Vol.2009-ARC-183 No.13
Vol.2009-OS-111 No.13
2009/4/23

仮想計算機における入出力命令の一括処理による入出力の高速化

新城　靖[†1]　高橋由直[†1]

この論文は、完全仮想化に基づく仮想計算機における入出力を高速化する方法を提案している。提案方式では、ゲストＯＳのデバイス・ドライバに含まれる入出力を行う機械語命令を静的に書き換え、複数の出力命令を一括に処理する。これにより、仮想計算機とホストＯＳの切替え回数が削減され、入出力が高速化される。提案方式は、仮想計算機モニタ Linux KVM で実現されている。提案方式をネットワークに対するデバイス・ドライバに対して適用した所、最大でネットワークのスループットが Linux で 5.5 倍、FreeBSD で 4.9 倍に高速化された。

Accelerating I/O Performance by Batched Execution of I/O Instructions in Virtual Machines

Yasushi Shinjo[†1] and Yoshinao Takahashi[†1]

This paper proposes an acceleration method of I/O operations in virtual machines based on full virtualization. The proposed method statically rewrites I/O instructions in device drivers of guest operating systems in order to execute multiple I/O instructions together. This reduces the number of switches between a guest operating system and a host operating system, and improves I/O performance. The proposed method has been implemented in the virtual machine monitor KVM of Linux. By applying the proposed method to a network device driver, network throughputs were sped up by up to 5.5 times in Linux and 4.9 times in FreeBSD.

†1 筑波大学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, University of Tsukuba, Japan

1. はじめに

仮想計算機は、様々なタイプの計算機において、主要なシステム要素の 1 つになりつつある。メインフレームでは、従来、高価なハードウェアを効率的に共有するために利用されていた。コモディティのハードウェアを用いるサーバ計算機においては、1 つのサービスを提供するためには CPU 性能が高すぎるということがある。このような場合でも、仮想計算機を用いて複数のサーバを 1 台の物理計算機に集約することにより、高い CPU 性能を有効に利用できるようになる。個人用計算機では、ある OS(Operating System) 用に開発された応用プログラムを別の OS で実行することを目的として仮想計算機がしばしば利用される。

x86 アーキテクチャのハードウェアで動作する仮想計算機では、仮想化のオーバヘッドが大きいという問題があった[12)]。VMware では、この問題を機械語命令を動的に書き換えることで、実用的な性能を得ている[1)]。Xen では、準仮想化 (paravirtualization) により CPU の仮想化のオーバヘッドを低減している[4)]。その後、Intel VT や AMD-V のようなハードウェアによる仮想化支援機能が x86 アーキテクチャでも利用可能になり、CPU 仮想化のオーバヘッドはかなり低減された[17)2)]。その結果、CPU 処理を中心とする応用プログラムを実行する場合、仮想計算機のオーバヘッドは問題にならない程度になってきた。

ハードウェアによる仮想化支援機能が利用可能になったとしても、仮想計算機の入出力性能が低いという問題は残されている。その原因としては、次のようなものがある。

- デバイスのエミュレーションが重たい。
- 入出力命令の実行や割り込み処理で発生する、仮想計算機のゲスト OS とエミュレータが動作する環境 (多くの場合はホスト OS) との切り替えのオーバヘッドが大きい。
- 様々な抽象化の層の間でデータのコピーが発生する。

入出力についても、準仮想化により高速化することは一般的に行われている[4)13)]。準仮想化により、重たいエミュレーションが回避され性能は大きく改善されるが、準仮想ドライバ (paravirtual driver) の開発コストが大きいという問題が残っている。

本研究では、仮想計算機における入出力性能が低いという問題のうち、上記の２番目の原因に着目し、入出力を高速化する方法を提案する[16)]。具体的には、入出力命令の実行に伴い生じるゲスト OS とホスト OS の切り替え回数を、入出力命令を一括処理することにより削減する。本方式では、まず既存のデバイス・ドライバのソース・プログラムを入力とし、それに含まれている入出力命令をライブラリ関数呼び出しへ書き換える。そのライブラリ関数では、出力ポート番号、内容、サイズをキューに保存し、あるタイミングでホスト OS

Vol.2009-ARC-183 No.13
Vol.2009-OS-111 No.13
2009/4/23

の仮想計算機モニタに処理を移す。ホスト OS の仮想計算機モニタでは、キューに溜められている情報からどのような命令が実行されたかを再現し、内部に含まれているエミュレータを呼び出す。

提案方式の特徴は、既存のデバイス・ドライバを再利用しながら入出力を高速化していることにある。このため、従来の準仮想化に基づく方法とは異なり、手書きで準仮想ドライバを開発する必要はない。本方式では、既存のデバイス・ドライバのソース・プログラムを、特別に用意したコンパイラでコンパイルし、ゲスト OS のカーネルに動的にロードするだけで入出力の高速化が計られる。

提案方式を、Linux で動作する標準的な仮想計算機モニタである KVM において実装した[10]。機械語命令変換を行うプログラムを、既存のアセンブラ・プリプロセッサを改変して実現した。提案方式を、KVM が扱うことができるエミュレータのうち、AMD PCNET32 のデバイス・ドライバに適用した。その結果、既存のエミュレーションを行う方法と比較して、ネットワークのスループットが Linux で最大 5.5 倍、FreeBSD で 4.9 倍に高速化された。

この論文の構成は、以下のようになっている。2 章では、関連研究について述べる。3 章では、提案方式について述べる。4 章では、Linux KVM における実装について述べる。5 章では、実験結果を示す。6 章では、今後の課題について述べる。最後に 7 章では、この論文のまとめを行う。

2. 関 連 研 究

仮想計算機の入出力性能が低い問題を解決するために、準仮想ドライバをゲスト OS に組込むことは広く行われている。Xen、および、Linux KVM Virtio では、ブロック・デバイスとネットワーク・デバイスについて準仮想化により入出力性能を改善している[4][13]。この方法では、仮想計算機内で動作する OS(ゲスト OS) のデバイス・ドライバを手書きの準仮想ドライバに置き換える。準仮想ドライバは、仮想計算機固有の機能を利用して、多くの場合ホスト OS で動作しているバックエンドのモジュールに入出力の要求を伝える。バックエンドのモジュールは、ホスト OS の機能の低レベルの入出力機能を利用して入出力を行う。たとえば、(ファイルではなく) ブロック入出力や (Socket API ではなく) ネットワーク・フレーム入出力の機能を用いる。

準仮想ドライバにより、エミュレーションが不要になるので入出力性能は大きく改善される。しかしながら、準仮想ドライバを用いる方法には、開発コストが大きいという問題が残っている。準仮想ドライバは、ゲスト OS と仮想計算機モニタの両方に依存している。ある準仮想ドライバが存在した時、同じタイプのデバイスであったとしても、別のゲスト OS や別の仮想計算機モニタでは動作しない。たとえば、仮想計算機モニタ Xen で動作する Linux のネットワーク・デバイスのための準仮想ドライバは、別の仮想計算機モニタ Linux KVM では動作しない。また、同じ Xen であったとしても、FreeBSD では動作しない。

アウトソーシング (*outsourcing*) は、準仮想ドライバとは異なり、ゲスト OS の高水準モジュールを置き換えることでエミュレーションを不要にしている[8]。たとえば、ゲスト OS のソケット層の処理を、ホスト OS のバックエンドのモジュールに投げることで、ゲスト OS 内での TCP/IP 処理やネットワーク・デバイス・ドライバの処理を不要にしている。ゲスト OS のモジュールとホスト OS のバックエンド・モジュールの間は、遠隔手続き呼び出し (Remote Procedure Call) で結ばれている[8][14]。

本研究では、準仮想化やアウトソーシングとは異なり、デバイスのエミュレーションを行いながら入出力を高速化する。また、既存のデバイス・ドライバを再利用することで、準仮想ドライバの開発や高水準モジュールの置換えを行うことなく入出力を高速化する。

LilyVM は、静的な機械語命令変換により CPU を仮想化している[7][6]。LilyVM では、アセンブラのプリプロセッサを用いて特権命令やセンシティブな非特権命令を以下のいずれかの方法で書き換えている。

(1) 無効命令を埋め込む。これによりセンシティブな非特権命令の実行を検出する。
(2) ゲスト OS の空間に用意したライブラリ関数を呼び出す。これによりトラップのオーバヘッドを下げる。

Afterburner (Pre-virtuliation) では、LilyVM の技術を発展させ、アセンブル時には特定の命令の場所を記録し、後に行う書き換えに必要な空間を nop (no operation) 命令を埋め込むことで確保する[11]。そして OS をロードする時に、環境 (Xen, L4, 実機) に合わせて、nop 命令を含めて書き換える。また、機械語命令を変換することにより、ネットワークで接続された計算機群を共有メモリ型マルチプロセッサとして利用できるようにする研究もある[9]。本研究においても、これらの研究と同じく静的な機械語命令変換を行う。これらの研究と比較して、本研究の特徴は、完全仮想化の仮想計算機において入出力の高速化を目的として機械語命令変換を行うこと、および、システム全体ではなくモジュール (デバイス・ドライバ) を単位として機械語命令変換を行うことにある。

VMware VMI では、入出力命令やその他の CPU 仮想化に必要な命令を VMI-ROM と呼ばれる一種のライブラリで提供する[3]。各ゲスト OS の開発者は、VMware VMI を利用

Vol.2009-ARC-183 No.13
Vol.2009-OS-111 No.13
2009/4/23

するように OS を修正することで仮想化のオーバヘッドを削減する。VMI-ROM は、実機で実行された場合には、該当する命令をそのまま実行し、VMware Workstation 等で実行された場合には、仮想計算機モニタの機能を呼び出す。VMI-ROM で入出力命令を解釈しているのは、現在の所、local APIC (Advanced Programmable Interrupt Controller) のみであり、大部分の入出力命令はデバイスのエミュレータに渡される。Linux においては、VMware VMI への対応は、CPU 仮想化については、paravirt_opt への対応ということでメインラインでなされていたが、入出力命令の部分は未対応である。本研究では、デバイス・ドライバに含まれる入出力命令を言語処理系で自動的に変換する所が VMware VMI の手法と異なる。

VMware Workstation では、ネットワークの高速化のために様々な手法が用いられている[15]。その中の 1 つに、出力命令をカーネル・レベルのハイパバイザ内で溜め、後にユーザ・レベルのエミュレータで一括処理をすることも含まれている。一括処理するタイミングとしては、特定のデバイス (AMD Lance) の性質を活用している。この本研と比較して、本研究は、デバイスの種類から独立している点、および、ゲスト OS と同じ空間で動作するライブラリ内で出力命令を溜めている点が異なる。

3. 仮想計算機における入出力命令の一括処理

3.1 提案方式の概要

提案方式の概要を、既存方式と対比して **図 1** に示す。この図では、ホスト型仮想計算機[⋆1]においてゲスト OS が動作している。仮想計算機モニタは、カーネル・レベルで動作するハイパバイザ (Hypervisor) とユーザレベルで動作する部分 (User Level VMM) に分かれている [⋆2]。ユーザレベルの仮想計算機モニタには、デバイスのエミュレータが含まれている。Linux KVM や Xen HVM (Hardware Virtual Machine) では、デバイスのエミュレータとしては、Qemu[5] に含まれているものが使われている。

図 1 (a) は、従来の入出力命令の処理方式を示している。ゲスト OS のデバイス・ドライバが、入出力命令を実行すると、それは、ハイパバイザより捉えられ、ユーザ・レベルのエミュレータに通知される。この通知は、基本的には入出力命令の 1 命令の実行ごとに行われる。デバイス・エミュレータは、ある特定のハードウェアのエミュレーションを行う。たとえば、AMD PCNET32 のエミュレーションを行う。エミュレータは、入出力がが必要になった場合、システム・コールを発行し、ホスト OS の機能を利用する。ホスト OS の内部では、デバイス・ドライバが動作し、入出力を行う。

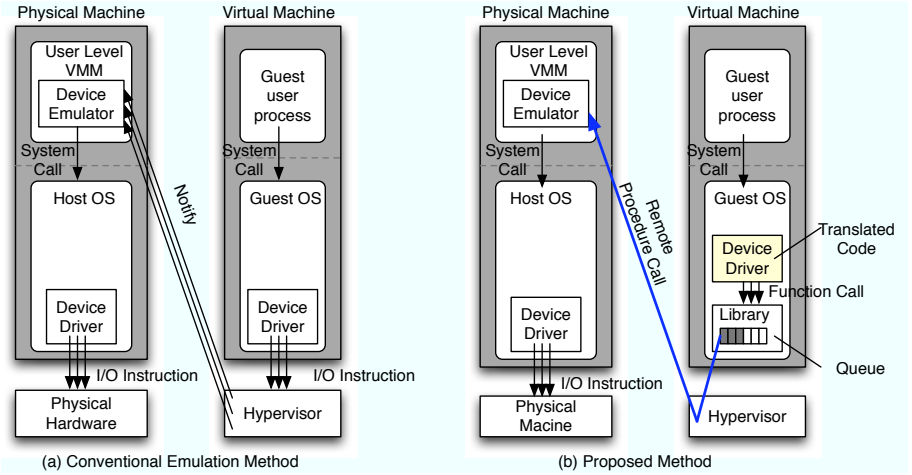

図 1 デバイス・エミュレータを含む仮想計算機モニタにおける入出力の処理
Fig. 1 I/O processing in a virtual machine monitor including a device emulator.

図 1(b) は、提案方式における入出力命令の処理方式を示している。ゲスト OS には変換されたデバイス・ドライバとライブラリが組込まれている。変換されたデバイス・ドライバは、入出力命令を実行する代わりにライブラリ関数を呼び出す。ライブラリ内には、キューがあり、出力命令を溜める。入力命令が発行された場合など、必要な時には、ホストへ遠隔手続き呼び出しを行い、キューをフラッシュする。あるいは、ゲスト OS がアイドル・ループに入った時など、暗黙的にキューがフラッシュされることもある。ホスト側では、キューの内容を解析し、エミュレータを呼び出す。このように、キューで複数の出力命令を溜め、後に一括で実行することで、ホスト OS とゲスト OS の切り替え頻度を大きく下げることができる。

3.2 静的な機械語命令の変換による入出力命令の補足

本研究では、2 章で述べた LilyVM と同様に静的に機械語命令の変換を行う。その様子を **図 2** に示す。このように、既存のデバイス・ドライバのソースコードを入力として、高速なデバイス・ドライバが生成されカーネルにロードされている。C 言語のスタブを用いる

⋆1 Xen のように、入出力のための特別な仮想計算機を用いる手法も含む。
⋆2 Linux KVM では、ハイパーバイザは、ホスト OS のカーネルのコンテキストで動作する。

Vol.2009-ARC-183 No.13
Vol.2009-OS-111 No.13
2009/4/23

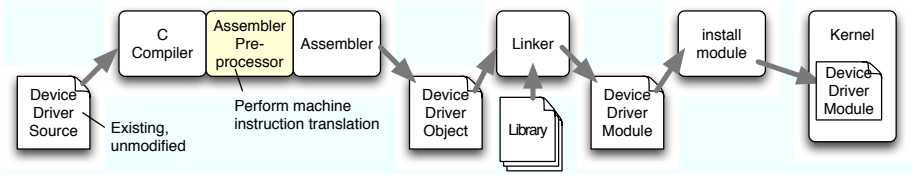

図 2 アセンブラ・プリプロセッサによる静的な機械語命令変換によるドライバの生成
Fig. 2 Generating a device driver along with static machine instruction translation by an assember preprocessor.

方法[18] と比較して、アセンブリ言語のレベルで変換する方法の利点としては、デバイス・ドライバでしばしば使われるインライン・アセンブリ言語記述 (asm() 文) に対応できることがあげられる。この手順は、OS から独立しており、複数の OS で実行可能である。本研究では、実際に Linux、および、FreeBSD という 2 つの代表的なオープンソースの OS に対して提案方式が適用できることを確認した。

LilyVM と異なる点は、第 1 に、CPU の仮想化ではなく、入出力に焦点を当てている点である。LilyVM では、入出力には、主に手書きの準仮想ドライバを用いていたが、本研究ではデバイスのエミュレータを用いる。第 2 の相違点は、OS 全体ではなく、デバイス・ドライバというモジュール単位で適用している点にある。これにより、LilyVM 以外の完全仮想化に基づく仮想計算機においても、提案手法を用いることが可能になっている。

3.3 溜めた命令を処理するタイミング

溜めた出力命令を処理するタイミングとしては、次のようなものが考えられる。

(1) 入力命令が発行された時。
(2) 仮想計算機の CPU がアイドルになった時。
(3) 溜めた出力命令の数がある閾値に達した時。
(4) ハードウェアからホスト OS へ割り込みが発生し、仮想計算機の処理を中断する必要が生じた時。
(5) 指定された最大の待機時間が経過した時。
(6) デバイス固有のタイミング。たとえば、イーサネット・デバイスで 1 フレームを送信ための最後の出力命令を実行した後に、それまでの出力命令を一括で処理する。
(7) 元のデバイス・ドライバで、一連の処理（たとえば、1 フレームの送信）の完了を識別し、それまでの出力命令を一括で処理する。

出力命令を溜めることで、大域的に見て入出力命令の実行タイミングが元のもの異なることになる。このことにより、意味的な問題と性能的な問題が発生する可能性がある。意味的な問題とは、デバイスのエミュレータが受け取る命令列が元の物と異なり、正しくデバイスのエミュレーションができなくなることである。我々は、ゲスト OS のデバイス・ドライバが正しく記述されていれば、個々のデバイスのエミュレータのレベルにおいては、元の入出力命令列と一致し、時間切れを除き意味的な問題は生じないと考えている。

性能的な問題としては、溜めた出陸命令を処理するタイミングとして上記のどの項目を選択するか、また、溜める個数の上限や最大の待機時間を定める必要がある。これらを決めることは、それほど容易なことではない。デバイス、アプリケーション、および、ワークロードによって異なる。スループットを重視する場合とレイテンシを重視する場合で要求が異なることがある。

本研究では、いくつか実験を行った所、上記の項目のうち、(1) から (3) を採用している。(4) については、複数の命令を溜める効果が薄れてしまうという側面もあるため、ネットワークのスループットが低下することがあったので現在は標準では用いていない。(6) については、デバイス固有の知識が必要になる。たとえば、VMware Workstation 2.0 では、AMD Lance という特定のネットワーク・デバイスに特化して、高負荷の時には 3 個の送信パケットが溜まるまで出力命令を溜める[15]。これに対して本研究では、デバイスの固有の性質を使わないでどこまで高速化できるかを探りたいと考えている。(7) は、(6) の近似であり、デバイス固有の知識を使わないで実現できる。それを実現するには、機械語命令変換の時に、大域的な関数のリターン命令を書き換えたり、デバイス・ドライバの関数の表を置き換える方法が考えられる。

今後、着目する項目や溜める個数や最大の待機時間等のパラメタの設定を自律的に行う仕組みや、動的に変化するワークロードに追随する仕組みを実現していきたいと考えている。

4. Linux KVM における実装

4.1 機械語命令変換の実装

機械語命令変換のために、Afterburner[11] において実装されたアセンブラ・プリプロセッサを改変して使用した。このプリプロセッサは、Antlr という言語処理系を用いて記述されている。対象とした x86 アーキテクチャのアセンブラの文法は複雑であり、文字列の置き換えでは限界がある。Afterburner のアセンブラ・プリプロセッサは、着目している特権命令やセンシティブな非特権命令を見つけると、ロード時の書き換えに備えて、nop 命令で空間を作り、かつ、その位置を特別のセグメントに記録する。本研究では、まず、それらの不

Vol.2009-ARC-183 No.13
Vol.2009-OS-111 No.13
2009/4/23

要な機能を削除した。次に、以下のような入出力命令が現れると、それを手続き呼び出し (call 命令) を出力するようにした。

```
in, inb, inw, inl, out, outb, outw, outl
```

以下に、outw 命令の書き換えの例を示す。

書き換え前

```
outw %ax, %dx
```

書き換え後

```
pushl %ax
pushl %dx
call  outw_add_queue
addl  $8, %esp
```

このように、元の outw 命令のオペランドをスタックに引数を積み、call 命令でライブラリ関数を呼び出している。

4.2 ライブラリ関数の実装

アセンブラ・プリプロセッサにより、in 命令や out 命令は、C 言語で記述されたライブラリ関数の呼び出しへ書き換えられる。ライブラリ関数は、ホストで動作しているユーザ・レベル仮想計算機モニタとの間に共有のデータ構造を持っている。このデータ構造には、出力命令を溜めるキューと入力命令のオペランドや結果を保持するための変数がある。

out 命令の処理の概略を以下に示す。

(1) 共有データ構造の初期化が済んでいなければ初期化を行う。
(2) 出力命令を溜めるキューに、ポート番号、値、語長を保存する。
(3) 出力命令を溜めるキューの長さが閾値を超えていた場合、遠隔手続き呼び出しによりユーザ・レベル仮想計算機モニタを呼び出し、キューをフラッシュする。

in 命令の処理の概略を以下に示す。

(1) 共有データ構造の初期化が済んでいなければ初期化を行う。
(2) ポート番号、語長を共有メモリ上のデータ構造に保存する。
(3) 遠隔手続き呼び出しによりユーザ・レベル仮想計算機モニタに制御を移し、出力命令を溜めるキューのフラッシュと入力処理を行う。
(4) ユーザ・レベル仮想計算機モニタから返された値を、in 命令の結果とする。

4.3 Linux KVM における実装

本研究では、Linux KVM を改変して、ゲスト側のライブラリ関数をクライアント、ホスト側のユーザ・レベル仮想計算機モニタにある、本研究で拡張した部分をサーバとする遠隔手続き呼び出しを可能にした。この遠隔手続き呼び出しの実装は、アウトソーシングの実装[8] で用いているものとほとんど同じである。具体的には、ゲスト側では、Intel VT の vmmcall 命令の実行を遠隔手続き呼び出しの契機として用いている。KVM のカーネル内のプログラムは、vmmcall 命令の実行を補足すると、ユーザ空間にある仮想計算機モニタを呼び出す。

4.4 ユーザ・レベル仮想計算機モニタの拡張

KVM は、デバイスのエミュレータとして、Qemu に由来するものを用いている[5]。Qemu/KVM では、デバイスのエミュレータは、入出力命令や MMIO (Memory Map I/O) におけるデータの読み書きを入力とするオートマトンとして実現されている。今回の実装では、デバイスのエミュレータ部分は、一切改変することなくそのまま利用した。

ユーザ・レベル仮想計算機モニタは、カーネル・レベルの KVM のコードから様々な通知を受け取る。本研究では、その通知の解析部分を変更し、ゲスト OS からの遠隔手続き呼び出しを認識するようにした。ゲスト OS からの遠隔手続き呼び出しの要求を受け取ると、本研究で実現したサーバを呼び出す。

遠隔手続き呼び出しのサーバは、次の 3 つの手続きを提供する。

init() ゲスト OS との共有のデータ構造の初期化を行う。ゲスト側で割り当てたメモリのホスト空間内での番地 (host virtual address) を計算する。

outflush() 共有データ構造にある出力命令を溜めたキューからポート番号、値、語長を取出して、エミュレータを呼び出す。

outflush_and_in() outflush() の処理を行った後に、共有データ構造からポート番号、語長を取出して、エミュレータを呼び出す。結果の値を、共有データ構造に書き込む。

5. 実　　験

5.1 環境と設定

実験に用いたコンピュータは、CPU として Intel Core2 Duo 2.40GHz、メモリが 2GB、物理ネットワーク・インタフェース・カードとして Intel 82566DC (e1000) を備えたものである。これを、スイッチ Dell Gigabit Ethernet Switch PowerConnect 2616 に接続した。利用したホスト OS は、kernel:2.6.22-14、仮想計算機モニタとしては、KVM-48 を用いた。ゲスト OS としては、Linux 2.6.22-14、および、FreeBSD 7.0 RELEASE を用いた。実験では、ゲスト OS には、512MB のメモリを与えた。

Vol.2009-ARC-183 No.13
Vol.2009-OS-111 No.13
2009/4/23

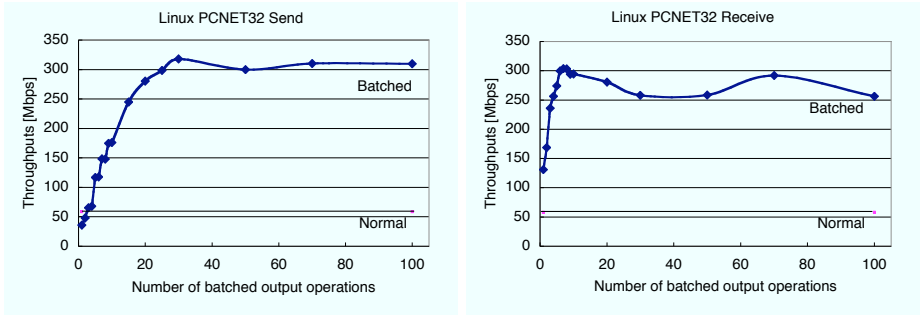

図 3 Linux におけるスループット (PCNET32)
Fig. 3 Throughputs in Linux (PCNET32)

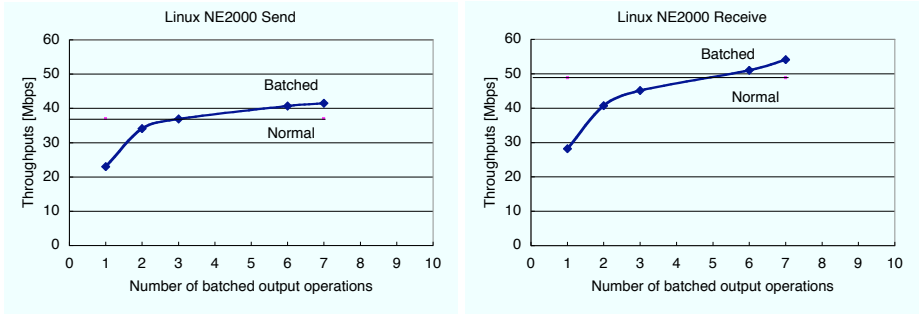

図 4 Linux におけるスループット (NE2000)
Fig. 4 Throughputs in Linux (NE2000)

実験として、次の 2 つのデバイスのドライバに提案手法を適用した。

- AMD PCNET32 (Linux では pcnet32.c、FreeBSD では if_le.c)
- NE2000 (Linux では ne2k-pci.c、FreeBSD では if_dei.c と if_de_pci.c)

これらのデバイスを選んだ理由は、KVM (Qemu) がエミュレータを内蔵していること、および、多くのゲスト OS でこれらのデバイスに対するドライバを持っていることによる。

実験では、ネットワークのスループットを iperf を用いて測定した。iperf に与えたパラメタは、全てデフォルト (メッセージサイズが 8KB) である。実験では、通信相手としては、ネイティブ動作している Linux (2.6.22-14) を用いた。スループットを促成する実験では、iperf を 10 回実行し、その結果の平均を求めた。また、ネットワークの往復時間 (Round Trip Time) を、ping コマンドで測定した。測定は 10 回行った。最初の 1 回は、初期化等の影響が見られたので、それを除外するため、最初の 1 回を除き、平均を求めた。

5.2 Linux ゲストでのスループット

ゲスト OS として Linux を動作させ、本提案方式に基づいて変換したデバイス・ドライバを用いて、ネットワークのスループットを測定した。エミュレートされたデバイスとして PCNET32 を用いた時の実験結果を**図 3** に示す。PCNET32 では、通常のドライバを用いた場合、スループットが送信で、58M bps、受信で 127 Mbps であった。このように、PCNET32 では、送信では 30 個命令を溜めるまで性能が向上し、最大で 318M bps のスループットを得た。これは、通常のドライバの 5.5 倍の性能である。受信では、6 個の命令を溜めた時に最大の 303 Mbps のスループットを得た。これは、通常のドライバの 2.4 倍の性能である。このように、PCNET32 については提案方式が有効であった。

別のネットワーク・デバイスである NE2000 においても、同様の実験を行った。結果を**図 4** に示す。この場合、通常のドライバでは、送信のスループットが 37.0M bps、受信のスループットが 48.8M bps でった。本研究を適用したドライバでは、スループットが送信で最大 41.5 Mbps、受信で最大 54.1 Mbps になった。すなわち、それぞれ、提案方式により 12% 、および、10% 高速化された。これ以上の多く溜めた時、および、特定の回数 (5 回と 6 回) では、ドライバがうまく動作しなかった。現在、その原因を追求している。

図 3 の PCNET32 の結果と 図 4 の NE2000 の結果を比較すると、PCNET32 の方が提案方式がより有効に働いていると言える。現在、デバイス、または、ドライバの性質と提案方式の効果の現れ方の関係を解析している。

5.3 Linux ゲストでのネットワークの往復時間

出力命令を溜め一括処理を行うと、ネットワークのスループットは改善されるが、レイテンシは悪化することが予想される。その効果を調べるために、PCNET32 について、ネットワークの往復時間を調べる実験を行った。その結果を、**図 5** に示す。この実験では、往復時間は、出力命令を溜める個数を変えても有意な変化は見られなかった。またこの時間は、通常のデバイス・ドライバとの差も見られなかった。

往復時間が出力命令を溜める個数を変えても変化しない理由としては、この実験では 3.3 節で述べた、仮想計算機の CPU がアイドルになった時に出力命令を処理することが有効に機能していることがあげられる。この実験では、仮想計算機内では、ping コマンドのみを実行しており、他に CPU を消費しているプロセスは存在しない。そのため、ゲスト OS は、すぐに hlt (halt) 命令を実行して仮想計算機をアイドル状態にする。この時に、ping コマ

Vol.2009-ARC-183 No.13
Vol.2009-OS-111 No.13
2009/4/23

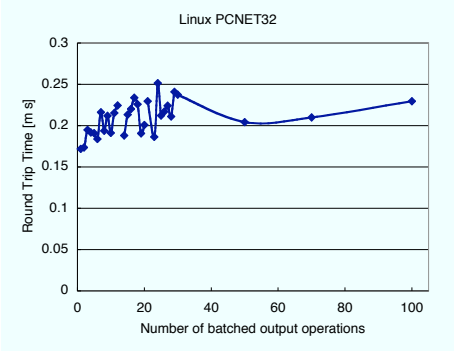

図 5 Linux における往復時間 (PCNET32)
Fig. 5 Round Trip Times in Linux(PCNET32)

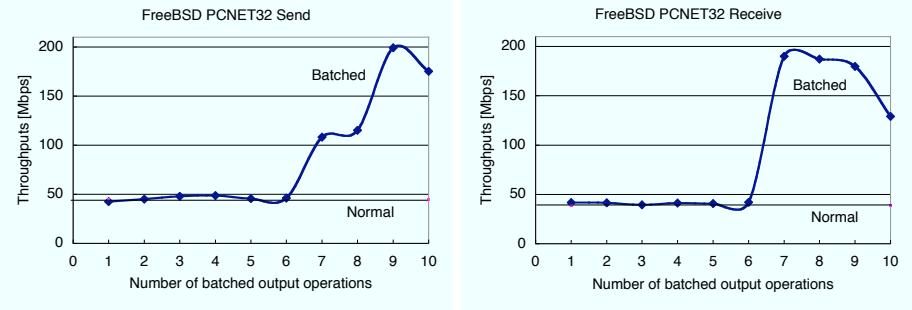

図 6 FreeBSD におけるスループット (PCNET32)
Fig. 6 Throughputs in FreeBSD (PCNET32)

ンドがネットワークに対してメッセージを送信するために溜めた命令がすぐに処理される。

5.4 FreeBSD ゲストでのスループット

FreeBSD においても、Linux と同様に、PCNET32、および、NE2000 について本提案方式を適用したデバイス・ドライバを用いてネットワークのスループットを測定した。

PCNET32 における結果を、**図 6** に示す。このように、PCNET32 においては、10 個まで出力命令を溜めた場合に、ネットワークのスループットが通常のドライバと比較して送信が最大で 3.5 倍、受信が 4.9 倍に高速化された。11 個以上溜めた場合には、動作しなかった。NE2000 においては、3 個まで出力命令を溜めることはできたが、この範囲においてはネットワークのスループットは改善されなかった。

図 3 に示した Linux の結果と図 6 に示した FreeBSD の結果を比較すると、異なる OS においても、PCNET32 では出力命令の一括処理の効果が大きいことがわかる。

6. 今後の課題

現在の実装では、出力命令を溜める個数によっては、プログラムが動作しないことがある。今後まずこの原因を追及し、完成度を高めて行きたいと考えている。また出力命令を溜める個数等の様々なパラメタについても、デバイス、OS、および、ワークロードに応じて自律的な設定を可能にしていきたい。

現在、入力命令については一括処理の対象とはしていない。デバイスの種類や入力の性質によっては、何らかのヒントを与えることで一括処理が可能であることが存在すると思われる。今回用いたデバイス・ドライバではいずれも DMA によりデータを転送していたが、PIO (Programmed I/O) による処理の場合には、入力命令を一括処理する効果が絶大であることが予想される。ただし、ネットワークやハードディスク等の高速化が求められているデバイスでは、DMA を使うことが一般的であり、PIO のための最適化の恩恵を得られる事例は少ないと思われる。VMware Workstation 2.0 では、AMD Lance のメモリ的なセマンティクスを用いて、デバイスのステータスを得るための入力命令を高速化している。この例に習い、特定のポート番号については、出力命令の実行や入出力の完了等のイベントがない間は、前回と同じ結果を返すことで入力命令の高速化が計られるものと思われる。

デバイスによっては、入出力命令ではなく、MMIO (Memory Mapped I/O) により入出力を行うものがある。そのようなデバイスについては、そのままでは入出力命令を書き換える本手法は適用できない。しかしながら、多くの OS では、MMIO によりデバイスをアクセスする場合においても、特定の関数を呼び出すようにしている。たとえば、Linux では、readb() や writeb() という関数 (マクロ) が使われている。このような性質を利用することで、MMIO を利用するデバイスについても一括処理を行いたいと考えている。

現在、デバイスのエミュレータは、ホスト OS で動作しているユーザ・レベルの仮想計算機モニタの 1 つのモジュールとして動作している。このエミュレータを、ゲスト OS 側へ移動させることで、かなりの高速化が可能になると思われる。さらに、インライン・アセンブラの問題を解決すれば、特化を適用することも可能であると思われる[18)]。

Vol.2009-ARC-183 No.13
Vol.2009-OS-111 No.13
2009/4/23

7. おわりに

この論文では、完全仮想化に基づく仮想計算機における入出力を高速化する方法を提案した。提案方式では、ゲストＯＳのデバイス・ドライバに含まれる入出力を行う機械語命令を静的に書き換え、複数の出力命令を一括に処理する。これにより、仮想計算機とホストＯＳの切替え回数が削減され、入出力が高速化される。

提案方式は、仮想計算機モニタ Linux KVM で実現されている。提案方式をネットワーク・デバイス AMD PCNET32 のドライバに対して適用した所、最大でネットワークのスループットが Linux で 5.5 倍、FreeBSD で 4.9 倍に高速化されている。

今後の課題は、入力命令についても一括処理やゲスト OS 内で完結させる処理を実現することである。また、入出力命令を溜める個数等の様々なパラメタについても、デバイス、OS、および、ワークロードに応じて自律的な設定を可能にしていきたいと考えている。

参考文献

1) Adams, K. and Agesen, O.: A Comparison of Software and Hardware Techniques for x86 Virtualization, *Proceedings of the 12th international conference on Architectural support for programming languages and operating systems (ASPLOS-XII)*, pp.2–13 (2006).
2) Advanced Micro Devices (AMD), Inc.: AMD Secure Virtual Machine Architecture Reference Manual (2005).
3) Amsden, Z., Arai, D., Hecht, D., Holler, A. and Subrahmanyam, P.: VMI: An interface for paravirtualization, *Ottawa Linux Symposium* (2006).
4) Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I. and Warfield, A.: Xen and the art of virtualization, *Proceedings of the ACM Symposium on Operating Systems Principles, ACM SIGOPS Operating System Review*, Vol.37, No.5, pp.164–177 (2003).
5) Bellard, F.: QEMU, a fast and portable dynamic translator, *Proceedings of the USENIX Annual Technical Conference, FREENIX Track*, pp.41–46 (2005).
6) 榮樂英樹，　新城靖，加藤和彦：カーネル・レベル・コードによるユーザ・レベル VMM の移植性の向上，情報処理学会研究報告システムソフトウェアとオペレーティング・システム 2007-OS-104，　Vol.2007, No.10, pp.17–24 (2007).
7) Eiraku, H. and Shinjo, Y.: Running BSD kernels as user processes by partial emulation and rewriting of machine instructions, *BSDC'03: Proceedings of the BSD Conference 2003 on BSD Conference*, USENIX Association, pp.91–102 (2003).
8) Eiraku, H., Shinjo, Y., Pu, C., Koh, Y. and Kato, K.: Fast Networking with Socket-Outsourcing in Hosted Virtual Machine Environments, *Proceedings of the 24th Annual ACM Symposium on Applied Computing*, pp.310–317 (2009).
9) 金田憲二，大山恵弘，米澤明憲：単一システムイメージを提供するための仮想マシンモニタ，情報処理学会論文誌. コンピューティングシステム (ACS13)，　Vol.47, No.SIG3, pp.27–39 (2006).
10) Kivity, A., Kamay, Y., Laor, D., Lublin, U. and Liguori, A.: kvm: the Linux virtual machine monitor, *Proceedings of the Linux Symposium*, pp.225–230 (2007).
11) LeVasseur, J., Uhlig, V., Chapman, M., Chubb, P., Leslie, B. and Heiser, G.: Pre-Virtualization: Soft Layering for Virtual Machines, Technical Report 2006-15, Fakultät für Informatik, Universität Karlsruhe (TH) (2006).
12) Robin, J. and Irvine, C.: Analysis of the Intel Pentium's ability to support a secure virtual machine monitor, *Proceedings of the 9th USENIX Security Symposium, Denver, CO*, pp.14–17 (2000).
13) Russell, R.: virtio: towards a de-facto standard for virtual I/O devices, *SIGOPS Oper. Syst. Rev.*, Vol.42, No.5, pp.95–103 (2008).
14) 齊藤　剛，新城　靖，榮樂英樹，佐藤　聡，中井央肯三：仮想計算機におけるアウトソーシングのためのゲスト-ホスト間 RPC，情報処理学会第 20 回コンピュータシステム・シンポジウム (ComSys2008) ポスター＆デモセッション (2008).
15) Sugerman, J., Venkitachalam, G. and Lim, B.-H.: Virtualizing I/O Devices on VMware Workstation's Hosted Virtual Machine Monitor, *Proceedings of the General Track: 2002 USENIX Annual Technical Conference*, pp.1–14 (2001).
16) 高橋由直，新城　靖，佐藤　聡，中井　央，板野肯三：モード遷移削減による仮想計算機の高速化，第 70 回情報処理学会全国大会，pp.151–152 (2008).
17) Uhlig, R., Neiger, G., Rodgers, D., Santoni, A.L., Martins, F. C.M., Anderson, A.V., Bennett, S.M., Kagi, A., Leung, F.H. and Smith, L.: Intel Virtualization Technology, *IEEE Computer*, Vol.38, No.5, pp.48–56 (2005).
18) 山本悠輔，新城　靖，榮樂英樹，板野肯三，佐藤　聡，中井　央，加藤和彦：仮想計算機におけるデバイスエミュレーションの特化による高速化，情報処理学会研究報告システムソフトウェアとオペレーティング・システム 2008-OS-107，　Vol.2008, No.9, pp.103–110 (2008).