Vol.2010-OS-113 No.8
2010/1/28

ゼロコピー通信処理を可能にする実メモリ交換機能の提案

門　直史[†1]　田端利宏[†1]　谷口秀夫[†1]

Ethernet 通信環境において 1GbEthernet や 10GbEthernet のような高速な通信路が普及した結果，分散処理環境を構成する計算機間のデータ通信処理全体におけるプロセッサ処理の比率は高くなってきている．このため，各パケットに対する送受信処理，特にデータ複写処理を削減することが全体の速度向上に大きな意味を持つ．既存手法によるゼロコピー通信では，データ受信時のゼロコピー通信の実現は難しい．本稿では，実メモリ交換機能を提案し，データ送信処理に加えデータ受信処理もゼロコピー通信とすることができることを示す．また，***Tender*** の資源の分離と独立化を利用してデータ送信処理とデータ受信処理においてゼロコピー通信を実現した Ethernet 通信機構の実現方式について述べ，評価を行う．

A proposal of physical memory exchange function to enable Zero-copy communication processing

Naofumi Kado,[†1] Toshihiro Tabata[†1]
and Hideo Taniguchi[†1]

The ratio of the processor processing in the whole data communication processing between a computer constituting distributed processing environment rises as a result that high-speed channels such as 1GbEthernet and 10GbEthernet spread in Ethernet communication environment. Thus reducing data copy processing contribute for total speedup transmission and receiving processing for each packet. As for the realization of Zero-copy communication at the time of the data receiving, it is difficult to be Zero-copy communication by the existing technique. In this paper, we propose physical memory exchange function and show what we can do with Zero-copy communication processing using separation and the independence of resources of ***Tender***. We also describe and evaluate data transmission and receiving a realization method of the Ethernet communication mechanism that realized Zero-copy communication.

1. はじめに

計算機間の通信路の高速化が進み，計算機間を高速な通信路で結んだ分散処理環境の構築が容易となってきている．このような分散処理環境では，計算機同士の協調した動作のために，データの送受信を行う必要がある．このため，各計算機間のデータ通信性能が全体の処理性能に与える影響は大きく，分散処理環境下におけるデータ通信は高速である必要がある．分散処理環境の場合，遠隔の計算機とのデータ通信を行うために Ethernet を用いることが多い．Ethernet 通信環境では，1GbEthernet や 10GbEthernet のような高速な通信路の普及に伴い，データ通信処理全体におけるプロセッサ処理の比率が高くなっている．このため，TCP/IP のプロトコル処理を H/W が肩代わりする TOE（TCP/IP Offload Engine）の研究[1]，および TCP/IP 通信処理中のプロセッサ処理時間の削減が進められている[2]．

一方で，シンクライアントシステムやネットワークストレージシステムが普及して大容量データに対する送受信処理が頻発するようになり，メモリ間データ複写処理によるオーバヘッドが増加している．大容量データに対するデータ送受信処理時間を削減する手法の 1 つとしてゼロコピー通信がある．PM/Ethernet[3)4)5)] や OPEN-MX[6] といった高速なデータ通信機構では，送信対象データを格納した領域を物理メモリにピンダウンしておくことでカーネル空間を介さずに NIC への直接の DMA 転送を可能にし，データ送信時のゼロコピー通信を実現している．しかし，データ受信処理の場合には同様の手法でゼロコピー通信を実現することはできない．これは，受信したパケットは最初にカーネル空間にマッピングされた受信バッファに格納されるためである．

ここでは，実メモリ交換機能を提案する．実メモリ交換機能は，2 つの仮想メモリ空間上の領域の間で仮想メモリに割り当てている実メモリを交換することでメモリ間データ授受を複写レスで実現する機能である．これにより，データ送信時だけでなくデータ受信時でもゼロコピー通信を実現してデータ送受信処理時間の削減を実現する．

Tender オペレーティングシステム[7]（以下 ***Tender*** と略す）では，OS の操作する対象を資源として，分離し独立化している．これにより，資源「実メモリ」のように既存 OS では単体で存在できない資源を管理して操作することが可能である．本稿では，この資源の分離と独立化を利用して実メモリ交換機能を ***Tender*** に実現し，データ送受信ともにゼロ

†1 岡山大学 大学院自然科学研究科
Graduate School of Natural Science and Technology, Okayama University

コピー通信を可能とした Ethernet 通信機構の実現方式について述べ，評価を行う．

2. 既存手法の問題点

2.1 データ送信時のゼロコピー通信

OPEN-MX では，送信対象データのサイズに応じてデータ複写による送信とゼロコピー送信を使い分けている．送信対象データが 128 バイト未満の場合，送信バッファ（skbuff）のデータ格納領域へ送信対象データを複写する．送信対象データが 129 バイト以上 32KB 未満の場合，ユーザプロセスとカーネルの共有領域をピンダウンしておき，この共有領域へ送信対象データを複写する．複写を行った後，共有領域を skbuff にアタッチしてデータ送信を行う．送信対象データが 32KB 以上の場合，送信対象データを格納する領域に対してピンダウンを行い，skbuff にアタッチすることでゼロコピーでの送信を行う．

SCore では，クラスタを構成する計算機間での高速なデータ通信を提供する PM ライブラリの 1 つとして PM/Ethernet が実装されている．PM/Ethernet では，データ送信に利用するバッファを確保し，この送信バッファ上の送信対象メッセージを構築した後にデータ送信を行う．

上記のように，特殊なハードウェアを利用しない一般的な Ethernet 通信環境でのゼロコピー通信を可能にするために送信対象データを格納している領域を物理メモリにピンダウンしておき，カーネル空間を介さずに NIC へ DMA 転送を行うことでゼロコピー通信を実現している．

2.2 データ受信時のゼロコピー通信

OPEN-MX では，データ受信時のゼロコピー通信は実現されておらず，1 回以上のデータ複写が発生する．受信データが 32KB 未満の場合，ユーザプロセスとカーネルの共有領域をピンダウンしておき，受信バッファ（skbuff）からこの共有領域へ受信データを複写し，その後受信データ格納領域へデータ複写を行う．受信データが 32KB 以上の場合，受信データを格納する領域に対してピンダウンを行い，skbuff から受信領域へデータ複写を行う．

PM/Ethernet では，PACS-CS でのデータ通信用に設計された PM/Ethernet-HXB でゼロコピー受信の手法が提案されている．これは，受信用 skbuff に ID を与えておき，データ受信前に全ての skbuff をユーザ空間にマッピングしておく．デバイスドライバでの受信処理終了後，受信データを格納した skbuff の ID をユーザプロセスに通知することでゼロコピーで受信データを参照することが可能になる．

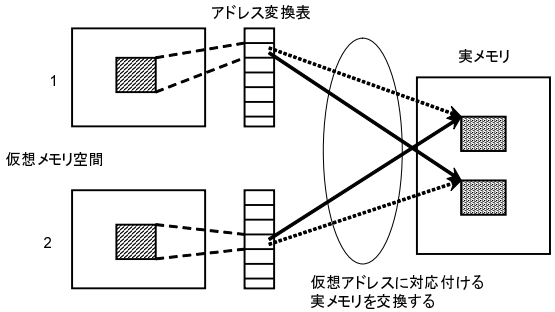

図 1 実メモリ交換機能

2.3 問 題 点

ピンダウンを用いる手法では，データ受信時にはゼロコピー通信を行うことができない．この理由を Linux を例に述べる．NIC がパケットを受信すると最初に受信用 skbuff に格納される．この skbuff は，カーネルによって確保されており，カーネル用の仮想空間にマッピングされている．このため，ピンダウンした受信領域を用意しても NIC から受信データ格納先領域への直接の DMA 転送を行うことはできない．

RDMA（Remote Direct Memory Access）[8] のようにデータ送受信処理ともに NIC とユーザプロセスの仮想メモリ空間上の領域間で DMA 転送を行う手法も提案されている．しかし，RDMA に対応した NIC が必要になりコストが高くなる．

3. 実メモリ交換機能

3.1 考 え 方

実メモリ交換機能は，仮想メモリ空間上の n ページの大きさの 2 つの領域について，仮想メモリに対応する実ページを交換する機能である．実メモリ交換機能の様子を**図 1** に示す．仮想メモリ空間上の領域は，アドレス変換表によって仮想アドレスと実メモリを対応付けられており，領域のデータを参照する際はアドレス変換表によって仮想アドレスを実アドレスに変換して参照する．本機能では，仮想メモリ空間上の 2 つの領域について，それぞれの領域に対応するアドレス変換表を参照して各ページの仮想アドレスに対応する実アドレスを交換し，アドレス変換表を更新する．これにより，2 つの領域間で複写レスでのメモリ間データ授受を実現する．この時，複数ページ分の実メモリに対しても交換を行うことが可能である．また，2 つの領域は異なる仮想メモリ空間上に存在していても良い．

なお，本機能は，アドレス変換表の実メモリの内容を変更するのみであるため，仮想アドレスに対しての変更は発生しない．

3.2 期待される効果

実メモリ交換機能を利用することで期待される効果として，以下のものがある．

(1) 複写レスでのメモリ間データ授受を実現
OS 処理にとって，メモリ間複写は非常に大きなオーバヘッドであり，データサイズに比例して処理時間が増加する．このため，複写レスでのメモリ間データ授受を実現することによって，OS 処理全体の処理時間を削減することが期待できる．このため，データ複写処理が主となる処理，例えばプロセス間通信や計算機間での LAN 通信に関しては，実メモリ交換を実現することによる効果は特に大きいと考えられる．

(2) 仮想アドレスの再マッピングと比べ，領域の解放確保を削減
複写レスでのメモリ間データ授受を行う他の手法としては，仮想アドレスの再マッピングがある．この処理では，仮想メモリ空間上の領域の移動を伴うため，領域の解放や再確保といった処理が必要になる場合がある．一方，実メモリ交換機能の場合は，仮想メモリ空間上の領域の移動は発生しないため，領域の解放や再確保といった処理は不要である．

次に欠点について述べる．

(1) 実メモリ交換を行う単位は，ページの整数倍
実メモリ交換を行った場合，交換対象のページに格納されているデータ全てが交換されてしまう．このため，交換を行うページの実メモリ領域内に交換を望まないデータを格納しておくことはできず，内部断片化が発生しやすくなる．

(2) TLB フラッシュが必要
ページ変換テーブルの更新を行うため，TLB フラッシュが必要になる．

4. *Tender* への実装と評価

4.1 資源の分離，独立化

Tender では，OS の操作する対象を資源として，分離し独立化している．資源には，資源名と資源識別子を付与し，資源操作のインタフェースを統一している．更に，各資源を操作するプログラム部品（資源管理処理部と呼ぶ）を資源ごとに分離し，共有プログラムを排除している．また，各資源の管理情報も資源ごとに分離し，各資源の管理表の間の参照関係を禁止している．

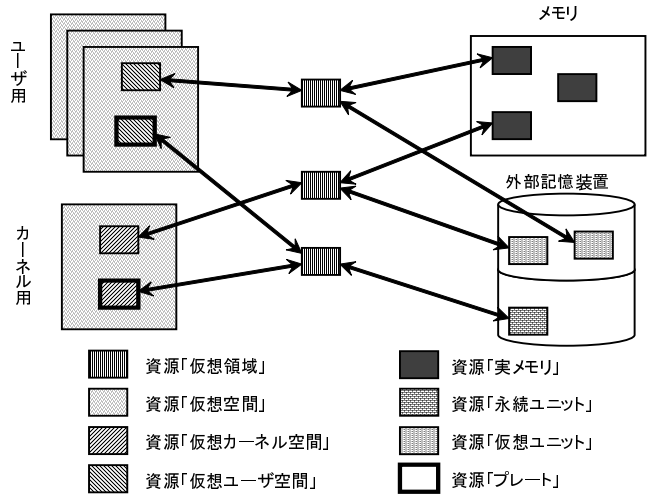

図 2 ***Tender*** のメモリ管理機構

このように，資源の分離と独立化を行うことで，資源の事前用意や保留により，資源の作成や削除を伴う処理を高速化している．さらに，プログラムを部品化できるため，機能の追加や変更が容易になっている．

4.2 メモリ管理機構

Tender のメモリ管理機構を**図 2** に示す．資源「仮想空間」とは，特定のアドレス領域を持つ仮想的な空間であり，仮想アドレスから実アドレスへのアドレス変換表に相当する．資源「仮想領域」は，メモリイメージを仮想化した資源であり，実体は実メモリもしくは外部記憶装置に存在する．仮想領域管理のインタフェースを**表 1** に示す．資源「仮想領域」を作成する際に資源「実メモリ」を割り当てる場合，作成する資源「仮想領域」のサイズに応じてページ単位で資源「実メモリ」を作成して割り当てる．***Tender*** での 1 ページは 4KB である．資源「仮想ユーザ空間」は，メモリイメージを仮想化した領域である資源「仮想領域」をユーザ用の資源「仮想空間」に貼り付けることで作成できる．「貼り付ける」とは，仮想アドレスを実アドレスに対応付けすることであり，具体的には，当該の仮想アドレスに対応するアドレス変換表のエントリに，実アドレスまたは外部記憶装置のアドレスを設定する．一方，仮想アドレスと実アドレスの対応付け解除を「剥がし」と呼ぶ．資源「仮想領域」の実体は，資源「実メモリ」または外部記憶装置上に存在する．なお，外部記憶装置上

表 1 仮想領域管理の提供インタフェース

形式	機能
create_vr(size, mem, dk, vr_op, name)	size で指定された大きさの仮想領域を確保する．mem=1 の場合，実メモリを確保しない．mem=2 の場合，vr_op にしたがって実メモリを確保する．
delete_vr(vrid)	vrid で指定した仮想領域を削除する．
read_vr(vrid, vaddr, offset)	vrid で指定した仮想領域に対応するディスク領域の offset からのデータをメモリ上の vaddr で指定するアドレスに読み込む．
write_vr(vrid, vaddr, offset)	vrid で指定したメモリ上の vaddr で指定するアドレスのデータを vrid で指定した仮想領域に対応するディスク領域の offset から書き出す．
ctrl_vr(vrid, vr_op, offset, size, *name, vaddr, reserve, *buff, change_size, offset1, vrid2, offset2, pagesize, *raddr1, *raddr2)	vr_op=0x1 の場合，vrid で指定した仮想領域のサイズを返す．vr_op=0x2 の場合，vrid で指定した仮想領域の offset からの実アドレスを返す．vr_op=0x100 の場合，仮想領域の offset から size 分の実メモリを割り当てる．vr_op=0x200 の場合，vrid で指定した仮想領域の offset から size 分の実メモリを解放する．vr_op=0x8000 の場合，vrid で指定した仮想領域の offset1 から pagesize 分の実メモリと vrid2 で指定した仮想領域の offset2 から pagesize 分の実メモリを交換する．

表 2 実メモリ交換機能の提供インタフェース

形式	機能
exchange_pmem(vrid1，vrid2，offset1，offset2，pagesize)	仮想領域 vrid1 の offset1 からの実メモリと仮想領域 vrid2 の offset2 からの実メモリを pagesize 分交換する

の領域の種類として，資源「永続ユニット」と資源「仮想ユニット」の 2 種類がある．資源「仮想カーネル空間」は，資源「仮想領域」をカーネル用の資源「仮想空間」に貼り付けることにより作成される．

4.3 実メモリ交換機能

Tender で実現した実メモリ交換機能の提供インタフェースを**表 2** に示す．実メモリ交換機能は，vrid1 と vrid2 に対応付けられている資源「実メモリ」を交換することで仮想空間 1 と仮想空間 2 の間のデータ授受を実現する．この時，仮想領域を特定する情報としてカーネルに対して仮想領域の資源識別子を渡す．カーネルは，この仮想領域識別子から仮想領域を特定し，対応付けられている実メモリ情報を参照する．この際，仮想領域に対応付けている実メモリの先頭からのオフセットと交換するページ数を指定することで複数ページの交換も可能である．

4.4 資源「入出力」

Tender では，NIC を含む入出力デバイスを統一的に管理する資源として資源「入出力」を持つ[9]．資源「入出力」は入出力管理によって管理制御され，入力領域と出力領域を用いてデータの入出力を行う機能を提供する資源である．資源「入出力」は，HDD，Myrinet，および Ethernet といった入出力デバイスを抽象化してどの入出力デバイスに対しても統一的なインタフェースを提供する．

資源「入出力」では，入出力を生成して各入出力ごとに出力（データ送信に相当）と入力（データ受信に相当）を行う．この時，入出力の生成時に出力の個数と入力の個数を決定しており，複数のデータを連続して送受信する場合に逐次的に送受信処理を行うのではなく一括して送受信処理を行うことができる．これにより，カーネルへの処理依頼回数や受信待ち状態のプロセスを起床させる処理を削減できる．また，***Tender*** では，データ送信時に使用する mbuf は ***Tender*** の初期化処理時に確保しておき繰り返して使用する．

入出力管理の提供インタフェースを**表 3** に示し，その機能を以下に説明する．

(1) 生成
入出力の種類により指定された入出力デバイスに対する資源「入出力」を生成する．入出力識別子を返却する．

(2) 削除
指定された資源「入出力」を削除する．

(3) 入力
指定された入出力識別子に対応する資源「入出力」の入力領域へデータを入力する．

(4) 出力
指定された入出力識別子に対応する資源「入出力」の出力領域からデータを出力する．

(5) 入出力への入出力領域の登録，入出力相手装置のアドレス情報登録
指定された入出力識別子に対応する資源「入出力」に対し，入力領域と出力領域の登録，入出力相手装置へのアドレス情報登録を行う．

入出力の種類として，通信相手を特定する情報（通信路の種類など）に加え，入出力装置を特定する情報も含めている．

資源「入出力」によるデータ送受信処理の流れを以下に説明する．なお，データ送受信処理を行う前に入出力領域の登録を行い，入力領域と出力領域の実アドレスと仮想領域識別子を登録しておく必要がある．

表 3 入出力管理の提供インタフェース

形式	機能
get_io(dev_no，numofinput，numofoutput，ioid)	入出力の種類，入力の個数，出力の個数，入出力識別子を dev_no，numofinput，numofoutput，ioid で指定して入出力を生成する．
free_io(ioid)	ioid で指定した入出力を削除する．
input_io(ioid，∗size，∗position)	ioid で指定した入出力に対して入力を行う．入力の位置を指定する場合，position で指定した配列に格納する．size には入力したデータサイズを格納する．
output_io(ioid，∗vars，∗size，∗position)	ioid で指定した入出力から出力を行う．出力の位置を指定する場合，position で指定した配列に格納する．size には出力するデータサイズを格納する．
ctrl_io(ioid，io_op，∗buff，∗input_vmid，∗input_vrid，∗input_addr，∗input_size，∗output_vmid，∗output_vrid，∗output_addr，∗output_size，machine_num)	io_op=0x01 の場合，ioid で指定した入出力に対して入力領域と出力領域の仮想空間識別子 vmid，仮想領域識別子 vrid，仮想アドレス addr を登録する．io_op=0x10 の場合，ioid で指定した入出力に対して入出力相手装置番号 machine_num と buff に格納したアドレス情報を対応付ける．

(1) データ送信処理
  (A) データ送信プロセスは，カーネルに対してデータの出力を依頼する．
  (B) カーネルは，データ送信用 mbuf のデータ格納領域へのポインタを繋ぎ変えて出力領域の実アドレスを登録し，パケットヘッダを生成する．出力の個数分のデータに送信処理を終えた後に NIC に対してパケット送信処理を依頼する．
  (C) NIC は，mbuf のデータ格納領域へのポインタを参照する．これにより，ユーザ用仮想空間上に存在するデータをカーネル用仮想空間に複写することなく直接 NIC へと DMA 転送を行うことができる．その後，送信パケットを生成して通信路上に送信する．
(2) データ受信処理
  (A) データ受信プロセスは，カーネルに対してデータの入力を依頼する．
  (B) カーネルは，受信パケットが未だ到着していない場合は受信プロセスを休眠させ，パケット受信待ち状態にする．
  (C) NIC は，パケットを受信するとデータ受信用 mbuf に受信パケットを DMA 転送し，受信割り込みを発生させる．
  (D) カーネルは，パケット受信処理を開始する．この時に受信割り込み発生を禁止し，以降はポーリングによる受信パケット検出を行う．パケット受信処理としてパケットヘッダの解析を行い，受信データ格納先の入力領域と mbuf のデータ格納領域に対して実メモリ交換によるデータ授受を行う．入力の個数分のパケットを受信すると受信割り込みを許可し，受信プロセスを起床させる．

上記のようにデータ送受信処理を行うことで送受信処理ともにゼロコピー通信を実現することができる．

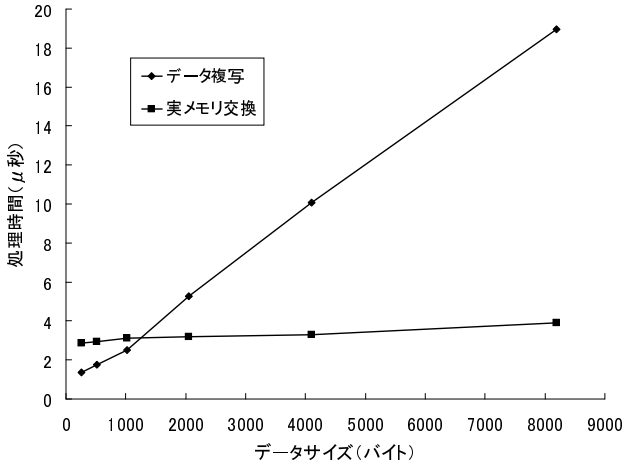

図 3 実メモリ交換とデータ複写の処理時間比較

4.5 評　　価

4.5.1 評 価 環 境

実メモリ交換機能を ***Tender*** に実現し，Linux でのデータ送受信処理と比較する．Linux は Fedora 10（2.6.30 カーネル）であり，Packet Socket を用いる．測定は，計算機 2 台を直結して行った．

4.5.2 データ複写との比較

実メモリ交換とデータ複写でのメモリ間データ授受に要するプロセッサ処理時間を ***Tender*** で測定した．測定結果を**図 3** に示す．

(1) データサイズが小さい場合，実メモリ交換よりもデータ複写の方が処理時間が短い．

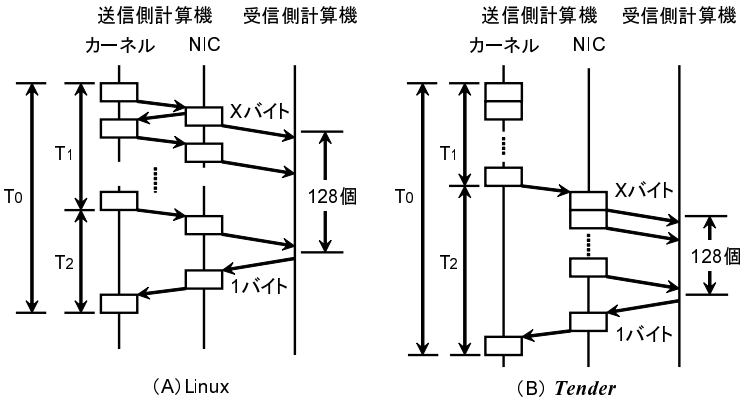

図 4 測定処理の流れ

これは，実メモリ交換処理のオーバヘッドがデータ複写処理のオーバヘッドを上回るためである．

(2) データ複写はデータサイズに比例して処理時間が増加するが，実メモリ交換は，データサイズが 4KB 以下の場合に処理時間の変化はない．これは，実メモリ交換の処理時間はページ数に依存するためであり，ページ数が一定であればデータサイズに変更が生じても処理時間への影響はない．

(3) データサイズが大きくなるほど実メモリ交換とデータ複写の処理時間差は大きくなり，データサイズが 4KB の場合には約 2.8 倍，8KB の場合には約 4.8 倍の処理時間差となる．

以上のことから，実メモリ交換は大容量データ授受に対して大きな効果を発揮することが分かる．

4.5.3 スループット

スループット測定処理の流れを**図 4** に示す．送信側計算機から 128 個のパケットを受信側計算機に対して連続で送信し，受信側計算機では全ての受信パケットを処理した後に 1 バイトのデータを持つパケットを送信側計算機に対して返信する．この処理に要する処理時間を測定してスループットを算出した．ここで，T_0 は送信側計算機がデータ送信処理を開始した時間から受信側計算機からの返信パケットの受信処理を完了するまでの時間，T_1 は，送信側計算機がデータ送信処理に要するプロセッサ処理時間，T_2 は，送信側計算機がデータ送信処理中のプロセッサ処理を終了した時間から返信パケットの受信処理を完了するまでの時間である．

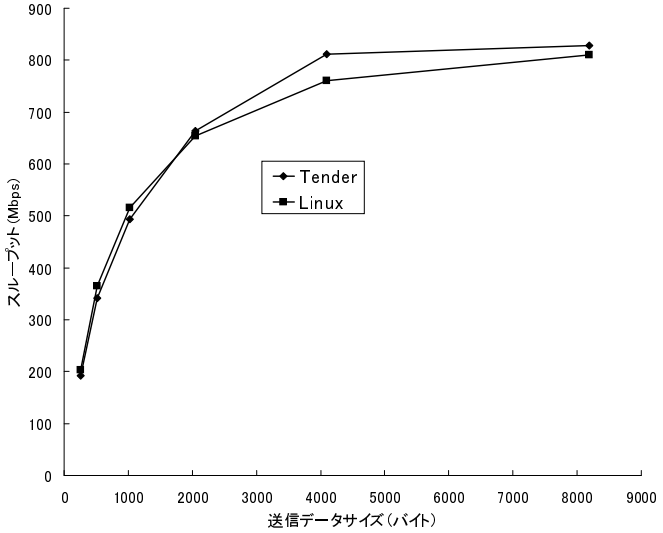

図 5 スループット

T_0 から算出したスループットを**図 5** に示し，T_1 と T_2 の時間を**図 6** に示す．

(1) データサイズが小さい場合，***Tender*** より Linux の方がスループットが高い．これは，パケット受信時の実メモリ交換処理のオーバヘッドがデータ複写処理のオーバヘッドを上回るためである．

(2) ***Tender*** と Linux の両者ともスループットの上昇は 800Mbps 前後で停止しており．通信路の提供するスループットの 8 割程度しか性能を発揮できていない．これは，T_0 に返信パケットの送受信処理に要する時間を含んでいるためであり，一方向の転送能力はさらに高いと考えられる．

(3) ***Tender*** の T_1 はデータサイズを増加させても変化がなく，Linux と比較して非常に短い（データサイズが 8KB の場合には Linux の約 3.4%程度）．これは，データ送信時にゼロコピー通信を実現していること，および送信対象のデータ全てに対してカーネルの送信処理を終えてから NIC にパケット送信処理を依頼しているためである．このようにデータ送信処理を行う場合，カーネルの送信処理と NIC の送信処理

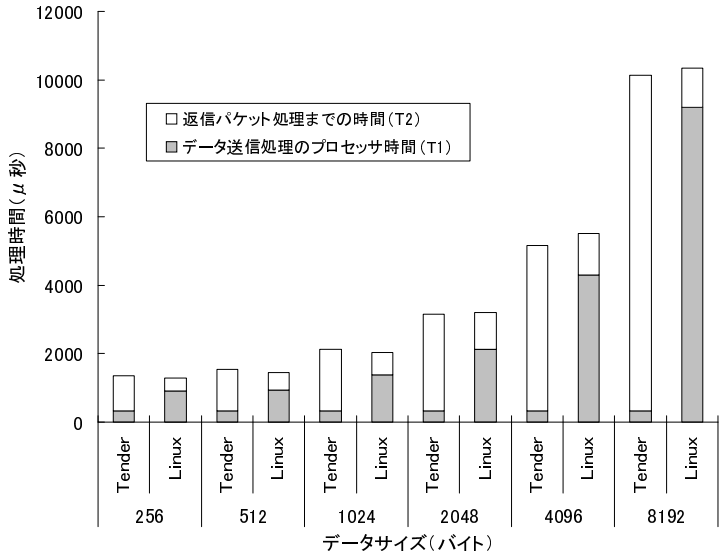

図 6 処理時間（スループット）

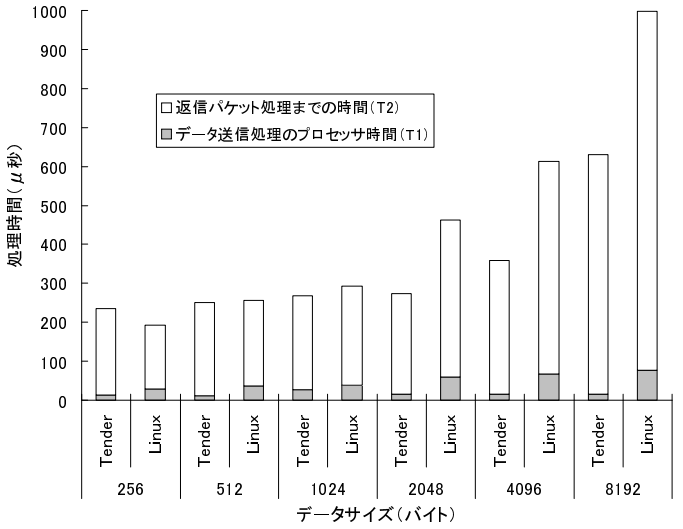

図 7 処理時間（ラウンドトリップタイム）

を並列に行えないという欠点がある．しかし，プロセッサ処理を早期に終了させることでプロセッサ負荷を大きく軽減できる点は大きな利点である．

4.5.4 ラウンドトリップタイム

図 4 での送信パケットを 1 個とし，受信側計算機からの返信パケットのデータサイズを送信パケットと同じサイズにした場合の処理時間（ラウンドトリップタイム：RTT）を測定した．T_1 と T_2 の時間を**図 7** に示す．

(1) データサイズが小さい場合，***Tender*** より Linux の方が T_0 が短い．これは，パケット受信時の実メモリ交換処理のオーバヘッドがデータ複写処理のオーバヘッドを上回るためである．

(2) Linux はデータサイズが大きくなるにつれて T_1 が増加するのに対し，***Tender*** はデータサイズを増加させても変化がない．これは，***Tender*** はデータ送信時にゼロコピー通信を実現しており，データサイズの影響を受けないためである．

(3) データサイズが大きい場合，***Tender*** の方が Linux よりも T_0 が短くなり，データサイズが大きくなるほど ***Tender*** と Linux の処理時間差は大きくなる．これは，Linux ではデータ複写に要する処理時間が増大するためである．データサイズが 8KB の場合には Linux と比較して T_0 を約 63%程度削減できており，計算機間で大容量データに対して逐次応答処理を実行する場合に非常に大きな効果を発揮することが期待できる．

5. おわりに

仮想メモリ空間上の 2 つの領域に対して対応付ける実メモリを交換することでデータ授受を実現する実メモリ交換機能を提案した．実メモリ交換機能を実現することにより，複写レスでのメモリ間データ授受を行うことが可能になる．***Tender*** における資源の分離と独立化を利用して実メモリ交換機能を実現することでデータ送信時のみでなくデータ受信時もゼロコピー通信とすることを可能にした．

データ複写と実メモリ交換の処理時間の評価により，データサイズが 8KB の時にはデータ複写処理よりも約 4.8 倍高速になることを示し，大容量データ授受に対して非常に大きな効果があることを示した．また，Linux とのラウンドトリップタイムの評価により，処理時間を約 37%削減した．しかし，データサイズが小さい場合には，実メモリ交換のオーバ

Vol.2010-OS-113 No.8
2010/1/28

ヘッドがデータ複写のオーバヘッドを上回るため，データサイズに応じてデータ複写と実メモリ交換を使い分けるといった対処を行う必要がある．

残された課題として，送受信対象データのサイズに応じて実メモリ交換とデータ複写を使い分けるといった対処を行い，データサイズに適した送受信方式を実現することがある．

参 考 文 献

1) Feng，W.，Balaji，P.，Baron，C.，Bhuyan，L.N. and Panda，D.K.：Performance characterization of a 10-Gigabit Ethernet TOE，Proc. 13th Symposium on High Performance Interconnects，pp.58-63，(2005)．
2) Menon，A. and Zwaenepoel，W.：Optimizing TCP Receive Performance，Proc. USENIX 2008 Annual Technical Conference on Annual Technical Conference，pp.85-98，(2008)
3) 住元真司，堀　敦史，手塚宏史，原田　浩，高橋俊行，石川　裕，：既存 OS の枠組を用いたクラスタシステム向け高速通信機構の提案，情報処理学会論文誌，Vol.41，No.6，pp.1688-1696，(2000)．
4) 住元真司，堀　敦史，手塚宏史，原田　浩，高橋俊行，石川　裕，：高速通信機構 PM2 の設計と評価，情報処理学会論文誌，Vol.41，No.SIG 5(HPS 1)，pp.80-90，(2000)．
5) 住元真司, 大江和一, 久門耕一, 朴　泰祐, 佐藤三久, 宇川　彰，：複数 Gigabit Ethernet を用いた PACS-CS のための高性能通信機構の設計と評価，情報処理学会論文誌コンピューティングシステム，Vol.49，No.SIG 12（ACS 15），pp.25-34，(2006)．
6) Goglin，B.：Design and Implementation of Open-MX: High-Performance Message Passing over generic Ethernet hardware，Workshop on Communication Architecture for Clusters, held in conjunction with IPDPS 2008，(2008)．
7) 谷口秀夫，青木義則，後藤真孝，村上大介，田端利宏，：資源の独立化機構による ***Tender*** オペレーティングシステム，情報処理学会論文誌，Vol.41，No.12，pp.3363-3374 (2000)
8) Pinkerton，J.：The Case for RDMA，RDMA Consortium（online），available from ⟨http://www.rdmaconsortium.org/home/⟩.
9) 門　直史，田端利宏，谷口秀夫，：***Tender*** における資源「入出力」を用いた Ethernet 通信の設計，電子情報通信学会 2008 年総合大会講演論文集，Vol.2008, pp.94，(2008)