社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2004－OS－97 (12)
2004／8／1

10GbE環境でのワイヤスピードストレージシステムの実装と評価

渡辺 高志† 大江 和一† 西川 克彦†

† 株式会社富士通研究所 〒211-8588 川崎市中原区上小田中 4-1-1(研 35)
E-mail: †{wtnb,ooe.kazuichi,west}@jp.fujitsu.com

概要 ネットワーク上に分散させたメモリをキャッシュとして用いるワイヤスピードストレージシステムを、InfiniBand の HCA(Host Channel Adapter) のファームウェアを変更した RDMA 対応の 10 Gigabit Ethernet の通信システムを利用して動作させ、ブロックアクセスで最大 220MB/s のスループットを達成した。

キーワード 10GbE, InfiniBand, PM2, RMA, RDMA, ストレージ, SCSI, SRP

Implementation and Evaluation of the "Wire Speed Storage System" using 10-Gigabit Ethernet

Takashi WATANABE†, Kazuichi OE†, and Katsuhiko NISHIKAWA†

† FUJITSU Laboratories Ltd. kamikodanaka 4-1-1, Nakahara-ku, Kawasaki-shi, Kanagawa 211-8588, Japan
E-mail: †{wtnb,ooe.kazuichi,west}@jp.fujitsu.com

Abstract The "Wire Speed Storage System"(WSS) is a storage system which uses distributed memory in a network as disk cache. We implemented the WSS on the RDMA-enabled 10-gigabit Ethernet Network which is realized by the InifiniBand HCA(Host Channel Adapter) with customized firmware. This system achieved 220MB/s throughput with block I/O.

Key words 10GbE, InfiniBand, PM2, RMA, RDMA, Storage, SCSI, SRP

1. はじめに

ネットワークの帯域やレイテンシといった性能は向上しているが、ストレージの性能向上はそれに比べると鈍い。特にディスクの構造上、レイテンシに関しては平均シーク時間が数 ms のままで、劇的に短かくなることは期待できない。そのため、アクセスパターンを充分長いサイズのシーケンシャルアクセスにしなければ拡大するネットワーク帯域を使い切ることはできない。

そこで我々はこれまでネットワーク上のメモリにディスクキャッシュを分散させ、データを RDMA(RMA) で転送することで性能を高めるワイヤスピードストレージ (WSS) というシステムを試作してきた。約 100MB/s、RTT40us の emulex cLAN や、約 800MB/s、RTT40us の DAPL(注1)/InfiniBand(IB) で評価し、cLAN では期待通りの性能を達成したが、IB ではメモリ登録のレンテンシの問題などの問題点が見つかった。

以前に見つかった問題点 (詳細は 4.4 節参照) を踏まえ、10-gigabit Ethernet(10GbE) のネットワークに WSS を移植し、性能を測定した。

(注1): Direct Access Programming Library

2. WSS とは何か？

WSS は、ネットワークに分散したメモリをディスクキャッシュとして利用してキャッシュ容量を増やし、ヒット率を上げることによりディスク I/O のボトルネックを解消するアーキテクチャである。ネットワーク上にストレージエンジン (SE)、キャッシュエンジン (CE)、実ストレージ (RS) があり、同じネットワーク上にクライアントが接続されている (図 1)。

WSS では、クライアントからのリクエストは SE に送り、RDMA を用いてデータを保持している CE や RS からクライアントに直接 (リクエストを受信した SE を経由することなく)

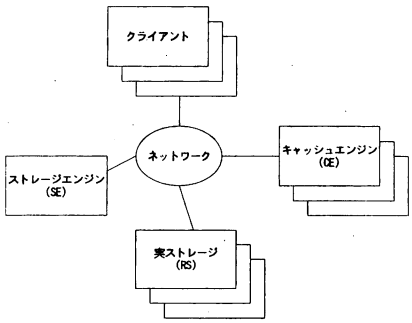

図 1 WSS のアーキテクチャ

データを転送するアウトバンド方式を採用している。SE がストレージの仮想化とキャッシュのマップの管理を行い、クライアントからのリクエストを内部プロトコルに変換して CE や RS に転送する。CE や RS は SE に指示された通りにクライアントのバッファ領域に直接 RDMA でデータを転送し、転送完了後にリプライを返す (図 2)。

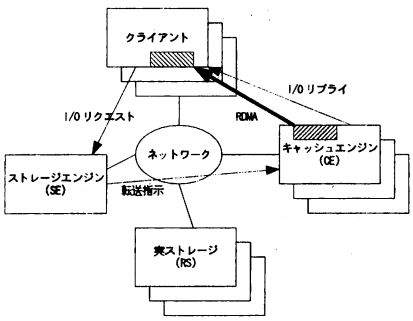

図 2 WSS のメッセージの流れ

2.1 これまでの研究経緯

これまで我々は cLAN(通信 API は VIPL[注2]) や IB(通信 API は DAPL) を利用して WSS を実装し、性能を測定してきた (表 1)。

cLAN はスループット 1Gbps、RTT40us のネットワークで、RDMA Write が利用できる。RDMA Read がサポートされていないため、メッセージ通信と RDMA Write を組み合わせてエミュレーションを行なった。コピーの影響を排除した場合でネットワーク帯域をほぼ使い切り、SCSI ドライバを経由した Raw I/O アクセスで 60MB/s のスループットを出すことができた [4]~[6]。

IB はスループット 10Gbps、RTT 約 40us のネットワークであるが、4.4 節に述べる問題により、スループットは 100MB/s にとどまった [7]。

表 1 性能比較

発表時期	ネットワーク ブロックアクセス性能
2002 年 [4], [5]	VIPL/cLAN 60MB/s
2004 年 [7]	DAPL/IB 100MB/s

3. 実験環境

3.1 利用した機器

利用した PC 環境は表 2 の通りである。次の 3.2 節に示す通り、ネットワークの API に SCore [1] で利用されている PM2 を採用したこともあり、OS に SCore 5.6.1 を利用している。

表 2 PC サーバ環境

CPU	Pentium4 Xeon 2GHz x2
	PCI-X 133MHz
メモリ	3GB(bcopy 性能：実測 550MB/s)
OS	SCore 5.6.1(Linux-2.4.21)

3.2 10GbE のネットワーク環境と RDMA

10GbE は通信速度が 10Gbps の Ethernet 規格[注3]であり、今後普及が見込まれる高速ネットワークである。

一方で現在の一般的な PC サーバには DDR SDRAM のメモリが搭載されており、ノード内メモリコピーの性能は約 550MB/s である。10GbE を使うと、ネットワークの最大スループットがノード内のメモリコピーよりも高速になる。そのため、ノード内メモリコピーの削減のため、RDMA を使うことが有効であると考えられる。

そこで、10GbE で RDMA が使える環境として、富士通研究所で試作中の、IB HCA を使って 10GbE ネットワークを実現するシステムを使った (図 3)。富士通製 IB HCA のファームウェアを Ethernet 用に書き換えたものと、IB のパケットを 10GbE のパケットに変換するブリッジからなっている。ホストから通常の Ethernet のドライバとしては使えないが、出てくるパケットが Ethernet フレームになっており、Ethernet 用のスイッチで接続することができる。また、ファームウェアの機能で RDMA を利用できる。

このネットワークで動く SCore [1] の通信ライブラリ PM2 を使って、WSS を動作させた。

ただし、本論文執筆までにブリッジが入手できなかったため、ブリッジ及びスイッチを介さずノード同士を直結 (図 4) しての測定となった。

(注2) : Virtual Interface Provider Library

(注3) : IEEE 802.3ae

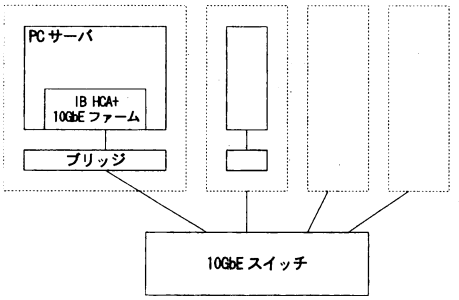

図 3　ネットワーク環境

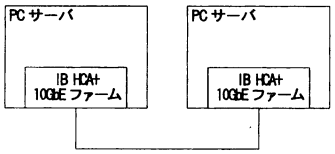

図 4　ネットワーク環境 (直結)

4. 実装内容

直結で 2 ノードまでの環境しか用意できなかったため、図 5 のように SE と CE を同じノード、同じプロセスで動作させた。キャッシュヒット時の性能を測定するため、RS は省略し、ネットワーク経由の RAM ディスクのような構成となっている。

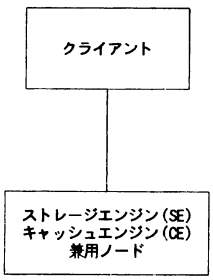

図 5　今回の WSS の構成

4.1 SRP

RDMA を使うストレージ処理のプロトコルには SRP [2] を使った。以前に VIPL や DAPL での評価のために実装した SRP ドライバの通信部分を PM2 に対応させた。

クライアントはカーネルレベルの SCSI ドライバであるが、ユーザレベルのプログラムとして SCSI コマンドを処理してサーバに送信し、性能を測定することもできるようにもなっている。ユーザ/カーネルはコンパイル時のオプションにより切り替わる。

4.2 Kernel PM2

クライアントはカーネルで SCSI インタフェースのドライバとなるが、PM2 をカーネルから利用するインタフェースは無いため、PM2 のカーネル版 API(kPM2) を作った。

kPM2 はほとんどの部分で PM2 と互換性を持つが、カーネル内で利用することを想定しているため、以下のような変更点がある。

（1） 初期化の際、PM2 はノード名等が記述された設定ファイル名を argc/argv の形式で与えるが、kPM2 ではドライバモジュールをロードした後に設定コマンドでノード名の設定を行う。

（2） PM2 の受信や完了待ちはポーリングが基本だが、カーネル内でポーリングすることを避けるため、割り込みを使ってコールバックで通知する。

4.3 全体のモジュール構成

クライアント (SCSI 用語ではイニシエータ)、SE(同じくターゲット)、通信ライブラリ PM2、IB-HCA の関係は、図 6 のようになっている。ターゲットや CE、RS はユーザレベルのプログラムであり、PM2 を利用する。一方でカーネルレベルで動作するイニシエータは kPM2 を使って通信を行い、ブロックデバイス層、SCSI 共通層を経由してユーザの I/O 要求を受けつける。

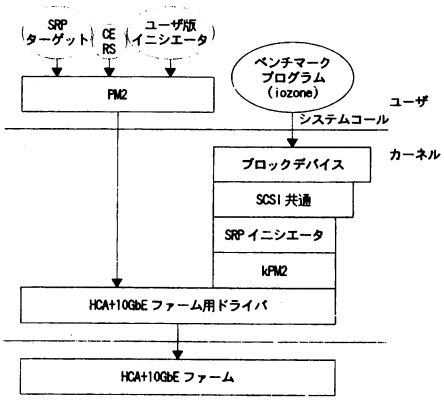

図 6　構　　成

4.4 既知の問題と解決方法

これまでの研究で、次の問題が明らかになっている。

（1） Linux の共通 SCSI ドライバのバッファ構成は、1 つ 4〜128KB の I/O コマンドに 4KB 程度の離散したバッファが連なっているという特徴がある [5]。この特徴により、1 回あたりの mlock/munlock のコスト、4KB の scatter/gather の速度が性能に影響を与えることなどが判明している [7]。

これはイニシエータ側のノードで mlock/munlock をバッファ

の個数回実行し、ターゲット側のノードがRDMAをバッファの個数回実行するためである。回数が多くなることによって性能が遅くなるならば、1つのバッファにまとめてイニシエータのノード内で離散バッファにコピーすることで回数を減らす方法も考えられるが、それではRDMAを使う意味がなくなってしまう。

（2） 今回使ったノードは実装メモリが2GBであるが、最近は4GB以上のメモリを搭載したPCサーバも増えている。4GBより大きなメモリを搭載したマシンの場合、仮想アドレスを用いて登録する方式では、狭い仮想メモリ内の領域しかバッファキャッシュに使えなくなってしまう。Linuxのバッファキャッシュ機構はもともと仮想メモリ外のアドレスも管理できるようになっているが、これを使うためには、HighMemoryと呼ばれるメモリ領域にDMAでアクセスする必要がある。このとき、HighMemory領域は仮想アドレスでは指定できない。

上記2つの問題を解決するため、複数に離散したHighMemory領域を一度にカードに登録/解除し、リモートからは1回のRDMAで書き込むための機能をファームウェアに加え、kPM2からその機能を使うAPIをkpmMLockV/kpmMUnlockVとした。

Linuxのカーネルでscatter/gatherの用途に使われているscatterlistがHighMemoryの指定に対応しているため、scatterlist構造体を使ってメモリ領域を受け渡しすることにした。

I/O要求のバッファはSCSI共通層からscatterlist構造体の配列として渡されるが、scatterlist構造体の中には仮想アドレスが入っており、仮想アドレスがNULLの場合はpage構造体とオフセット値を使う。HighMemoryを使う場合は仮想アドレスはNULLに設定され、page構造体とオフセット値を使って実アドレスを算出することができる。1つのscatterlistには1ページ(4KB)未満のバッファしか入らない(図7)。

```
struct scatterlist
{
    char *address;          // 仮想アドレス
    struct page *page;      // address==NULL の場合
    unsigned int offset;    // page 内オフセット
    dma_addr_t dma_address; // 実アドレス
    unsigned int length;
};
```

これらを考慮し、離散領域の登録APIを次のようにした。

```
int kpmMLockV(kpmContext *ctxt, int rmt_node,
              int sgnum, struct scatterlist sg[],
              kpmAddrHandle *hndl_out);
int kpmMUnlockV(kpmContext *ctxt, kpmAddrHandle hndl);
```

MLockVで登録した領域は、実際には離散したメモリ領域であるにも関わらず、リモートからのpmRead()/pmWrite()では連続した領域に見えるようになる。

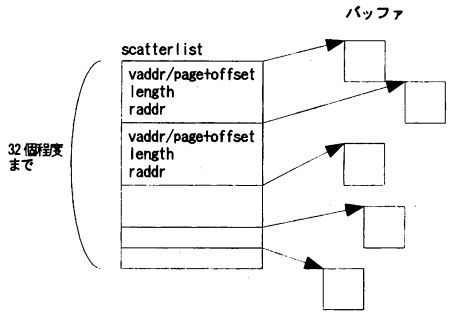

図7 scatterlist構造体

5. 性能評価

5.1 ネットワーク性能

ネットワークの基礎性能を測定した結果を以下に示す。

5.1.1 send/recv

pmSend()とpmReceive()を使って、pingpong通信のレイテンシを測定した(図8)。8バイトのメッセージでは、往復約28usで通信が行なわれる。WSSではメッセージ通信には50～100バイト程度のメッセージが多く使われているが、これらはおおむね片道15us程度で到達するであろうと言える。

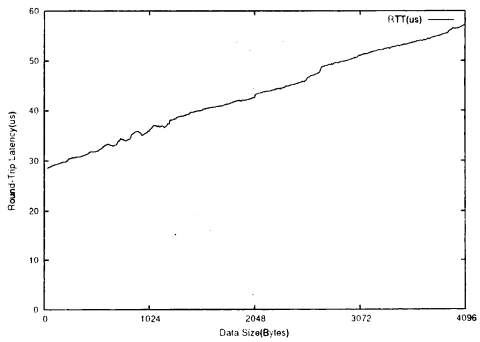

図8 send/recvのレイテンシ

5.1.2 RDMAの性能

pmWrite()～pmIsWriteDone()、pmRead()～pmIsReadDone()までの性能をサイズを変えながら測定した。1回のRDMAでのレイテンシを計測した結果を図9に、レイテンシの情報からスループットを計算したものを図10に示す。

次に、MLock/MUnlockのレイテンシを計測すると、MUnlockは25～30usに収まっているのに対し、MLockに

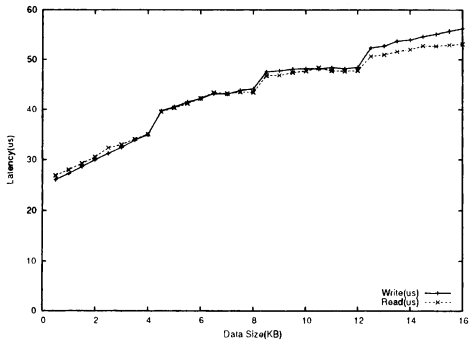

図 9　RDMA のレイテンシ

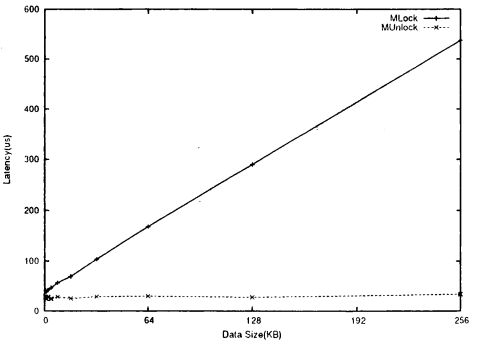

図 11　MLock/MUnlock のレイテンシ

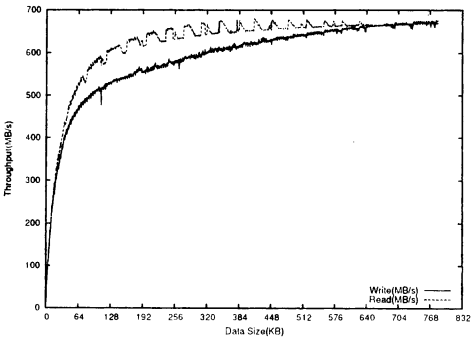

図 10　RDMA のスループット

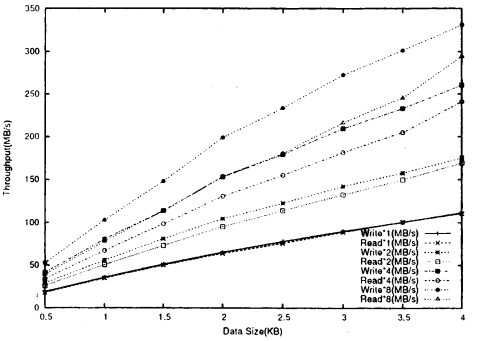

図 12　連続 RDMA のスループット

ついてはデータサイズに比例してレイテンシが増加していた (図 11)。MLock 自体のスループットは 490MB/s 程度であった。従って RDMA ごとに毎回 MLock/MUnlock を呼び出すと、あまり性能が出ないことになる。

WSS では初期化時に CE のキャッシュ領域の全てに MLock を行なっておくため、ストレージ側ではこの部分は性能上の問題はないはずであるが、クライアント側のバッファは上位のブロックデバイスの共通層が管理している領域であり、あらかじめ Mlock で登録を済ませておくことができず、性能上の問題になりうる。ただし、カーネル内では MLock ではなく MLockV を利用し、MLockV の性能の特性は MLock とは異なる (5.1.3 節を参照)。

MLockV を使わない場合、Linux の SCSI ドライバによるアクセスで使う典型的なパターンは、4KB 未満の小さなサイズの RDMA を連続して実行するものである。そこで、1～4KB サイズの RDMA を連続して 1～8 回連続して送る部分の性能を計測した (図 12)。ブロックアクセスでよく使う部分は性能が 250～350MB/s 程度であり、このままでは本来出したいスループットを出せないことがわかる。

5.1.3　kpmMLockV/kpmMUnlockV の効果

kpmMLockV を使った 512Bytes～4KB の離散バッファへのアクセスを測定した。図 13 の横軸が離散したバッファ数、縦軸がスループットである。kpmMLockV を使った場合、Linux の SCSI ドライバによるアクセスで使う典型的なパターンは、128KB(4KB 単位でバッファ数 32) のアクセスである。Raw I/O では 64KB(512Bytes 単位でバッファ数 128) の離散バッファとなる。

グラフからもわかるように、MLockV を使わない場合の 250～350MB/s から、570～600MB/s まで性能が向上しており、連続バッファの RDMA 転送に比べても遜色ない性能が出ている。従って、MLockV はスループットの向上に貢献すると推測できる。

また、MLockV 自体の処理時間はバッファ数の影響をあまり受けず、バッファ数が 1 個でも 32 個でも、約 20～30us 程度だった。ユーザの MLock よりも高速なのは、カーネル内ですでに実メモリに固定されたバッファを一度に HCA に登録する

ため、ユーザメモリを登録する処理と比べて必要な操作が少ないためだと考えられる。

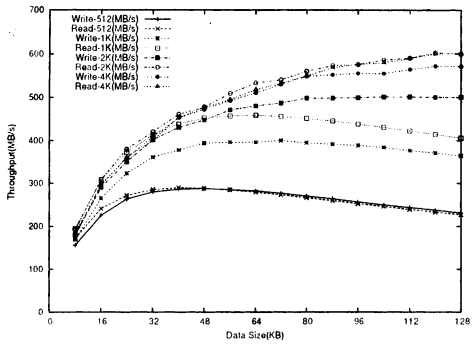

図 13　MLockV の効果

5.2　ユーザレベルでの SCSI 性能

ユーザ版イニシエータで、SCSI コマンドを発行するテストプログラムを用いて 1 つの SCSI I/O コマンドが戻ってくるまでの時刻を計測した (図 14)。512 バイト (1 セクタ) のアクセスを約 180us で処理している。

5.1.2 節の結果より、512 バイトの RDMA の処理が 26us、5.1.1 節より、送受信が往復で 30us、MLock に 40us、MUnlock に 30us かかるため、ネットワークの処理には約 126us 程度が必要とされている。残りの約 54us、約 30%が WSS のオーバーヘッドとなっている。

データサイズが大きくなっても性能があまり伸びないが、これはユーザレベルの MLock の処理時間が増大するためである。例えば 64KB アクセスのレイテンシは約 300us だが、RDMA 自体は 120us 程度で完了するものの、MLock に 170us もかかっている。この問題はユーザレベルではピンダウンキャッシュでバッファ登録の情報を再利用すれば改善できる可能性がある(注4)。また、カーネルレベルでは高速に複数の領域を登録できる MLockV を使っても解決できるだろう。

5.3　システムの性能

SCSI ドライバとして組み込んだときの性能を計測した。ベンチマークソフトには iozone [3] を利用し、以下の 2 通りのアクセス方法で計測した。

- Raw I/O
- Block I/O

Raw I/O はバッファキャッシュの影響を排除できるが、Read/Write 毎に SCSI の Read/Write コマンドが発行されるため、SCSI コマンドの多重度を上げられなくなるという問題がある。

(注4)：本稿執筆時点でピンダウンキャッシュには未対応

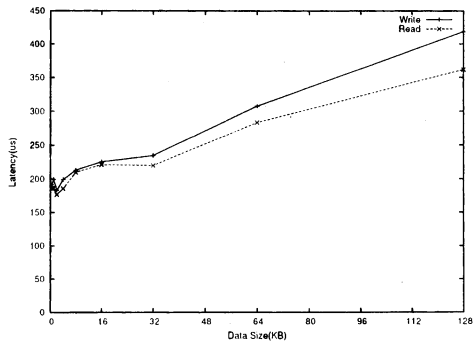

図 14　ユーザレベルの性能 (レイテンシ)

Block I/O はディスクアクセスに用いられる通常のアクセス方法であるが、バッファキャッシュの影響を受け、キャッシュにヒットした場合に I/O が出ないなど、SCSI デバイス自体の性能が計測できないことがある。

どちらの方法でも、カーネル-ユーザ間でメモリコピーが 1 回発生する。RDMA でカーネル内のバッファメモリに書き込まれた後、CPU がユーザプロセス側のメモリにコピーを行う。

Raw I/O で計測した結果が図 15 である。70MB/s 程度にとどまっており、ユーザレベルで測定した場合と比べて遅くなっている。

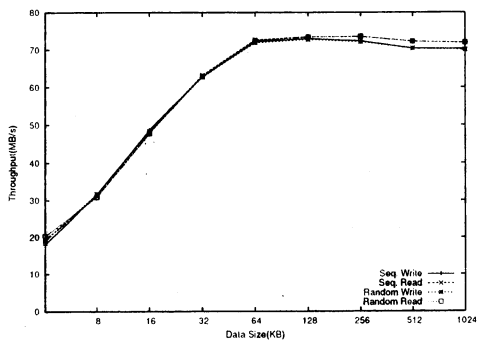

図 15　raw I/O の性能

Block I/O で計測した結果が図 16 である。Read が約 220MB/s、Write が約 210MB/s となった。バッファキャッシュの影響については、Write に関しては、iozone の機能 (-e オプション) で sync が完了するまでの時間を基準にスループットを計測することで避けた。Read に関しては、ターゲット側の統計情報を見る限り、Read を行なった総セクタ数と Write を行なった総セクタ数が 99.0%の精度で一致していることから、期待した Read アクセスは充分発生しており、バッファキャッシュ

の影響は排除できていると判断した。

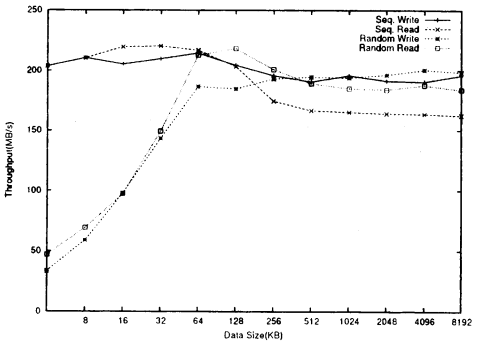

図 16 ブロックアクセスの性能

6. 考　　察

ネットワークの単体性能はメッセージ通信のレイテンシが往復 25us、RDMA のスループットが 670MB/s(離散したバッファが対象であっても 600MB/s) と、ほぼ問題ないと言える性能が出ている。

ネットワーク帯域 (600MB/s)+メモリコピー 1 回 (550MB/s) という動作の理論上の最大値は 1/(1/600 + 1/550) =287MB/s であり、その程度の性能が期待される。

ところが、Raw I/O が 70MB/s、Block I/O のスループットも最大 220MB/s 程度に留まるなど、期待値までは性能が出ないという結果になった。

6.1 Raw I/O の詳細

5.3 節で述べたように、Raw I/O では 70MB/s しか出なかった。strace コマンドによるシステムコール発行のタイムスタンプと WSS の内部トレース情報を分析し、64KB の Write 時の処理時間の内訳を調べた。

表 3 は、クライアント内でユーザプロセス (iozone) が write システムコールを発行してから pmSend が呼ばれるまでの時間と、SE/CE で SRP のメッセージを受信してから pmRead、pmSend が呼ばれるまでの時間、クライアントで SRP のリプライメッセージを受信してから write システムコールが終了するまでの時間を示している。strace の出力結果を見ると、Write システムコールの完了までに約 920us を要しているようだが、この値は表 3 のそれぞれの処理時間の総和に SRP のメッセージ通信時間 (50~100 バイトの往復) の 30us を加えたものとほぼ一致する。また、64KB の転送に 920us という値は、スループットの 70MB/s ともほぼ一致する。

表 4 は、表 3 を処理の種類ごとにまとめたものである。「メモリコピー」の部分はコピー性能とユーザ/カーネル間のコピーが 1 回必要になるという動作から推測した値である。

表 3 64KB Write の処理時間の内訳

クライアント送信		SE/CE		クライアント受信	
Write		pmRecv		pmRecv	
⇓	374	⇓	11	⇓	2
ドライバ呼出		ストレージ処理	21	pmMUnlockV	19
⇓	16	その他内部処理	12	⇓	2
pmMLockV	26	⇓	19	SCSI 完了通知	
⇓	1	pmRead 起動	4	⇓	146
pmSend		⇓	230	Write 完了	
		RDMA 完了			
		⇓	8		
		pmSend			
合計	417		305		169

(単位:us)

最も割合が大きいのが「その他」の部分で、半分近くになっている。この部分の処理時間を削減することが性能向上につながるが、カーネルの他の部分を改造しなければ改善されない。

次に割合が大きいのは通信関連である。これは MLockV、Send-Recv、RDMA、Send-Recv、MUnlockV の和である。RDMA が 230us と最も大きく通信関連の 75%、送受信が往復 30us で通信の 10%、MLockV と MUnlockV が合わせて 15%となっている。削減できるとすると MLockV と MUnlockV だが、処理時間の比率は小さく、削減しても効果はほとんどないと思われる。RDMA の 230us は OS から与えられるバッファ構成が 512 バイト単位であるためで、これを例えば Block I/O のように 4KB 単位の離散バッファに改善すれば半分の 120us 程度にまでは短縮できるはずである。

次に割合が大きいのはメモリコピーだが、ユーザ/カーネル間のコピーが発生するのは避けられない上、全体の 12%とあまり大きな比率ではない。しかし、Raw I/O ではカーネル内にバッファキャッシュを持たず、Read/Write が呼び出されてから I/O が終わるまではユーザ側に制御が移らないため、ユーザの与えたバッファを MLock で固定し、ユーザのバッファに直接 RDMA でアクセスしてコピーを発生させずに済ませる方法もあり得る。ただし、ストレージデバイスへの Raw I/O はそれほど広くは利用されておらず、OS がそのような最適化をする可能性は低いように思われる。

最後に WSS と SRP の処理時間は全体の 10%に収まっており、現状ではボトルネックにはなっていないものと思われる。

表 4 64KB Write の処理時間の内訳 (種類ごと)

種類	時間 (us)	割合 (%)
通信関連	309	33.6
WSS と SRP	92	10.0
メモリコピー	114	12.4
その他	405	44.0
合計	920	100.0

6.2 Block I/O の詳細

Block I/O と Raw I/O の動作の主な違いはユーザプログラムからのアクセスがバッファキャッシュを介することだが、デバイスドライバ側から見ると、以下の 2 つの違いがある。

- バッファが 4KB 単位で離散している (Raw I/O は 512 バイト単位)。
- 前のコマンドの終了を待たずに多重に I/O コマンドがキューに渡される。

まず、5.1.3 節の図 13 からもわかるように、512 バイト単位よりも 4KB 単位で離散しているほうが性能は高いため、バッファ構成の違いが大きく性能に影響すると考えられる。この 4KB というのは OS のページサイズで、OS がバッファキャッシュを割り当てるときに空きページを割り当てている。バッファキャッシュで複数の連続したページを割り当ててまとめて管理したり、あるいは (OS のかなりの部分に影響してしまうが) ページサイズを 4KB よりも大きな単位にすれば、さらに性能が向上するかもしれない。

Block I/O 時の WSS 内部のトレース情報を調べると、個々のコマンドの SRP ドライバ及び SE/CE の処理時間はほぼ同じだったが、WSS 以外の部分の処理時間が短縮されていることがわかった。

Raw I/O ではコマンドの処理が終了してから次のコマンドが渡されるまでに 500us 以上かかっていたが、Block I/O では 65〜70us に短縮されている。これは Raw I/O ではコマンドの終了を待ってユーザプロセスに戻ってからユーザプロセスが次の I/O を要求する、といったようにユーザプロセスまで含めてシリアライズされているのに対し、Block I/O はバッファキャッシュを介しており、ユーザプロセスまで戻らずに次のコマンドを発行できるためではないかと思われる。

上記の理由により、Block I/O が Raw I/O と比べて 2〜3 倍まで高速化されているが、理論上の 287MB/s という値の 76.7%程度であり、性能向上の余地はあると考えられる。

カーネル側のブロックデバイスの性能問題に関しては Linux のカーネル 2.6 系で改善されている可能性もあるため、カーネル 2.6 系で実験することにも意義がある。

7. まとめと今後の課題

RDMA に対応した 10GbE ネットワークに WSS を移植し、ネットワーク性能と WSS の I/O 性能を比較した。

実験では性能は Raw I/O で Read/Write ともに 70MB/s、Block I/O で Write 210MB/s、Read 220MB/s となった。Raw I/O は内部の動作を見る限り、WSS 内部よりは I/O を行なう OS 側の問題が大きく、高速化のためには OS が用意するバッファの構成や I/O 要求の送出間隔を縮める必要がある。Block I/O に関しては RDMA+コピー 1 回という動作の理論上の最大値 287MB/s の 76.7%となった。

また、今回は実験機材の制約により 2 ノードの対向通信でしか実験できなかった。ブリッジを入手次第、3 ノード以上での性能測定を行う予定である。

文　献

[1] SCore クラスタシステムソフトウェア, PC クラスタコンソーシアム, URL: http://www.pccluster.org/

[2] "SCSI RDMA Protocol(SRP) Revision 16a", T-10 Project 1415D, 2002

[3] IOzone Filesystem Benchmark, URL: http://www.iozone.org/

[4] "Wire Speed Storage(WSS) アーキテクチャ", 大江, 渡辺, 西川 (富士通研), SWoPP 湯布院 2002

[5] "ワイヤスピードストレージアーキテクチャ(WSS) における SCSI RDMA Protocol(SRP) の実装と評価", 渡辺, 大江, 西川 (富士通研), SWoPP 湯布院 2002

[6] "Wire Speed Storage アーキテクチャ" 大江, 渡辺, 西川 (富士通研), ACS4, 2004

[7] "10Gbps 環境における Wire Speed Storage (WSS) の性能評価", 大江, 渡辺, 西川 (富士通研), SACSIS2004(ポスター)