Gfarm ファイルシステムにおける RDMA アクセスの設計

建部 修見[1,a)]　佐々木 慎[2]　高橋 一志[3]　大山 恵弘[4]

概要：Gfarm ファイルシステムの遠隔ファイルアクセスをユーザレベルの RDMA を利用して高速化するための設計を行う．RDMA を効率的に用いるためのメモリ登録方式について検討を行い，静的に登録する static 方式と動的に登録する dynamic 方式について比較を行う．IP over InfiniBand (IPoIB) に比べ，書込で 1.7 倍，読込で 2.2 倍の性能向上を達成した．また，並列アクセスにおいては，より少ないクライアントプロセス数でネットワークの物理性能に近い性能を達成した．

1. はじめに

CPU の介在なく，カーネルをバイパスして通信する Remote Direct Memory Access (RDMA)[8] により，ネットワークの物理性能に近い性能を達成することができる．RDMA は, 近年 InfiniBand, RDMA Over Converged Ethernet (RoCE) などの普及，および OpenFabrics Alliance (OFA)[6] によるオープンソースのソフトウェア開発により幅広く使われるようになってきた．

本研究では，ソケットインターフェースで実装されている Gfarm ファイルシステム [14] の遠隔ファイルアクセスを RDMA を用いて高速化するための設計を行う．RDMA を用いたファイルアクセスの高速化については，これまで Lustre[1], GPFS[10] などの並列ファイルシステムの他, NFS over RDMA[2] などの研究開発が行われてきた．ファイルアクセスは遠隔手続き呼出 (RPC) で行われるが, 多くの RPC のメッセージ長は短いため，RDMA の効果はあまりない．そのため，read や write など数 KB を越えるメッセージ長となるデータアクセスについて RDMA が用いられる．

本研究では，これまでの Gfarm ファイルシステムの設計を大幅に変更しない範囲で RDMA アクセスの設計を行う．メモリ登録の方法に関する検討を行い，静的に登録する static 方式と動的に登録する dynamic 方式について実装方式を示す．Gfarm API と POSIX API について，また並列アクセス時のアクセス性能について性能評価を行う．その結果，dynamic 方式でブロックサイズを大きくすると性能が向上すること，POSIX API ではカーネルのページキャッシュが性能に大きく影響することが分かった．また，並列アクセスにより，ネットワークの物理性能に近い性能を達成可能なことも分かった．

1　筑波大学計算科学研究センター
2　日本総合研究所
3　IzumoBASE
4　筑波大学システム情報系
a)　tatebe@cs.tsukuba.ac.jp

2. 関連研究

NFS では，NFS の RPC を RDMA で行う NFS over RDMA[2] が提案されている．Linux ではクライアントは Linux 2.6.24 から，サーバは Linux 2.6.25 から利用可能である．NFS-RDMA は IETF で標準化が進められ，RPC-RDMA プロトコル [12]（ONC RPC のための RDMA トランスポート）と RPC-RDMA への NFS のマッピング [11] が規定されている．ほとんどの RPC におけるメッセージ長は短いため，RDMA read, RDMA write を用いるメリットはなく，RDMA send が用いられる．read, write 等の転送サイズの大きな RPC では RDMA read, RDMA write が用いられる．

Lustre ファイルシステム [1] は，LNet (Lustre Networking) と呼ばれるネットワークインターフェースを持っている．LNet ではソケット，RDMA 等に対応しており，ネットワークレイヤを隠蔽している．

HDFS を RDMA を用いて高速化する研究として [5] がある．この研究では，HDFS の write についてプロトコルを RDMA で書き換え, それ以外は変更を加えていない．write は，ファイル複製を伴い，ネットワーク転送量が多いためである．IP over InfiniBand (IPoIB) との比較で，TestDFSIO, Yahoo! Cloud Serving Benchmark (YCSB)[3] において 15 ∼ 30% の性能向上を示している．

Gfarm ファイルシステムのカーネルドライバ [9] においては，カーネルレベルの RDMA によりページキャッシュに直接転送している．カーネルの仕組みを用いるため，

ⓒ 2017 Information Processing Society of Japan

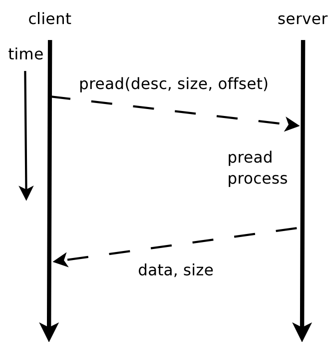

図 1 遠隔手続き呼出.

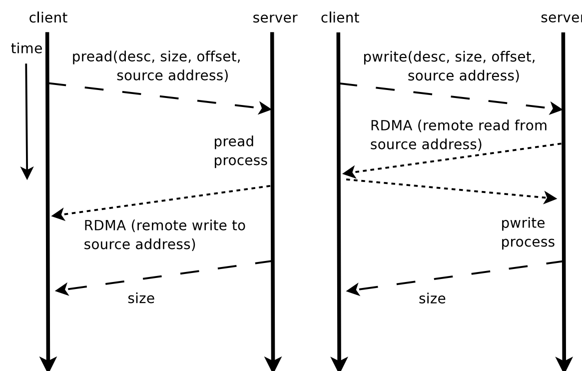

図 2 RDMA を用いてファイルデータの転送を行う遠隔手続き呼出.

readahead 等のカーネルの最適化機構を用いることができる．一方，本研究では，ユーザレベルで RDMA を行い，カーネルをバイパスしユーザバッファあるいはユーザレベルのライブラリのバッファについて RDMA 転送を行う．

3. 設計

3.1 遠隔ファイルアクセス

Gfarm ファイルシステム [14] における遠隔ファイルアクセスは，クライアントとファイルサーバ（gfsd）間の pread, pwrite の RPC で行われる．pread では，ファイルオープン時に取得するファイルディスクリプタとサイズとオフセットを gfsd に送信し，gfsd で該当ファイルのオフセットからサイズ分のデータを読込み，その読込サイズと読込データをクライアントに返す．pwrite では，ファイルディスクリプタ，書込データ，サイズ，オフセットを gfsd に送信し，gfsd で該当ファイルのオフセットからサイズ分のデータを書込み，書込サイズをクライアントに返す．

一方，RDMA は主に RDMA write と RDMA read がある．自ノードのメモリアドレス，遠隔ノードのメモリアドレスを指定することにより，RDMA write では自ノードから遠隔ノードのメモリへのコピー，RDMA read では遠隔ノードから自ノードのメモリへのコピーを行うことできる．RDMA は，CPU の介在無しに行うことができるため，オーバヘッドが小さいが，同期操作と共に用いる必要がある．同期操作を伴う RDMA 転送には RDMA send がある．RDMA send は，あらかじめ登録された受信側のメモリ領域に転送し，受信側でシグナルをあげるものである．あるいは，RDMA send を用いない場合は，RDMA write で同期操作を行うために，受信側で特定のメモリ領域に対する書込をポーリングするなどの手法が取られる．

RPC は，引数等のサーバへの送信，サーバでの処理，返値のクライアントへの送信で構成される．**図 1** に pread の RPC を示す．これらの通信では，送信，受信の同期が必要である．サーバプロセスでは受信をいつ行うか分からないため，常にポーリングを行うか，割り込み等の機構を用いることとなる．

これまで Gfarm ファイルシステムではこの RPC にソケット通信を用いてきた．そのため，この RPC の通信を全て RDMA と同期操作に置き換えるのは大きな設計変更が必要となってしまう．そのため，RDMA は pread, pwrite 等のファイルデータの転送だけに用い，引数や返値の転送はこれまで通りのソケット通信の send, recv を用いる．

図 2 に RDMA でファイルデータの転送を行い，引数，返値はソケット通信で行う RPC を示す．ここで，クライアントは pread の引数として，クライアントのメモリアドレスを送信し，サーバは受信したクライアントのアドレスに対し RDMA write によりファイルデータを遠隔書込みする．遠隔書込後，読込サイズをソケット通信で送信する．図中，破線はソケット通信を表し，点線は RDMA 通信を表している．一方，pwrite の場合は，サーバが RDMA read によりクライアントのデータを遠隔読込みし，ファイルに書出す．このようにすることで，これまでのソケット通信における設計を大きく変更することなく，ファイルデータを RDMA で転送することが可能となる．

引数と返値の送信に RDMA send ではなく，ソケット通信の send, recv を用いるため，RDMA send を用いた通信に比べ RPC の遅延が大きくなることが想定されるが，これは設計変更を少なくすることとのトレードオフとなる．また，RDMA のデータサイズがある程度以上となれば，その遅延は隠れると思われる．

3.2 メモリ登録

RDMA でデータ転送するためには予めメモリ領域をピンダウンしてスワップアウトされないようにし，カーネルに登録する必要がある．さらに，メモリ領域のアドレスを予め遠隔ノードに伝えておく必要がある．

登録されたメモリ領域はピンダウンされ，スワップアウトされないため，実メモリ領域を圧迫する．そのため，多くの領域を登録する訳にはいかない．また，登録するためのオーバヘッドもあるため，なるべく登録した領域を再利用した方がよい．ただし，再利用するためにメモリコピーが必要であれば，そのオーバヘッドも考慮する必要がある．

メモリ登録のオーバヘッドについては，環境によるこ

ⓒ 2017 Information Processing Society of Japan

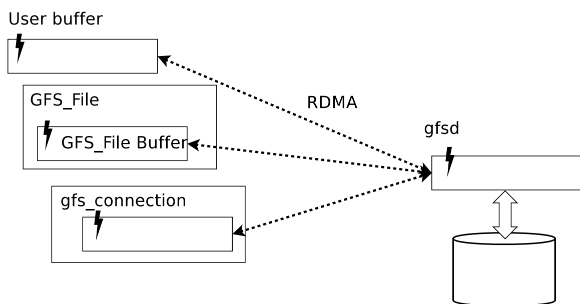

図 3 メモリ登録するバッファと RDMA.

ともあるため，メモリ登録の方式については static 方式と dynamic 方式を検討する．static 方式は，RDMA 転送用のメモリ領域を予め静的に登録する方式であり，dynamic 方式は転送時にメモリ領域を動的に登録する方式である．詳細は実装の章で述べる．

4. 実装

RDMA アクセスの実装にあたり，libibverbs[7] を用いた．libibverbs は，ユーザプロセスが RDMA の低レベルな Verbs とよばれる機能 [4] を用いるためのライブラリで，ネットワークの物理的な性能に近い性能を出すことができる．また，libibverbs は InfiniBand だけではなく，RoCE や Internet Wide Area RDMA Protocol (iWARP) でも用いることができる．

メモリ登録を行う可能性があるのは，クライアント側では**図 3** のユーザバッファ，GFS_File 構造体のバッファ，gfs_connection 構造体のバッファである．ユーザバッファは pread, pwrite をユーザが発行するときに指定するバッファである．GFS_File 構造体は，ファイルをオープンするときに作成される構造体で，デフォルトでは 1 MiB のバッファが確保される．このバッファは pread, pwrite のリクエストサイズが小さいときに用いられる．ユーザバッファは pread, pwrite を発行するたびに指定されるため，この領域は予め登録することはできない．GFS_File 構造体は，ファイルをオープンする度に作成されるため，このバッファを静的に登録するとオープンするファイル数に応じて登録する領域が増えてしまう．gfs_connection 構造体は接続するファイルサーバである gfsd 毎に作成される構造体で，これまでは接続情報等が保持されていた．この領域は接続する gfsd の数だけしか作成されないため，ここに新たにバッファを確保すると，確保するバッファを gfsd の数に限ることができる．

これらの検討から，ユーザバッファおよび GFS_File 構造体のバッファは，静的に登録することは難しく，登録するとすれば動的に登録することとなる．一方，gfs_connection 構造体に新たにバッファを確保すれば，静的に登録することも可能になると考えられる．

一方，ファイルサーバの gfsd 側であるが，こちらはクライアントが接続する度にプロセスが生成される．クライアントからのアクセスは同期的な RPC しかないため，登録するバッファはプロセス当たり一つあればよく，静的に登録すればよい．

まとめると，静的に登録する場合は，クライアント側は gfs_connection 構造体に新たにバッファを確保し，登録する．gfsd 側は常にバッファを確保し静的に登録する．このとき，書込の場合，ユーザバッファのデータが 1 MiB 未満であれば GFS_File 構造体のバッファにバッファリングし，1 MiB を越えたとき，あるいばフラッシュされたときに，静的にメモリ登録された gfs_connection のバッファにコピーし，RDMA で転送する．1 MiB 以上の場合は，GFS_File 構造体のバッファはバイパスし，ユーザバッファのデータを，登録された gfs_connection のバッファにコピーし，RDMA 転送する．ここで，gfs_connection のバッファサイズと gfsd で登録したバッファサイズは，必要であれば拡張して登録しなおすことも考えられる．そのようにすると，RDMA の回数を減らすことができる．

読込の場合は，要求が 1 MiB 未満であれば，Gfarm では 1 MiB データを先読みするため，1 MiB のデータを gfsd_connection のメモリ登録されたバッファに RDMA で転送し，GFS_File 構造体のバッファにコピーする．その後，指定されたサイズだけユーザバッファにコピーすることとなる．1 MiB 以上であれば，gfsd_connection のメモリ登録されたバッファに RDMA で転送し，GFS_File 構造体のバッファはバイパスし，ユーザバッファにコピーする．以後，この方式を static 方式とよぶ．

一方，動的に登録する場合は，ユーザバッファあるいは GFS_File 構造体のバッファを登録する．書込の場合，1 MiB 未満のサイズのときは GFS_File 構造体のバッファにバッファリングされるため，フラッシュ時に GFS_File 構造体のバッファを動的に登録し，RDMA 転送する．1 MiB 以上の場合は，ユーザバッファを動的に登録し，RDMA 転送する．以降，この方式を dynamic 方式とよぶ．

なお，static 方式の場合も，（拡張した）静的に登録したバッファサイズ以上の書込の場合は，登録バッファサイズ毎にコピーして RDMA 転送を繰り返すか，あるいは dynamic に登録してコピーしないで RDMA 転送するかを選択することができる．

5. 性能評価

RDMA アクセスにおける static 方式，dynamic 方式の評価を行う．評価で用いた各ノードは，2.4GHz の 12 コア Xeon E5-2695v2 (Ivy Bridge-EP)×2 ソケット, 64GB メモリ (1866MHz 8GB DDR3×8) であり，InfiniBand FDR で接続されている．各ノードでは Linux 2.6.32 (CentOS 6.8)，libibverbs-1.1.8 を用いている．ストレージは，スト

ⓒ 2017 Information Processing Society of Japan

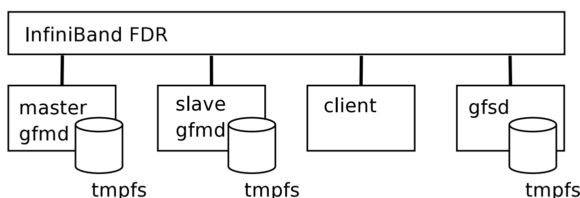

図 4　評価環境.

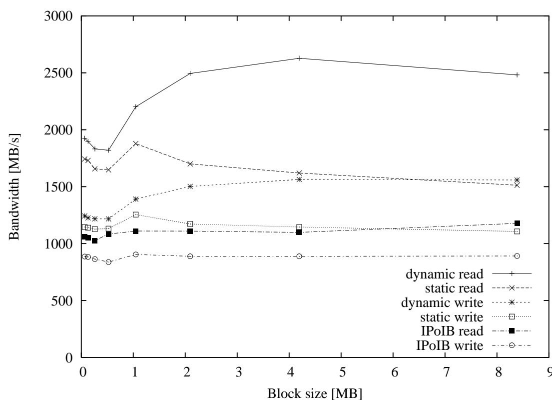

図 5　Gfarm API によるブロックサイズを変えたときのアクセス性能.

レージ性能がボトルネックとならないようにtmpfs（メモリ上のファイルシステム）で構成している．評価環境を**図 4**に示す．図中，master gfmd, slave gfmd はファイルシステムのメタ情報を管理するメタデータサーバであり，gfsd はファイルサーバである．InfiniBand FDR のシグナル帯域幅は 56.25 Gbps であるが，ib_write_bw による RDMA write のバンド幅計測では、6,079 MB/s であった．

5.1　Gfarm API による評価

Gfarm API を用いた書込，読込バンド幅の評価を行う．評価においては，thput-gfpio ベンチマーク [13] を用いた．8 GiB ファイルの読み書き性能を**図 5** に示す．

GFS_File 構造体のバッファは 1 MiB であり，ブロックサイズが 1 MiB 未満の場合は，書込においては，GFS_File 構造体のバッファにバッファリングされ，1 MiB になったところで書込がなされる．また，読込においては，バッファ分の 1 MiB のデータが先読みされる．そのため，全体的に 1 MiB を境に傾向が変わっている．IPoIB では RPC におけるデータ要求サイズは最大 1 MiB であるため，ブロックサイズが 1 MiB を超えたところでの性能の違いはほとんどなく，書込は 905 MB/s，読込は 1,178 MB/s であった．

書込においては, RDMA の static 方式では, GFS_File 構造体あるいはユーザのバッファから gfs_connection 構造体のバッファへコピーして RDMA 転送する．一方，dynamic 方式では，GFS_File 構造体あるいはユーザのバッファを動的にメモリ登録して RDMA 転送する．読込においては，RDMA の static 方式では，gfs_connection 構造体のバッファに RDMA コピーし，GFS_File 構造体あるいはユーザのバッファにコピーする．一方，dynamic 方式では，GFS_File 構造体あるいはユーザのバッファを動的にメモリ登録して RDMA 転送する．

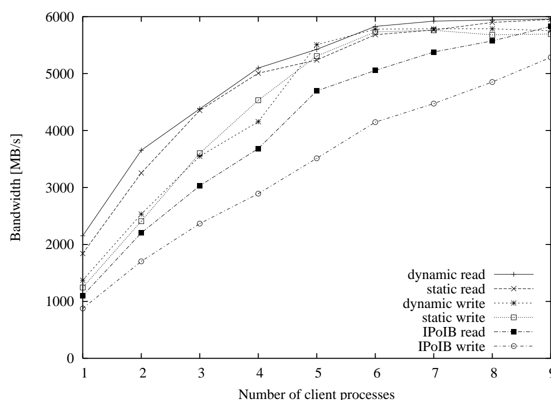

図 6　Gfarm API による並列アクセス性能.

図 5 より，dynamic 方式の方が static 方式より高速である．このため, メモリコピーのオーバヘッドはメモリ登録のオーバヘッドより大きいことが分かる．gfs_connection 構造体の, 静的に登録するバッファは 16 MiB まで拡張可能としているが，static 方式は 1 MiB を越えると性能が落ちている．static 方式ではユーザのバッファから gfs_connection 構造体のバッファへコピーを行っているが，そのメモリコピーのオーバヘッドが大きいと考えられる．一方, dynamic 方式は, ユーザのバッファを毎回メモリ登録して RDMA 転送しているが，アクセス性能は向上している．また，static 方式と dynamic 方式の性能差が広がっていることから，メモリコピーとメモリ登録のオーバヘッド差はデータサイズが大きくなると広がることが分かる．dynamic では，4 MiB ブロックサイズのときに書込で 1,563 MB/s，読込で 2,627 MB/s を達成している．IPoIB に比べると，書込で 1.7 倍，読込で 2.2 倍の性能向上をしている．

図 6 に複数クライアントプロセスによる並列アクセス性能を示す．並列アクセス性能は全クライアントのバンド幅の合計を示している．ブロックサイズは 1 MiB である．概ね性能はどのアクセス方法もクライアントプロセス数に対しスケールし，6 GB/s 近くまで達している．ブロックサイズが 1 MiB であるため，1 クライアントプロセスでは dynamic 方式と static 方式の性能差は読込で 1.7 倍，書込で 1.1 倍であるものの，クライアントプロセス数が増えるに従い，その差は小さくなっていく．

5.2　POSIX API による評価

gfarm2fs でマウントし POSIX API を用いてアクセスした場合のアクセス性能を**図 7** に示す．評価においては，thput-fsys ベンチマーク [13] を用いた．gfarm2fs は

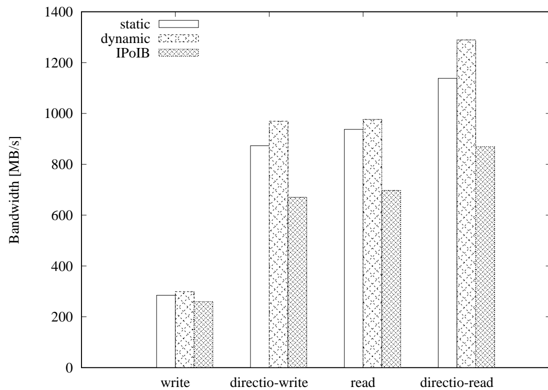

図 7　POSIX API によるアクセス性能.

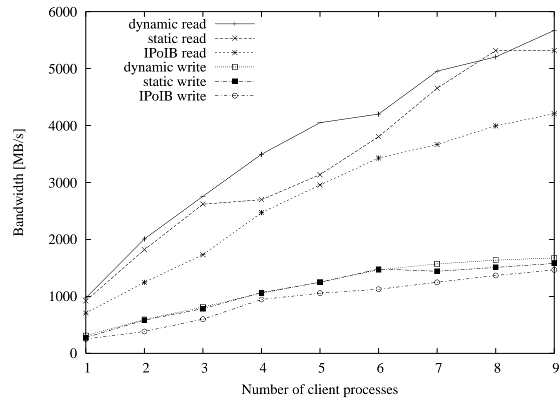

図 8　POSIX API による並列アクセス性能.

Gfarm ファイルシステムをマウントするプログラムである．directio-write, directio-read はそれぞれ direct_io オプションを用いてマウントした場合である．direct_io オプションをつけることにより，カーネルのページキャッシュをバイパスし，メモリコピーを削減することができる．また，direct_io オプションをつけない場合は，カーネルからの write リクエストのサイズは 4 KiB であるが，direct_io オプションをつけると 128 KiB まで増える．なお，read リクエストのサイズはいずれの場合も 128 KiB まで増える．

書込については，direct_io オプションをつけない場合，IPoIB で 260 MB/s，RDMA の static 方式で 285 MB/s，dynamic 方式で 299 MB/s と全体的に性能が低く，性能差は小さい．write のブロックサイズが 4 KiB と小さいのが原因の一つと考えられるが，Gfarm ライブラリはバッファリングし，1 MiB 単位でネットワーク転送を行うため，性能低下はここまで大きくないはずである．実際，Gfarm API を用いた 4 KiB ブロックサイズの RDMA の dynamic 方式の書込性能は 1,280 MB/s である．それに対し，POSIX API では 299 MB/s に低下しており，ページキャッシュ利用のオーバヘッドが大きいことが分かる．

direct_io オプションをつけると IPoIB で 2.5 倍，RDMA の dynamic 方式では 3.2 倍も性能が向上する．write のリクエストサイズが 128 KiB になったことと，ページキャッシュを利用しないことで大きく性能が向上していると考えられる．IPoIB では 670 MB/s であったが，RDMA の static 方式では 873 MB/s，dynamic 方式では 969 MB/s となり，dynamic 方式では IPoIB に比べ 45%の性能向上を示した．ただし，Gfarm API では 128 KiB ブロックサイズの RDMA の dynamic の書込性能は 1,226 MB/s であるため，POSIX API でカーネルを経由することにより性能は 20%低下している．

読込については，direct_io オプションをつけず，ページキャッシュを利用する場合でも書込のときほど性能は落ちていない．read のリクエストサイズが 128 KiB であることが影響していると考えられる．IPoIB では 697 MB/s で

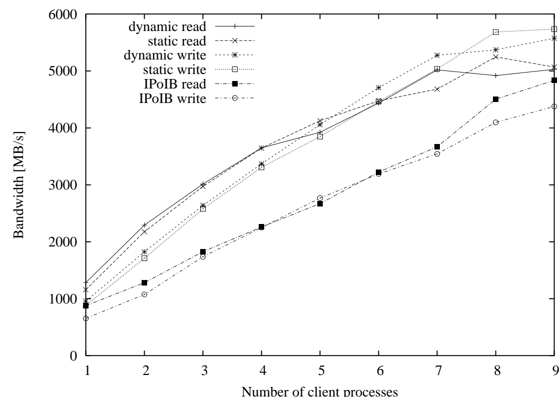

図 9　Direct IO でマウントした場合の POSIX API による並列アクセス性能.

あったが，RDMA の static 方式では 937 MB/s，dynamic 方式では 976 MB/s となった．ただし，Gfarm API では 128 KiB ブロックサイズの RDMA の dynamic 方式の読込性能は 1,897 MB/s であるため，POSIX API でカーネルを経由することにより性能は 49%低下している．

direct_io オプションをつけると，性能は 1.2 倍から 1.3 倍向上し，IPoIB では 869 MB/s，RDMA の static 方式で 1,138 MB/s，dynamic 方式で 1,289 MB/s となった．IPoIB に比べ RDMA の dynamic 方式では 48%の性能向上を示した．ページキャッシュを利用しないことにより性能が向上し，RDMA の dynamic 方式の性能では Gfarm API からの性能低下は 32%となった．

POSIX API を用いた複数クライアントプロセスによる並列アクセス性能を**図 8** に示す．direct_io オプションは用いていない．書込性能はどのアクセス方式でも低く，9 クライアントプロセスでの並列アクセスでも 1,700 MB/s に達していない．一方で，読込性能は 9 クライアントプロセスからの並列アクセスで RDMA の dynamic 方式では 5,671 MB/s を達成している．IPoIB では 9 クライアントプロセスからの並列アクセスで 4,209 MB/s であり，IPoIB に比べ RDMA の dynamic 方式の性能向上は 34%である．

direct_io オプションを用いてマウントした場合の, POSIX

ⓒ 2017 Information Processing Society of Japan

API を用いた複数クライアントプロセスによる並列アクセス性能を**図 9** に示す．IPoIB では書込，読込ともに 1 クライアントプロセスあたりの性能が低く，9 クライアントプロセスでも書込で 4,376 MB/s，読込で 4,834 MB/s ほどの性能しか達していない．RDMA では，9 クライアントプロセスでは概ねネットワークの物理性能近くまで達しているが，クライアントプロセス数が少ないときは，Gfarm API による並列アクセス性能に比べると 30%から 40%ほど性能が低い．

6. まとめ

Gfarm ファイルシステムにおける RDMA アクセスの設計を行い，その性能評価を行った．設計は，大幅な設計変更を伴わず，RDMA の性能が生かせることを目標として行った．そのため，pread, pwrite のファイルデータの転送に RDMA write, RDMA read を用い，RPC のメッセージ送信はソケット通信を用いている．RDMA 転送で必要となるメモリ登録方式については，静的に登録する static 方式と，動的に登録する dynamic 方式について評価を行った．その結果，static 方式より dynamic 方式の方が性能が高く，メモリコピーのオーバヘッドはメモリ登録のオーバヘッドより大きいことが分かった．RDMA を用いることにより，IPoIB に比べ，書込で 1.7 倍，読込で 2.2 倍の性能向上を達成した．また，並列アクセスにおいては，より少ないクライアントプロセス数でネットワークの物理性能近い性能を達成することができた．POSIX API におけるアクセスではページキャッシュをバイパスすることにより高い性能を得ることができ，IPoIB に比べ書込で 45%，読込で 48%の性能向上を達成した．なお，RDMA アクセスを行う Gfarm ファイルシステムは，2016 年 12 月に Gfarm 2.7.0[13] としてリリースした．

謝辞 本研究の一部は JST-CREST「ポストペタスケールデータインテンシブサイエンスのためのシステムソフトウェア」，「EBD：次世代の年ヨッタバイト処理に向けたエクストリームビッグデータの基盤技術」，「広域撮像探査観測のビッグデータ分析による統計計算宇宙物理学」による．

参考文献

[1] Braam, P. J.: *Lustre File System*. http://www.lustre.org/.
[2] Callaghan, B., Lingutla-Raj, T., Chiu, A., Staubach, P. and Asad, O.: NFS over RDMA, *Proceedings of the ACM SIGCOMM Workshop on Network-I/O Convergence: Experience, Lessons, Implications*, pp. 196–208 (2003).
[3] Cooper, B. F., Silberstein, A., Tam, E., Ramakrishnan, R. and Sears, R.: Benchmarking Cloud Serving Systems with YCSB, *Proceedings of the 1st ACM Symposium on Cloud Computing*, pp. 143–154 (2010).
[4] Hilland, J., Culley, P., Pinkerton, J. and Recio, R.: *RDMA Protocol Verbs Specification* (2003). https://tools.ietf.org/html/draft-hilland-rddp-verbs-00.
[5] Islam, N. S., Rahman, M. W., Jose, J., Rajachandrasekar, R., Wang, H., Subramoni, H., Murthy, C. and Panda, D. K.: High Performance RDMA-based Design of HDFS over InfiniBand, *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, pp. 35:1–35:35 (2012).
[6] OpenFabrics Allaiance: http://openfabrics.org/.
[7] RDMA core userspace libraries and daemons: https://github.com/linux-rdma/rdma-core/.
[8] Recio, R., Metzler, B., Culley, P., Hilland, J. and Garcia, D.: *A Remote Direct Memory Access Protocol Specification* (2007). RFC 5040.
[9] Sasaki, S., Takahashi, K., Oyama, Y. and Tatebe, O.: RDMA-Based Direct Transfer of File Data to Remote Page Cache, *Proceedings of 2015 IEEE International Conference on Cluster Computing*, pp. 214–225 (2015).
[10] Schmuck, F. and Haskin, R.: GPFS: A shared-disk file system for large computing clusters, *Proceedings of the 2002 Conference on File and Storage Technologies (FAST)*, pp. 231–244 (2002).
[11] Talpey, T. and Callaghan, B.: *Network File System (NFS) Direct Data Placement* (2010). RFC 5667.
[12] Talpey, T. and Callaghan, B.: *Remote Direct Memory Access Transport for Remote Procedure Call* (2010). RFC 5666.
[13] Tatebe, O.: *Gfarm File System*. http://sourceforge.net/projects/gfarm/.
[14] Tatebe, O., Hiraga, K. and Soda, N.: Gfarm Grid File System, *New Generation Computing*, Vol. 28, No. 3, pp. 257–275 (2010).

ⓒ 2017 Information Processing Society of Japan