーヘッドを軽減できる。

10. 今後の展望

本研究では、Infiniband HCA がハードウェアとして提供する機能に着目し、それを前提としてソフトウェアを設計することで VM の入出力性能を改善できることを示した。しかし、現在のデータセンタ内では Infiniband だけではなく Ethernet の普及も進んでいる。1. 節で述べたように、現在の仮想化されたデータセンタが抱える問題は仮想化技術によって柔軟性を確保した一方で入出力性能が低下することである。この問題は、データセンタで使用する物理メディアに関わらず発生する。本研究で設計・実装を行った準仮想化ドライバは Infiniband アーキテクチャが提供する機能に依存して動作するが、今後は物理メディアに関わらずオーバーヘッド軽減手法を適用した準仮想化ドライバを作成しやすくする工夫が必要である。

参考文献

[1] E. Bernstein, D.; Ludvigson. Networking challenges and resultant approaches for large scale cloud construction. In *Grid and Pervasive Computing Conference, 2009. Workshops*, GPC '09, 2009.

[2] H. Liu. Amazon data center size. `http://huanliu.wordpress.com/2012/03/13/amazon-data-center-size/` (Retrieved: 2014-09-04).

[3] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. A view of cloud computing. *Commun. ACM*, 53(4):50–58, 2010.

[4] Stephen M Rumble, Diego Ongaro, Ryan Stutsman, Mendel Rosenblum, and John K Ousterhout. It' s time for low latency. In *Proceedings of the 13th USENIX conference on Hot topics in operating systems*. USENIX Association, 2011.

[5] Chuanxiong Guo, Guohan Lu, Dan Li, Haitao Wu, Xuan Zhang, Yunfeng Shi, Chen Tian, Yongguang Zhang, and Songwu Lu. Bcube: A high performance, server-centric network architecture for modular data centers. In *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication*, pages 63–74, New York, NY, USA, 2009. ACM.

[6] Rusty Russell. Virtio: Towards a de-facto standard for virtual i/o devices. *SIGOPS Oper. Syst. Rev.*, 42(5):95–103, 2008.

[7] K. Salah, K. El-Badawi, and F. Haidari. Performance analysis and comparison of interrupt-handling schemes in gigabit networks. *Computer Communications*, 30(17):3425 – 3441, 2007.

[8] Jinho Hwang, KK Ramakrishnan, and Timothy Wood. Netvm: high performance and flexible networking using virtualization on commodity platforms. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2014.

[9] Intel Corporation. Intel data plane development kit: Getting started guide. Technical report, 2013.

[10] Ofed overview. `https://www.openfabrics.org/index.php/resources/ofed-for-linux-ofed-for-windows/ofed-overview.html` (Retrieved: 2014-09-04).

[11] Usingvhost. `http://www.linux-kvm.org/page/VhostNet` (Retrieved: 2014-09-04).

[12] A. Ali. Ethernet tunneling over ipoib. In *2012 OpenFabrics International Workshop*, 2012.

[13] The virtualization api(libvirt). `http://libvirt.org/` (Retrieved: 2014-09-04).

[14] Widmer; Albert X. Franaszek; Peter A. Byte oriented dc balanced (0,4) 8b/10b partitioned block transmission code, 1984.

[15] J. Chu and V. Kashyap. Transmission of ip over infiniband (ipoib). RFC 4391, IETF, 2006.