Vol.2011-ARC-195 No.11
Vol.2011-OS-117 No.11
2011/4/13

Solid State Drive 間の高速データ移動法

鈴木順[†] 高橋雅彦[†] 飛鷹洋一[††] 馬場輝幸[†]
大和純一[†] 樋口淳一[†] 菅原智義[†] 吉川隆士[†]

Solid State Drive (SSD)を始めとする広帯域 I/O デバイスの発展により，これらのデバイスを組み合わせて広帯域処理が行える可能性が高まっている．このために必要な技術の 1 つがデバイス間の高速データ移動である．現在のコンピュータシステムでは，I/O デバイス間で高速にデータを移動する場合，CPU によるコンテキストスイッチやカーネルとユーザメモリのコピーがオーバヘッドとなる．本稿では，I/O デバイス間のデータパスを最適化するため，ユーザ層より低位の層でデータを移動する手法を提案する．本手法により，ユーザプログラムはコンテキストスイッチや仲介バッファでのデータコピーのない広帯域なデータ移動が行える．提案手法の評価のため，ある SSD からデータをリードし，他の SSD にそのデータをライトするデータコピーのプロトタイプを作成した．評価の結果，従来のユーザ層でのコピーと比較して，177%の帯域向上と 62%の CPU 使用率削減が得られた．なお，これらの値は用いた SSD の性能に制限された値である．

High-Throughput Data Transfer between Solid State Drives

Jun Suzuki[†] Masahiko Takahashi[†] Yoichi Hidaka[††]
Teruyuki Baba[†] Junichi Yamato[†] Junichi Higuchi[†]
Tomoyoshi Sugawara[†] and Takashi Yoshikawa[†]

Recently developed high-throughput I/O devices such as solid state drives (SSDs) can be combined to cooperate on the high- throughput processing of a single task . While high-throughput data transfer between I/O devices is indispensable for such data processing, such data transfer on current computer systems involves high overhead caused by context switching and data copying between kernel and user memories, as conducted by CPUs. In this paper, we aim to optimize data paths between I/O devices and propose a data transfer method which is performed in a layer lower than the user layer. It provides a function for programs in user spaces to trigger the data transfers, achieving high-throughput data transfer without the need to copy data in side an intermediate buffer. In data copying between two SSDs, that is, "read" data form one SSD and "write" it to the other, we have obtained 177% enhanced bandwidth and 62% lower CPU usage over that with a conventional user-level "copy" process. The numbers are limited by the SSDs used in the experiments.

1. はじめに

クラウド時代には，ネットワーク接続された環境下で医療やマルチメディアなどの大規模データに対しスケーラブルな広帯域データ処理を行う技術が重要となる．このため，今日では並列性を生かして CPU によるデータ処理の広帯域化を実現する技術が導入されている．例えばコンピュータ内の技術としてはマルチスレッドが用いられ，コンピュータ間をまたがる技術としてクラスタリングが用いられている．その一方，コンピュータを構成する他の要素の 1 つとして I/O デバイスがあるが，I/O デバイスはこれまでのコンピュータにおいて CPU 上のソフトウェアから制御され，命令に従って単純な I/O 処理を行うだけだった．

しかし，近年，I/O デバイスを取り巻く状況が変化している．フラッシュデバイスや 40Gb/s のネットワークインタフェースカード(NIC)の登場により，I/O デバイスの帯域は飛躍的に増大した．さらに，デバイスによっては組み込み CPU を備え，豊富な種類の I/O サブシステムを提供することも可能である[1, 2].

我々はこれらの I/O デバイスを組み合わせ，複数の I/O デバイスを連携させながら広帯域なデータ処理を行うアーキテクチャの実現を目指している．そのようなアーキテクチャの下では，データはある I/O デバイスで処理され，続いて別のデバイスで処理されるため I/O デバイス間でデータを広帯域に移動させる技術が重要となる．

今日のコンピュータシステムでは，I/O デバイス間のデータ移動はユーザプログラムにより行う必要がある．つまり，ソース I/O デバイスからユーザ空間にデータを読み込み，読み込んだデータをターゲット I/O デバイスに書き込む．しかし，これらの処理はオーバヘッドが大きい．これはカーネルモードとユーザモードのコンテキストスイッチやカーネルメモリとユーザメモリ間のデータコピー，また，I/O デバイス間を 1 度しか移動しないデータをファイルキャッシュとしてキャッシュする処理などを実行するためである．これらの動作は，データ移動性能の劣化とハードウェアリソース消費の増大の要因となる．

これまで，ソフトウェアで I/O データを効率良く扱う手法は提案されている．例えば J. Pasquale らは Container Shipping を提案している[3]. ここでは I/O データに関係するプロセスが I/O パイプラインを形成し，I/O データを保持するバッファのアクセス権をパイプラインに沿って引き継ぐことにより，I/O データのコピーを回避する．一方 V. S. Pai らが提案している IO-Lite では，I/O データを保持するバッファのアクセス権が移動するのではなく，バッファをアプリケーション，I/O サブシステム，ファイル

[†] NEC システムプラットフォーム研究所
System Platforms Research Laboratories, NEC Corporation
[††]NEC IP ネットワーク事業部
IP Network Division, NEC Corporation

Vol.2011-ARC-195 No.11
Vol.2011-OS-117 No.11
2011/4/13

キャッシュの間で読み込み専用により共有する[4]．しかし，これらの手法のほとんどがユーザ空間と I/O デバイスの間のデータ移動に着目しており，I/O デバイス間のデータ移動には着目してこなかった．そのため，I/O デバイス間のデータ移動にはやはりユーザプログラムでの実現が必要であり，コンテキストスイッチによる CPU リソース消費も発生する．

また Weinsberg らは，CPU の仲介なく I/O デバイスの間で直接データを移動する手法を提案している[5]．この手法では，組み込み CPU を持つ I/O デバイスが用いられ，I/O デバイス間で直接データ移動を行うように各組み込み CPU をプログラムする．本手法のメリットは，ユーザプログラムで I/O デバイスを操作する必要がないことと，データ移動においてメモリバス帯域を使用しないことである．しかし，本手法は必ずしも I/O デバイス間の広帯域データ移動に寄与するものではなく，また本手法の適用も組み込み CPU を持つプログラマブルな I/O デバイスに限定される．

本稿では，I/O デバイス間のデータパスを最適化することを狙い，ユーザ層より低い層でデータを移動する手法を提案する．提案手法の狙いは，アプリケーションから扱いやすいプログラムモデルを提供すること，広帯域でハードウェアリソース消費が少ないこと，様々な I/O デバイスに適用可能なことである．提案手法では，扱いやすいプログラムモデルという点を，ユーザプログラムから I/O デバイス間のデータ移動を行う関数を呼ぶだけでデータ移動が行える，ということにより実現した．また広帯域でハードウェアリソース消費が少ないデータ移動は，カーネルモードでデータ移動を行い，コンテキストスイッチや仲介バッファでのメモリコピーを回避するということにより実現した．また様々な I/O デバイスに適用可能とするために，OS コードへの変更を最小限に抑え，データ移動機能をデバイスドライバとして実装することで，様々な I/O デバイスに対処した実装が可能となるようにした．

提案手法を用いて SSD 間のデータコピーのプロトタイプを作成し評価を行ったところ，従来のバッファード I/O を用いたユーザ層でのコピーと比較して，データ移動帯域の向上と CPU 使用率の削減が得られた．

以下本稿では，第 2 節で提案手法のシステムアーキテクチャ，第 3 節で提案手法の実装，第 4 節で実験結果，第 5 節で今後の予定を述べ，最後に第 6 節でまとめる．

2. システムアーキテクチャ

初めに，現在のコンピュータシステムでの I/O デバイス間のデータ移動時の問題点を説明する．現在のコンピュータシステムでは，すべての I/O デバイス間のデータ移動をユーザプログラムとして実現している．しかし，ユーザ層のデータ移動は移動帯域が低く，ハードウェアリソース消費のオーバヘッドが大きい．これはユーザ層のデータ移動では，ユーザモードとカーネルモードの間のコンテキストスイッチや，ユーザメモリとカーネルメモリの間の I/O データのコピーが生じるためである．メモリコピーは CPU リソースを消費するだけでなく，メモリからデータをリードし，そのデータを再びメモリにライトするためメモリバスの帯域も消費する．さらに CPU キャッシュとファイルキャッシュメモリは，I/O デバイス間の移動が 1 度限りでありキャッシュする必要のないデータのために消費される．

我々は，I/O デバイスの制御やそれらの間のデータ移動はユーザ層より低い層で行い，ユーザ空間はより複雑なアプリケーションプログラムを実現するために使用すべきだと考えている．このため，提案手法ではユーザからの関数呼出しにより I/O デバイス間でデータを移動できるようにし，それ以外のデータ移動の制御はユーザの関与なしに行えるようにした．

提案手法では，全ての I/O デバイスの制御とデータ移動をカーネル空間で行う．カーネル空間ではコンテキストスイッチやメモリコピーが生じないため，広帯域でハードウェアリソースの消費を抑えたデータ移動を行うことができる．また提案手法ではファイルキャッシュへのデータのキャッシュも行わない．これは I/O デバイス間を移動するほとんどのデータの移動は 1 度限りであるという状況を想定しているためであり，そのようなデータは CPU キャッシュにキャッシュする必要がない．また提案手法では，データ移動を行う I/O デバイスに対するリード処理とライト処理を並列に行い，さらにそれぞれの I/O デバイスに対する I/O 要求を多重化することでデータ移動帯域を高めることができる．

提案手法を一般のコンピュータシステムで広く利用可能とするためには，OS のコードに対する変更は可能な限り小規模にする必要がある．提案手法では，I/O デバイスの間でデータを移動する機能を，カーネルの一部ではなくデバイスドライバモジュールとして実装することで OS コードへの変更を抑える．これにより，様々なデバイス制御機能をデバイスドライバとして実現でき，あらゆる I/O デバイスの間のデータ移動に対応できる．この特徴は，今後様々な広帯域 I/O デバイスを広帯域データ処理システムに導入するために重要である．

将来的には，I/O デバイス間のデータ移動機能を I/O デバイスやブリッジなどの I/O バスデバイスのアシスト機能を用いて実現することが可能である．そのような環境では，本手法がデバイスドライバ層で行っている I/O デバイスの制御やデータ移動の一部をハードウェアにオフロードする．これにより，データ移動性能の向上と CPU やメモリバスなどのハードウェア資源の消費の削減を実現できる．

3. 実装

我々は，提案手法による I/O デバイス間のデータ移動をデバイスドライバ層に実装し，SCSI ディスクデバイス間のデータコピーを行うプロトタイプを作成した．本節で

Vol.2011-ARC-195 No.11
Vol.2011-OS-117 No.11
2011/4/13

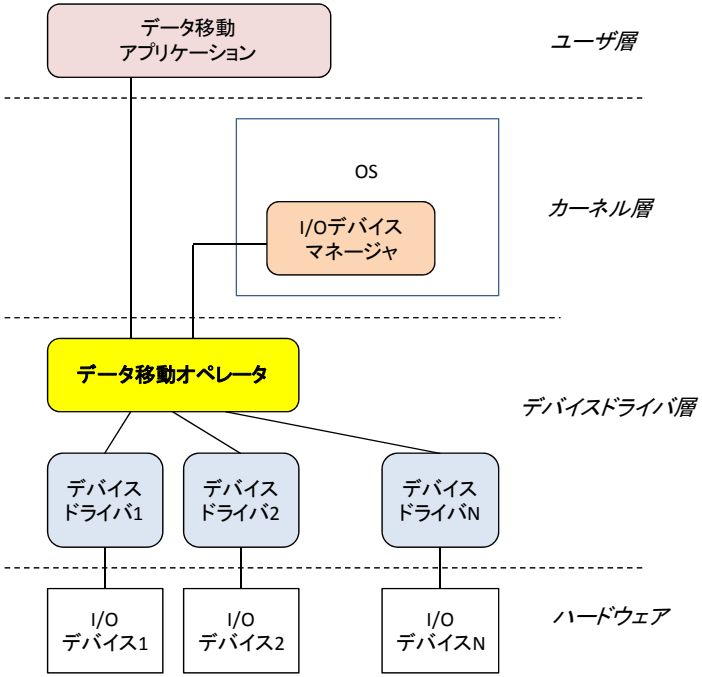

図 1　デバイスドライバ層によるデータ移動のアーキテクチャ

は，第 3.1 節でデータ移動のアーキテクチャについて，第 3.2 節で SCSI ディスクのデータコピープロトタイプについて述べる．

3.1　I/O デバイス間データ移動のアーキテクチャ

提案手法では I/O デバイス間のデータ移動をドライバ層で行う．図 1 にそのアーキテクチャを示す．

3.1.1 データ移動アプリケーション

データ移動アプリケーションは I/O デバイス間のデータ移動を行う関数の呼出しを行う。このとき，データ移動アプリケーションはソース及びターゲットデバイス，それらのオフセットアドレス，および移動データサイズの指定を行う．データ移動に良く用いられる関数は，プログラミングを補助するライブラリ関数として実装することも可能である．

3.1.2 データ移動オペレータ

データ移動オペレータはローダブルモジュールとして実現され，I/O デバイス間のデータ移動処理を行う．データ移動オペレータのデータ移動機能は，データ移動アプリケーションから OS のインタフェースを経由して呼び出される．

データ移動機能が呼び出される場合，ソースデバイスとターゲットデバイスはそれらのノード名で指定される．データ移動オペレータは，I/O デバイスマネージャにソースデバイスとターゲットデバイスの情報を保持する構造体のアドレスを問い合わせる．これらの構造体は，I/O デバイスを管理するために OS で定義されているものであり，システムのブート時に I/O デバイス毎に作成される．これらの構造体が保持する情報は，I/O デバイス間のデータ移動においてソースデバイスとターゲットデバイスを指定し，それらのデバイスドライバのデータ I/O 関数を呼び出すために必要となる．

データ移動オペレータは 2 つのメモリ空間を保持する．1 つは I/O コマンドのための空間である．データ移動オペレータは I/O コマンドを作成し，デバイスドライバに渡す．もう 1 つは，I/O デバイス間で移動するデータのデータバッファとして用いられるための空間である．データ移動では，ソースデバイスがデータバッファに direct memory access (DMA)を用いてデータをライトし，ターゲットデバイスもまた DMA を用いてそのデータをリードする．これにより，冗長なメモリコピーやそれによるメモリバス帯域の消費なく I/O デバイス間でデータ移動を行うことができる．今日使用されている多くの I/O デバイスは 1 つの DMA 命令のサイズが数百 KB 程度であり，データ移動オペレータのデータバッファサイズもこの値に設定している．データバッファのアドレスリストは，I/O デバイスドライバに渡す I/O コマンドに付加する．

データ移動オペレータのデータバッファを介した I/O デバイス間のデータ移動では，データ移動オペレータが I/O デバイスのドライバに I/O コマンドを渡すことで，デバイスとデータバッファの間のデータ移動を行う．図 2 にデータ移動を説明する図を示す．I/O コマンドの形式はデバイスの種別ごとに定義されている．従来のアプリケーションプログラムでは，I/O デバイスがアクセスされると，I/O コマンドが OS の I/O スタックを介してデバイスドライバのコールバックインタフェースに渡される．一方提案手法では，I/O コマンドはデータ移動オペレータからそのデバイスドライバのコールバックインタフェースに渡される．そして，渡された I/O コマンドはデバイスドライバのコマンドキューにキューイングされる．データ移動オペレータはキューイングした I/O コマンドにデータ移動オペレータ自身のコールバックハンドラを指定することで，データ移動オペレータが発行した I/O コマンドの完了時にのみ完了処理を行う．本手法により，データ移動オペレータによる I/O 処理とユーザプログラムによる I/O 処理をデバイスドライバ層で同時に行うことが可能となる．ここで，データ移動オペレータはデバイスドライバをコールバック関数のインタフェースから呼び出すため，呼出し方法は個別のデバイスドライバが実装する関数名に依存せず，同種のデバイスであれば全ての I/O デバイス提供事業者のデバイスに対応できる．

Vol.2011-ARC-195 No.11
Vol.2011-OS-117 No.11
2011/4/13

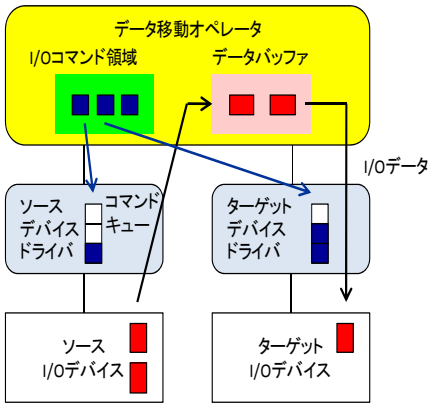

図 2　データバッファを介したデータ移動

データ移動オペレータは広帯域データ移動を実現するため，ソースデバイスに対するリード処理とターゲットデバイスに対するライト処理を並行して行うことで，各デバイスの I/O 帯域をできる限り活用する．また，データ移動オペレータは 1 つの I/O デバイスに対し同時に複数の I/O コマンドを発行する．このような処理を行うため，データ移動オペレータは内部に複数のデータバッファを保持する．それらのバッファに対し，データ移動オペレータは，各 I/O デバイスとの間のデータ移動を並列で行う．あるデータバッファに対するデータ移動処理が完了すると，データ移動オペレータは次のアドレスのデータ移動処理をデータバッファに割り当てる．この結果，ソースデバイスとターゲットデバイスは連続にアクセスされ，データ移動において I/O デバイスの先読み機能やキャッシュ効果を活用することができ，広帯域データ移動が可能となる．

3.1.3 I/O デバイスマネージャ

I/O デバイスマネージャはデータ移動処理対象となる I/O デバイスのリストを管理する．コンピュータの起動時や I/O デバイスのホットプラグ時では，OS は個々のデバイスを調査し，それぞれのデバイスの情報を保持する構造体を作成する．I/O デバイスマネージャは，OS が I/O デバイスを調査するときに自身が管理するリストにデバイスのエントリを作成する．これらのエントリはデバイスのノード名と OS が保持するデバイスの構造体を関連付ける．

I/O デバイスマネージャはデータ移動オペレータに対し、カーネル空間に OS が管理するこの構造体のアドレスを検索する機能を提供する．検索にはデバイスのノード名が検索キーとして使用される．

3.1.4 デバイスドライバ

提案手法では，各 I/O デバイスのデバイスドライバを変更なく利用する．デバイスドライバを使用することで，提案手法のソフトウェア規模を小さく抑えることが可能となる．例えば I/O デバイスをシステムで動作させるために必要なコンフィグレーション処理は各デバイスドライバが行う．このコンフィグレーション処理は個々の I/O デバイスにより異なるため，デバイスドライバを使用しなければ個別のデバイスに対応する処理の実装が必要となる．

データ移動オペレータはデータ移動処理においてドライバのデータパスの関数のみを用いる．前述のように，それぞれのデバイスドライバのデータパス関数は，それが登録されているコールバック関数の名前を用いて呼び出されるため，個々のデバイスドライバに依存しない．

3.2 SCSI ディスクコピープログラム

提案手法を用いて 2 つの SCSI ディスクの間でデータをコピーするプログラムを作成した，ソフトウェアプラットフォームは Linux kernel 2.6.18 (CentOS 5.5)を用いた．データ移動オペレータはローダブルモジュールとして実装した．I/O デバイス間のコピーを行う場合，データ移動オペレータは SCSI コマンドを作成しデバイスドライバに渡す．作成するコマンドはリードコマンドとライトコマンドの 2 種のみである．データ移動オペレータはコピープログラムと *ioctl* を用いて通信する．OS コードへの変更は 2 点だった．1 点は I/O デバイスマネージャの実装である．これはカーネルへ組み込むデバイスドライバとして実装し，ライン数はコメントを含め 300 行程度だった．2 点目は SCSI スタックの変更である．これは 4 行だけの追加であり，この変更は OS による I/O デバイスの調査時に I/O デバイスマネージャがデバイスのエントリを作成したり，デバイスがシステムから外される時にエントリを消去したりするために必要だった．

4. 実験結果

プロトタイプを用いた評価として，2 つの PCE Express (PCIe)型高速 SSD の間でのデータコピー性能を測定した．評価に用いた 2 つの SSD は，単一サーバの 2 つの PCIe スロットに挿入した．表 1 に用いた評価環境を示す．

表 1　評価環境

CPU	Intel® Xeon® CPU L5520 4 cores 2 processors
Memory	8GB
SSD	OCZ Z-DRIVE M84 PCE-EXPRESS SSD 256GB

Vol.2011-ARC-195 No.11
Vol.2011-OS-117 No.11
2011/4/13

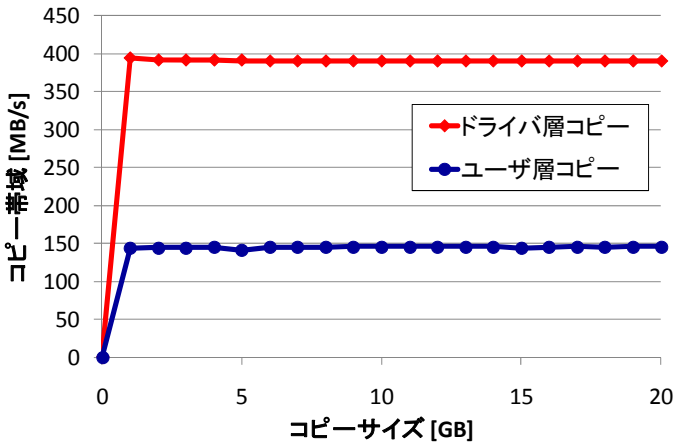

図 3 提案手法(ドライバ層コピー)のコピー帯域

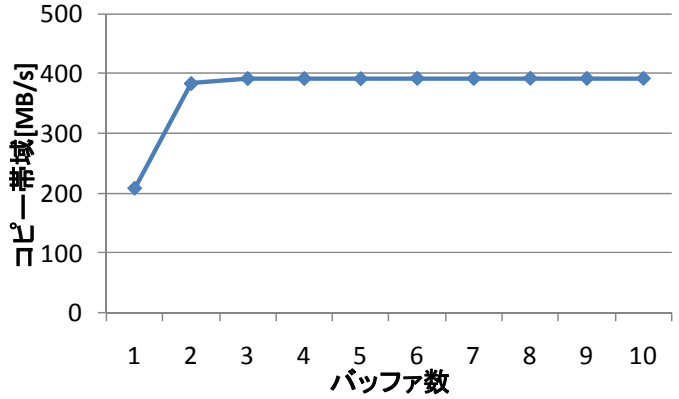

図 4 バッファ数変更時の提案手法のコピー帯域

提案手法による性能との比較のため，我々はユーザ空間でデータコピーを行うプログラムを作成した．以後，提案手法のプロトタイプをドライバ層コピー，ユーザ空間でのコピーをユーザ層コピーと呼ぶ．ユーザ層コピーは，ソース SSD からユーザ空間内のメモリにデータをリードし，リードしたデータをターゲット SSD にライトする．これらの I/O 処理はバッファード I/O によって行われる．リード処理とライト処理はユーザ入力で指定されたデータ量のコピーが完了するまで繰り返す．広く用いられている *dd* もこのような処理を行う．データがターゲット SSD に確実に書き込まれるようにするため，ターゲット SSD へのアクセスには同期 I/ O を用いた．ユーザ層コピーではソース SSD はバッファード I/O によってアクセスされるため，コピーするデータの一部は SSD ではなくキャッシュメモリからリードされる．ユーザ層コピーにおけるユーザ空間内のデータバッファは 512KB に設定した．これは提案手法のプロトタイプの内部で用いているデータバッファのサイズと同じである．

図 3 に測定したデータコピーの帯域を示す．提案手法であるドライバ層コピー，性能比較のためのユーザ層コピーとも全てのコピーサイズで帯域は一定だった．ドライバ層コピーでは 4 つのバッファを使用して並列 I/O を行った．ドライバ層コピーでは，ユーザ層コピーと比較して最大 177%の性能向上が得られる。これはコピー時間を 64%削減するものである．

図 4 はドライバ層コピーが並列 I/O で使用するデータバッファ数を変化させた場合のコピー帯域である．バッファ数は SSD に対する I/O 命令の並列度を決める．図 4 のコピー帯域は，バッファ数が 3 以上の場合，帯域が飽和することを示している．これは実験で用いた SSD の I/O 帯域がボトルネックとなっているためである．もし I/O 帯域がさらに高性能な SSD を用いた場合，帯域が飽和するバッファ数は増加する．これは I/O 帯域を全て使用するために必要となる並列度が増加するためである．

図 5(a)は各方式のデータコピーにおける実験サーバの単位時間あたりの CPU 使用率である．CPU 使用率は 2 つの場合で同水準である．ここで図 3 に示したようにデータコピーの帯域はユーザ層コピーより提案手法であるドライバ層コピーのほうが高い．そこで図 5(b)に，図 5(a)で示した CPU 使用率を各手法のコピー帯域で割った，帯域あたりの CPU 使用率を示す．図 5(b)によると，提案手法を用いることにより帯域あたりの CPU 使用率を 62%削減できる．なお図 5(a)に示した結果は評価で用いた 8 コアのプラットフォームの平均値である．このため 1 コアあたりの CPU 使用率に換算すると，双方のデータコピープログラムは 1 つのコアの約 50%のプロセッサ資源を使用していることになる．

図 5(a)を参照すると，ユーザ層コピーでは，ドライバ層コピーより多くの CPU 資源をシステム時間に使用していることがわかる．これは CPU 資源がデータコピーやファイルキャッシングを含むファイルシステム処理に使用されたためだと考えられる．一方ドライバ層コピーでは，より多くの資源をハード割り込みと I/O 待ちに使用している．これはドライバ層コピーがカーネル空間でデータ移動を行っており，図 3 が示すようにユーザ層コピーと比べて 2 倍以上の I/O 処理を行っているためである．

我々はまた，データコピーの間に発生する SSD からの割り込み数を測定した．ドライバ層コピーでは，ソース SSD とターゲット SSD の双方で発生する割り込み数は 1GB

Vol.2011-ARC-195 No.11
Vol.2011-OS-117 No.11
2011/4/13

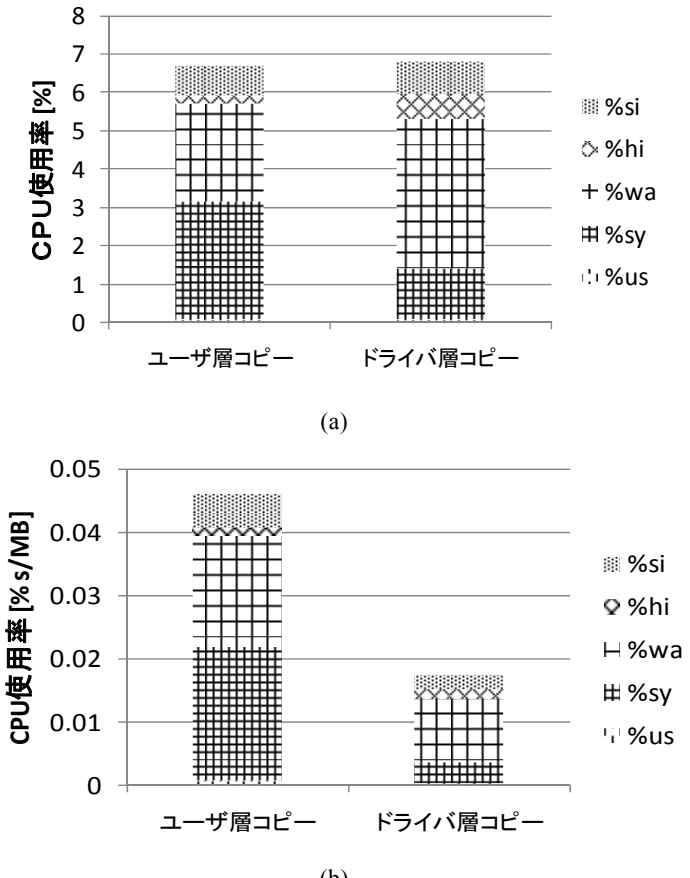

図 5　データコピー時の CPU 使用率 (a) 単位時間あたり　(b)コピー帯域あたり
si: Softwatware Interrupts, hi: Hardware IRQ, wa:iowait,
sy: System CPU Time, us: User CPU Time

のデータコピーにおいて正確に 2048 回だった．これはドライバ層コピーが，1GB のデータコピーにおいて512KBを単位とするI/O処理を2048回行っているためである．一方，ユーザ層コピーでは，割り込み数は約 4 倍であり，10 回の測定で 7843 回～8783 回のばらつきがあった．割り込み数の 4 倍の増加とその数値のばらつきは，ファイルや SCSI スタックで発生する I/O 命令の分割や結合と，ファイルキャッシュによってキャッシュされているデータが与える影響に起因すると考えている．

以上の実験結果は，提案手法による I/O デバイス間のデータ移動法が，従来用いられているユーザ層でのデータ移動法と比較して，より広帯域でハードウェアリソース消費が小さいデータ移動機能を提供することを示している．提案手法は，並列 I/O により SSD 帯域の限界性能でデータを移動することができ，コンテキストスイッチやメモリ内でのコピーを行わず，CPU とメモリバス帯域の消費を抑える．一方，ユーザ層コピーは性能が低く CPU 使用率が高い．これは，ユーザ層コピーは SSD 帯域の限界性能でデータを移動させることができず，また，コンテキストスイッチやメモリコピーを行うため CPU 消費量が増加するためである．

5. 今後の予定

本稿では，SCSI ディスクデバイス間でのデータ移動を評価した．提案手法の目的は異なる様々な I/O デバイスの間でデータ移動が行えるプラットフォームを提供することであり，今後は異種デバイス間でのデータ移動に取り組む．

また，我々は以前より標準イーサネットを用いてコンピュータと PCIe 準拠 I/O デバイスを接続する技術を提案している[6]．我々はこの技術を I/O デバイス間のデータ移動のためのハードウェア補助機能に応用することを検討している．これによりネットワークで接続されたデバイスの間で直接データ移動を行うことが可能となる．この直接データ移動は，ホストコンピュータのメモリバスや I/O バスの帯域を使用することなく I/O デバイスの間でデータ移動を行うことを可能とする．

6. まとめ

本稿では，I/O デバイスの間でデータを広帯域に移動する手法を提案した．提案手法はユーザ空間からの関数呼出しで I/O デバイス間のデータ移動を行うことを可能とする．データ移動はドライバ層で行われ，広帯域なデータ移動を実現できると共に．コンテキストスイッチ，メモリコピー，ファイルキャッシュを行わないことでハードウェア資源の使用を抑制する．提案手法ではデータ移動機能をドライバとして実現することにより OS コードへの変更を小規模に抑えることができ，様々な I/O デバイスに対するデータ移動機能を実装することが可能となる．

提案手法を用いて作成した 2 つの SCSI ディスク間でのデータコピープログラムでは，ユーザ層でのデータコピーと比較して 177%の帯域向上と 62%の CPU 使用率削減が得られた．これらの評価結果は，提案手法が I/O デバイス間の広帯域データ移動を実現する技術として有力な候補となり得ることを示している．

Vol.2011-ARC-195 No.11
Vol.2011-OS-117 No.11
2011/4/13

謝辞 本研究に関し、多くの有益なコメントをいただいた長谷部賀洋氏と原幹雄氏、並びに日頃ご指導頂く矢野由紀子氏、桐葉佳明氏、加納敏行氏をはじめ NEC 関係各位に感謝します。

本研究の一部は、総務省の委託研究「グリーンネットワーク基盤技術の研究開発」プロジェクトの成果である。

参考文献

1) E. B. Nightingale, O. Hodson, R. Mcllroy, C. Hawblitzel, and G. Hunt, “Helios: Heterogeneous Multiprocessing with Satellite Kernels,” 22nd ACM Symposium on Operating Systems Principles (SOSP’09), 2009.
2) J. C. Mogul, “TCP offload is a dumb idea whose time has come,” In Proceeding of the 9th conf. on Hot Topics in Operating Systems (HOTOS’03), vol. 9, 2003.
3) J. Pasquale, E. Anderson, and P. K. Muller, “Container Shipping: Operating System Support for I/O Intensive Applications,” IEEE Computer, vol. 27, issue 3, pp. 84-93, 1994.
4) V. S. Pai, P. Druschel, and W. Zwaenepoel, “IO-Lite: A Unified I/O Buffering and Caching System,” ACM Trans. on Computer Systems, vol. 18 no. 1, pp. 37-66, 2000.
5) Y. Weinsberg, D. Dolev, T. Anker, M. B. Yehuda, and P. Wyckoff, “Tapping into the Fountain of CPUs – On Oper-ating System Support for Programmable Devices,” 13th Int. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS’08), 2008.
6) J. Suzuki, Y. Hidaka, J. Higuchi, T. Yoshikawa, and A. Iwata, “ExpressEther – Ethernet-Based Virtualization Technology for Reconfigurable Hardware Platform,” In Proc. of 14th IEEE Symp. on High-Performance Intercon-nects (HOTI’06), pp. 45-51, 2006.