情報処理学会論文誌　コンピューティングシステム　Vol. 4　No. 1　83–94 (Feb. 2011)

MPI-IO 書き込みのパイプライン処理による高速化

堀　敦史[†1,*1]　太田一樹[†1,*2]　安井　隆[†2]
鴨志田良和[†1]　松葉浩也[†1,*3]
住元真司[†3]　石川　裕[†1]

並列プログラムのファイルアクセスは基本的にファイルに対する不連続なアクセスとなり，逐次プログラムのファイルアクセスに比べ性能が低下する場合がある．本稿では，並列ファイルへの書き込みにおいてファイルサーバとユーザプロセス間の通信トポロジをリングとし，リングに沿ったパイプライン処理により性能を向上させる技法を提案する．提案された技法は MPI-IO の ROMIO ADIO デバイスとして実装され，MPI-IO ベンチマークプログラムを用いて代表的な並列ファイルシステム PVFS2 および Lustre と性能を比較する．その結果，単純なアクセスパターンではハードウェア資源量の多い並列ファイルシステムの方が有利である一方，複雑なアクセスパターンでは本稿で提案する技法がより良い性能を発揮することを示す．提案する技法は，単一のファイルサーバを用いているが，複数のファイルサーバを用いる並列ファイルシステムの性能を上回り，より費用対効果が高い．本稿での提案は，パイプライン処理による並列ファイル書き込みの高速化という点でユニークと考える．

Pipeline Optimization of MPI-IO Write

Atsushi Hori,[†1,*1] Kazuki Ohta,[†1,*2] Takashi Yasui,[†2]
Yoshikazu Kamoshida,[†1] Hiroya Matsuba,[†1,*3]
Shinji Sumimoto[†3] and Yutaka Ishikawa[†1]

The file access pattern of a parallel file is usually a large set of non-contiguous access, and therefore the access speed of a parallel file degrades much worse than that of sequential access to a sequential file. In this paper, it is proposed to access a parallel file in a pipelined way along with a ring communication topology to improve the access speed of a file. The proposed technique is implemented as a ROMIO ADIO device of MPI-IO, evaluated with MPI-IO benchmark programs, and compared with the PVFS2 and Lustre parallel file systems. The evaluation results show that the parallel file systems having richer hardware resources exhibit better performance when the file access patterns are coarse and simple, however, the proposed technique having less hardware resources exhibits better performance when the access patterns are fine and complex. Thus the proposed technique is tought to have higher cost-performance. The proposed pipeline technique utilizing a ring topology is new and unique.

1. はじめに

磁気ディスクに格納されたファイルへのアクセスは，ファイルを逐次的にアクセスした方がランダムアクセスに比べ速いことは既知のとおりである．一方，クラスタのように複数のプロセスから構成される並列プログラムによるファイルの並列アクセスは，複数のプロセスがファイルのそれぞれ異なった部分を同時にアクセスする場合が多い[6]．これは根本的に不連続なアクセスであるだけでなく，並列度が大きい場合には小さいレコードを大量にアクセスすることから，著しい性能の低下を生じる場合があることが知られており，これに対処する方式も提案されている[1),4),15)]．

これまでに提案されたこの問題に対する解決策では，基本的にクライアント N に対しファイルサーバが M 台あり，それらの間は N 対 M の通信ネットワークがあるものと仮定されており，通信のトポロジに着目していない．本稿は，通信トポロジに着目し，1 台のファイルサーバへ並列書き込み要求をパイプライン的に最適化する技法について提案するものである．

本稿で提案される技法は，オンデマンドなファイルステージングシステム Catwalk[5),18),19)] をベースに，MPI-IO ROMIO[16)] の ADIO デバイス[14)]（Catwalk-ROMIO と呼ばれる）として実装されている．また Catwalk-ROMIO は NAS 並列ベンチマークの BT-IO[17)] お

†1 東京大学
The University of Tokyo
†2 日立製作所
Hitachi, Ltd.
†3 富士通研究所
Fujitsu Laboratories, Ltd.
*1 現在，理研 情報科学研究機構
Presently with Riken AICS
*2 現在，Preferred Infrastructure, Inc.
Presently with Preferred Infrastructure, Inc.
*3 現在，日立製作所
Presently with Hitachi, Ltd.

よび IOR[12] を用いて評価される．評価においては代表的な並列ファイルシステムである PVFS2[3] および Lustre[7] と性能の比較する．その結果から，複雑なファイルアクセスパターンの場合，1 台のファイルサーバしか用いない Catwalk-ROMIO の性能が，複数のファイルサーバを用いている並列ファイルシステムを上回ることを示す．このことから，パイプライン方式による並列ファイル書き込みの高速化技法の有効であると考える．Catwalk-ROMIO については文献 18) ですでに報告されているが，1) 新たな書き込み最適化技法の提案，2) MPI-IO ベンチマークプログラムによる評価，3) 代表的な並列ファイルシステムとの比較，の 3 点が本稿の焦点である．

2. Catwalk-ROMIO

2.1 背　　景

Catwalk-ROMIO[18] は Catwalk[19] 同様，リング状の通信トポロジを持ち，リングに沿ってデータをパイプライン転送する点がユニークな特徴である（**図 1**）．Catwalk では，ファイルの読み込みオープンの際にファイルサーバのファイルを計算ノードにコピーし，実際の読み込み要求で計算ノード上のユーザプロセスがコピーされたローカルファイルから直接読み込む．パイプライン転送によるファイルのコピー速度はノード数にほとんど依存しない．この結果，細粒度の遠隔ファイルアクセスをローカルに行うことができ，読み込み性能が向上する[5]．この方式は，見方を変えれば Data Sieving[15] と共通するアイデアである．ただし，Catwalk がつねにファイル全体をコピーするのに対し，Data Sieving は細粒度のアクセス群を包含するような連続したファイルの一部分をファイルサーバから転送する，という違いがある．

Catwalk-ROMIO は Catwalk をベースに MPI-IO の実装である ROMIO の ADIO デバイス[14] として開発されたものである[18]．最初の Catwalk-ROMIO[18] における共有ファイルの書き込みでは，計算ノード上のユーザプロセスからのファイル書き込み要求はすべてローカルディスク上にジャーナルとして記録される．ファイルがクローズされた時点で，ジャーナルファイルの内容をリングに沿ってファイルサーバに送り，最終的にファイルサーバ上の目的とするファイルに書き込まれる．リングに沿ってパイプライン転送する際，それぞれの計算ノード上でジャーナルレコードはマージソートの要領でソートされる．もしローカルに生成されたジャーナルレコードが正しくソートされていた場合，リング転送によるマージソートにより，最終的にファイルサーバでは余分なシーク操作をともなわずに高速な逐次書き込みが可能になる．

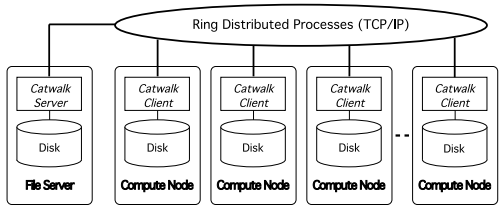


図 1　リング分散プロセス構造
Fig. 1　Ring distributed process structure.

文献 18) で判明した問題点のいくつかはすでに解消されている．1 つはリング通信トポロジ生成オーバヘッドの問題，もう 1 つは計算ノード上で複数のユーザプロセスが複数のジャーナルファイルに同時に書き込むことによる性能低下である．前者の問題は，MPI-IO の collective な操作に対応するために `MPI_File_open()` の呼び出しごとにリング状の分散プロセス構造を生成する必要があることに起因する．この問題への対処として，ジョブの起動と同時に Catwalk クライアントプロセスをすべての計算ノード上で起動しておき，`MPI_File_open()` が呼び出された時点で，リングを生成するためのプロセスを `fork()` するようにした．これにより，`MPI_File_open()` 呼び出しのつど，コストの大きい `exec()` を呼ばないで済む．後者の問題は，同じ計算ノード上のすべてのユーザプロセスからの書き込み要求は，いったん計算ノードに 1 つだけ存在する Catwalk クライアントプロセスを経由させ，Catwalk クライアントプロセスがまとめて 1 つのジャーナルファイルに書き込むことで回避した．

後者の改良は新たな問題を生じる．前述したように，Catwalk サーバプロセス上で受け取るジャーナルレコードが正しくソートされているためには，各ノードのジャーナルレコードが整列していなければならない．すべてのユーザプロセスがソートされた状態で書き込み要求を出したとしても，Catwalk クライアントプロセスが複数のユーザプロセスからの要求ストリームをマージした時点で，ソートされている保証はなくなってしまう．本稿はこの問題に焦点をあて，次章以降，この問題に対処するためのアイデアと，リングに沿ったパイプライン転送による最適化手法について提案する．

3. リングトポロジによる最適化

ここではリングトポロジを用いて，最終的に Catwalk サーバプロセスがファイルに書き込む速度を最大にする方法について検討する．基本的には非連続なファイル書き込み要求を

並べ直し，できるだけ逐次的なアクセスとすることと，細粒度の連続した書き込みをまとめてより大きな書き込みとすることの 2 点に着目する．

ジャーナルレコードは，ファイル内のオフセット，書き込むデータ長，そして書き込むデータ列で構成される．また，ジャーナルレコードのオフセットをキーとしてレコードをソートする．ソートの対象となるレコードの順序集合において，すべての隣り合うレコード間の順序とオフセットの前後関係が同じである場合，「完全なソート」と呼ぶことにする．すべてのレコードを完全にソートしたとしても，レコードにオーバラップがあったり，書き込みに間隙がある場合にはシーク回数をゼロにすることはできない．

3.1　ソートバッファの溢れ処理

メモリ内で完全なソートを実現するには，メモリ内にすべてのレコードを保持する必要がある．換言すれば，メモリ内にすべてのレコードを保持できないのであれば完全なソートはあきらめなければならない．ジャーナルをソートするための必要なバッファはメモリの制約から有限と考える必要があり，ソートバッファ内にすべてのレコードは収まらないと考えなければならない．ユーザプロセスがメモリの大半を利用できるようにするためである．本稿の議論では，ソートバッファの大きさの制限から完全なソートをあきらめている．本稿の目的はファイル書き込みの高速化であり，ソートが不完全なために生じるシーク操作は，少なければ少ないほど性能向上が見込めるが，かならずしもシーク操作の回数をゼロにする必要はない．シークの頻度が十分に少ないことが重要となる．

有限ソートバッファが満杯になった場合の処理として 2 つの方式を考える．1 つは，バッファ内で最も小さいオフセットを持つレコードをローカルなファイルに書き出し，`MPI_File_close()` が呼ばれた時点でこのファイルの先頭から順次リングに沿ってサーバプロセスに送る方法である（**図 2**）．もう 1 つの方法は，最も小さいオフセットを持つレコードをリングトポロジにおける隣のノードに送りつける方法である（**図 3**）．

前者では，OS カーネルがファイルに書き込むデータを空いているメモリにキャッシュするため，データ量が少ない場合には書き出すためのオーバヘッドは小さいが，空きメモリ容量が少ない場合には実際にディスクに書き出され，オーバヘッドが大きくなる．一方，後者の場合は，ユーザプロセスが要求した I/O により Catwalk クライアントプロセス間のリング通信（ノード間通信）が発生する．このため，計算と I/O のフェーズが明確に分離しているようなアプリケーションでは，トポロジ的に前のノードの I/O フェーズの最後で発生したバッファから溢れたメッセージを，次の計算フェーズ中に転送する可能性があり，アプリケーションプログラムの実行に擾乱を与えてしまう可能性がある．これはネットワークの速度が遅い場合に，より顕著となると考えられる．

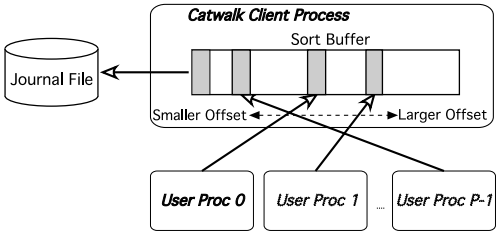


図 2　ソートバッファの溢れ処理—ローカルジャーナル
Fig. 2　Overflow process of sort buffer – local journal.

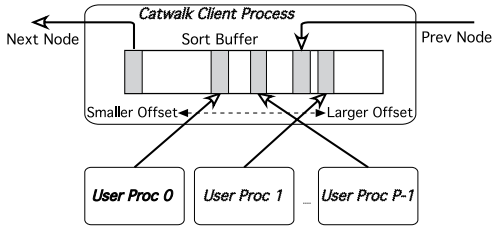


図 3　ソートバッファの溢れ処理—メモリ内ソート
Fig. 3　Overflow process of sort buffer – in-memory sort.

3.2　レコードのソート

Catwalk クライアントプロセスでは，同じノードの複数のユーザプロセスからのジャーナルレコードを整列させる「ノード内のソート」と，前のノードから送られてきたジャーナルレコードとローカルなレコードをマージソートして，次ノードに送るという「ノード間のソート」がある．ソートに必要なバッファサイズ（保持できるレコードの数）は，大きいほど，より完全なソートに近くなり，シークの回数を減らすことができる．ノード間ソートにおいては，ローカルなファイルに書き出した方がより完全なソートに近くなる可能性がある．これはノード間のソートにおいて，ローカルなジャーナルファイルをソートバッファの一部と考えることができ，より大きなソートバッファと見なすことができるからである．

3.3　レコードの結合

リングトポロジにおける転送はパイプライン動作が基本なので，Catwalk-ROMIO の内部処理で扱うレコードの最大長（`DATA_MAX`）は，ユーザプロセスからの書き込み要求を受

付けた時点から，ファイルサーバにおけるファイル書き込みまで同じ値になっている．

ここで 2 つのレコードに着目する．そのうちの 1 つのレコードのオフセットとデータ長を足した値が，もう 1 つのレコードのオフセットと同じであった場合，それらのレコードは「隣接している」関係にあると定義する．ソートバッファ内で連続した並びにある 2 つのレコードが隣接した関係にあり，かつそれら 2 つのデータ長の和が `DATA_MAX` 以下であれば，それら 2 つのレコードは結合して 1 つのレコードとすることができる．このようにレコードがソートしてあるとレコードの隣接判定が容易となる．より積極的に結合する方法として，データ長の和が `DATA_MAX` を超えた場合においても，次のレコードの一部のデータを結合する方法が考えられるが，本稿ではこの方法によるレコードの結合は行っていない．

複数のレコードを結合をすることで，I/O トランザクションの回数を減らすことが期待される．また，リングに沿ったパイプライン転送においては，レコード長が均一化され，パイプラインを構成する各段の処理時間も均一化されると期待でき，全体としてパイプライン処理が均等化され，より効率的となると考えられる．

4. 評　　価

T2K オープンスパコン（東大）[13] の 17 ノード（ファイルサーバ 1 台と計算ノード 16 台）を用いた．計算ノードの概要を**表 1** に示す．ファイルサーバは計算ノードと等価である．本クラスタには 1 Gb/s の Ethernet と並列計算用の Myrinet 10 G の 2 種類のネットワークがそれぞれ 2 本ある．表 1 には Bonnie++ [11] で計測したディスク性能もあわせて示した．

Catwalk-ROMIO は MPICH-2（mpich2-1.0.6 p1）の ADIO デバイスとして実装する．以下の評価プログラムの MPI 通信はすべて Myri10 G を使っている．Catwalk-ROMIO の通信には，Ethernet（1 Gb/s，MTU 1,500 Bytes）と，Myri10 G を用いた IP over Myrinet（10 Gb/s，MTU 9,000 Bytes）の 2 通り行った．

表 1　T2K（Tokyo）ノードの仕様
Table 1　Compute node specification of T2K (Tokyo).

CPU	AMD Barcelona, 2.3 GHz, 16 Cores
Memory	32 GiB
Local Disk	SATA
Network	Intel E1000（1 Gb/s）×2
	Myri10 G ×2
OS	RHELS 5.1
File System	EXT3
Bonnie++	Seq. Block In: 49.52 MiB/s
	Seq. Block Out: 39.76 MiB/s

ファイルサーバ上では 1,024 個の FIFO バッファ（それぞれ `DATA_MAX` 長）を持ち，書き込み専用のスレッドを立て順次ディスクに書き込む．ディスクのシークの影響を調べるためさらに別スレッドを立てて，書き込み専用スレッドが 64 MiB 書き込むごとに，Linux の `fdatasync()` システムコール[⋆1]を呼ぶようにした．`MPI_File_close()` の呼び出し時には，これらのバッファのすべての書き出しと，それに続く `fdatasync()` の完了を待ってからユーザプロセスに完了を通知している．`DATA_MAX` の値は 32 KiB とした．

ローカルディスクにジャーナルレコードを書き込む方式（これを “journal” と呼ぶ），書き込みをせずソートバッファ内だけでソートを行う方式（“sort”）と，そのそれぞれの方式でレコードを結合する/しない，を組み合わせた 4 方式に加え，ソートをまったく行わずにユーザプロセスから渡されたレコードをそのままリングに沿ってサーバに送る方式（“immediate”）の 5 つの方式で評価を行った．Immediate 方式は sort 方式におけるソートバッファの大きさがゼロの場合と見なすこともできる．Immediate 以外の 4 方式では，ソートバッファの大きさを，16 MiB，64 MiB，256 MiB，1,024 MiB と変えることで，ソートバッファの大きさが性能に与える影響を調べることとした．ソートバッファは，領域の管理を容易にするために `DATA_MAX` 長でブロック化されている．

比較のための PVFS2 [2),3)] は I/O サーバとメタデータサーバを兼用する T2K ノード（表 1）を別途 4 台割り当てた．計算ノードは 1 台のディスクを持つ．PVFS2 の設定は “TroveSyncMeta no”（メタデータの sync なし），“TroveSyncData yes”（データの sync あり），“TroveMethod alt-aio”（非同期 I/O）とした．PVFS2 のネットワークは Myri10G（IP over Myrinet ではない．トランキングの設定なし）を用いた．

以下の評価のすべてのケースにおいて，評価プログラムの実行には，計算ノード数は 16，ノードあたりのプロセス数は 16，合計 256 プロセスを用いている．初版の Catwalk-ROMIO ではノード数およびプロセス数が大きい場合に性能が低下することが判明している [18]．本稿での評価において，ノード数およびプロセス数は評価環境で実行可能な最大数に固定し，最悪と考えられる条件でのみ評価を行う．これはベンチマークプログラムおよびアクセスパターンの違いによる性能変化を調べることを優先したためである．

⋆1 `fsync()` 同様バッファキャッシュの内容をディスクに書き込むが，inode の属性の変更は必ずしもディスクに反映されない．

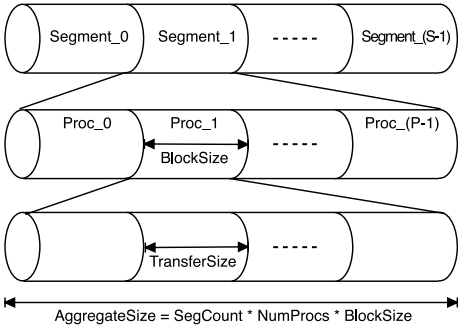


図 4　IOR パラメータの関係
Fig. 4　IOR parameters.

4.1　IOR

評価に用いた IOR [12]（バージョン 2.10.2）は MPI-IO 計測用にビルドした．**図 4** に IOR の計測パラメータの関係を示す．TrnasferSize は IOR が発行する 1 回の MPI-IO 呼び出しにおけるデータ長であり，1 つの MPI プロセスが BlockSize で規定された長さの I/O を発行する．したがって，BlockSize を TransferSize で割った数だけ MPI-IO の書き込み関数が呼ばれることになる．以上の処理は Segment 数だけ繰り返される．またアクセス方式は “single-shared-file” で collective でなく（`MPI_File_write_at()` を呼んでいる），ファイルポインタは “independent”（プロセスごとに独立）である．Catwalk-ROMIO 内部の `DATA_MAX` は 32 KiB なので，IOR の TransferSize が 32 KiB よりも小さくないとレコードの結合は発生しない．Catwalk-ROMIO では書き込みの一部がクローズ処理内で行われるが，IOR が計算し出力するバンド幅には，ファイルの書き込み時間だけでなく，オープンとクローズの時間も含まれている．

4.1.1　最も単純なパターン

最初の評価は最も単純なパターンである．IOR のパラメータの TransferSize と BlockSize を同じ値とし，Segment 数は 1 とした．この設定は，ファイルを単純にプロセス数で分割するだけである．**図 5** に 4 つのグラフを示す．上の 2 つのグラフは TransferSize，BlockSize ともに 1 MiB，下の 2 つのグラフは 4 MiB に設定した場合である．左側の 2 つのグラフは Catwalk-ROMIO のネットワークとして Ethernet を用いた場合，右側の 2 つは Myri10 G を用いた場合である．

TransferSize が Catwalk-ROMIO の `DATA_MAX` より大きいことからレコードの結合は原

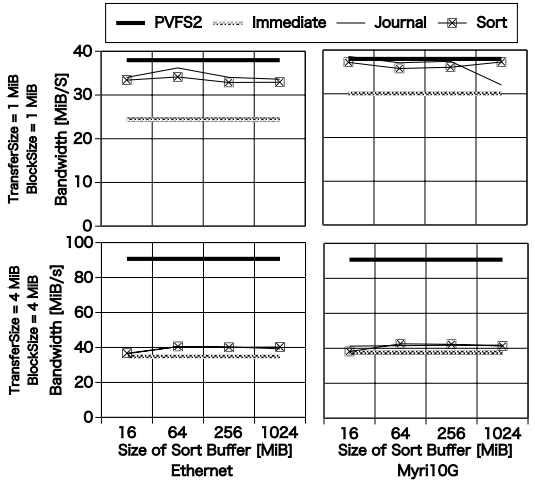


図 5　IOR—TransferSize==BlockSize
Fig. 5　IOR – TransferSize==BlockSize.

理的に発生しない．このため Catwalk-ROMIO は immediate，journal そして sort の 3 種類の方式で計測した．ちなみに TransferSize が 1 MiB のときに生成されるファイルの大きさは 256 MiB，4 MiB のときは 1 GiB である．図 5 のグラフではソートバッファの大きさの変化を横軸に，IOR が計測し出力したバンド幅を縦軸にとった．

Immediate 方式，PVFS2 ではソートバッファを用いないので水平の線とした．PVFS2 の計測においてネットワークは Myri10 G のみを使ったが，比較を容易にするため，Catwalk-ROMIO の Ethernet を用いたグラフにも同じ値の線を引いてある．これは他の IOR の評価グラフにおいても同様である．

TransferSize と BlockSize をともに 1 MiB に設定した場合（上側の 2 つのグラフ），Catwalk-ROMIO ではソートバッファの大きさの影響は少なく，およそ 30 から 40 MiB/s のバンド幅を実現している．4 MiB に設定した場合（下側の 2 つのグラフ），Catwalk-ROMIO では，ソートバッファの値とはほぼ無関係に，ディスクのバンド幅（表 1）に近い値になっている．まったくソートを行わない immediate 方式でも，ソートを行った場合に近い値を出している．また，まったくソートを行わない immediate 方式では，Myri10 G を用いた場合の性能が高く，ネットワーク性能に依存している度合いが，ソートを行っている場合に比

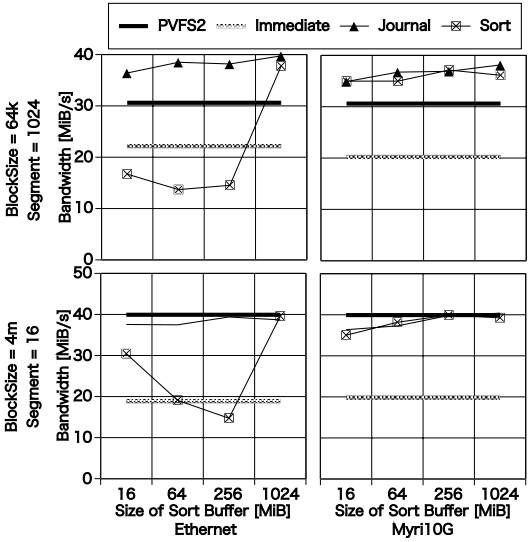


図 6　IOR—複数 Segment（ファイルサイズ一定）
Fig. 6　IOR – Multiple segments (same file size).

べ大きいと考えられる．

Catwalk-ROMIO の方式などの違いにより性能があまり変わらないのは，この IOR のパラメータではシークが発生する回数がもともと少ないためである．TransferSize と BlockSize の値が等しい設定では，最大でもプロセス数と同じ数しかシークが発生しない．実際，TransferSize が 1 MiB のときに Catwalk-ROMIO が書き込む総ブロック数は 8,192 であるが，書き込みが不連続となる回数は 220〜240 であった．本稿における評価ではプロセス数が 256 と Catwalk-ROMIO が扱うブロック数に比べ十分に小さく，性能への影響が少なかったものと考えられる．

PVFS2 は 4 台のサーバを従えているにもかかわらず，TransferSize と BlockSize を 1 MiB に設定した場合は 40 MiB/s 弱，4 MiB に設定した場合は 90 MiB/s 強程度のバンド幅であった．

4.1.2　複数セグメント（一定ファイルサイズ）

図 6 の上側 2 つのグラフは BlockSize を 64 KiB，Segment 数を 1,024 にした場合，下側 2 つは BlockSize を 4 MiB，Segment 数を 16 にした場合である．先の図同様，左側が

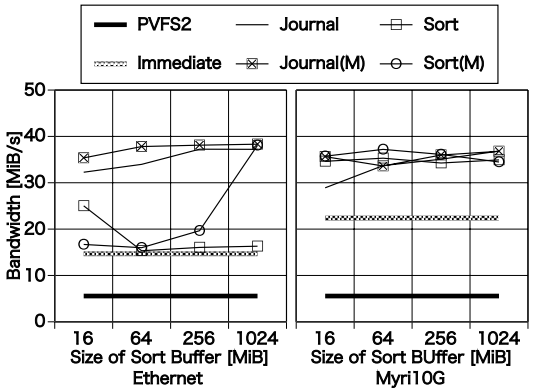


図 7　IOR—細粒度の書き込み
Fig. 7　IOR – Fine grain write.

Ethernet の場合であり，右側が Myri10 G を用いた場合である．どちらの場合でも最終的に生成されるファイルの大きさは 16 GiB である．また TransferSize は 32 KiB と `DATA_MAX` と同じ値なので Catwalk-ROMIO のレコード結合を有効とした計測は行っていない．

PVFS2 の性能は，Segment 数が 1,024 と多いときのバンド幅は約 30 MiB/s と Catwalk-ROMIO の journal 方式よりも下回っている．一方，Segment 数が 16 と少ないときでも Catwalk-ROMIO の journal 方式とほぼ同じ 40 MiB/s 程度しかない．

Catwalk-ROMIO で特徴的なのは，ソートバッファの大きさを変化させたときの sort 方式におけるバンド幅の変化が，Ethernet を用いた場合に大きいが，Myri10 G を用いたときには変化が少ないことである．この現象はソートバッファが溢れたときのレコードのバケツリレー式転送処理において，Ethernet の転送が Myri10G よりも遅いため，IOR の動作に悪影響をおよぼしたためと推測される．

4.2　細粒度の書き込み

図 7 は TransferSize を 4 KiB，BlockSize を 16 KiB，Segment 数を 1,024 とし，4 GiB のファイルを生成したときのバンド幅のグラフである（左側は Ethernet，右側は Myri10 G）．今回は TransferSize が小さいために Catwalk-ROMIO の方式に結合ありの journal 方式（図中，“Journal(M)”），結合ありの sort 方式（図中 “Sort(M)”）が加わっている．

同じ条件の PVFS2 のバンド幅は 5.56 MiB/s とかなり悪い結果となっている．一方，Catwalk-ROMIO の journal 方式では 40 MiB/s に近い値が出ている．前節の評価結果と同

様に，特に Ethernet の場合に Catwalk-ROMIO の方式の違いが顕著に現れている．Journal 方式では，結合ありの方が若干良い結果となっている．一方，結合なしの sort 方式では，ソートバッファが 1,024 MiB でも低いままである．これはレコードが結合されないため，見かけのバッファサイズが小さくなり，ソートが不十分だったためと考えられる．

4.3 BT-IO

NAS 並列ベンチマークプログラムの BT-IO（バージョン 3.3）[17] を用いて計測した．計測にあたり，BT プログラム中の Two-Phase I/O [4] のバッファの設定（`collbuf_nodes`）は 256 に変更してある[*1]．BT-IO にはいくつかの I/O 方式を選択できるが，今回はその中から MPI-IO を用いる simple と full で評価を行った．Catwalk-ROMIO のネットワークは Ethernet のみを用いた．BT-IO では別途内部で計算した I/O 速度が表示されるが，`MPI_File_close()` 処理時間が含まれていない．Catwalk-ROMIO では `MPI_File_close()` 処理の中で実際の I/O が発生するため，計算処理時間を含む Mop/s 値を用いて比較することとした．

4.3.1 BT-IO Simple

Simple での書き込みサブルーチンは 1 timestep ごとに呼び出され，その中で MPI-IO の file view を設定し，3 重ループの中で `MPI_File_write_at()` を呼び出している．Class A での 1 回の書き込み量は固定で 160 Bytes，Class B では 240〜280 Bytes と固定ではないが，どちらの場合も非常に小さい書き込み量である．ROMIO を経由して Catwalk-ROMIO に発行される I/O 要求も書き込みオフセットの値が前後するなど，かなり複雑な書き込みパターンになっている．

図 8 において左側のグラフは Class A，右側のグラフは Class B の結果である．縦軸は BT-IO が出力する Mop/s 値である．この値は，計算量を，ファイルをオープンした直後から計算終了しファイルをクローズし終わるまでに要した時間で割った値である．

Class A では sort 方式を除いて Catwalk-ROMIO の性能（最大 8,928.60 Mop/s）が PVFS2（347.06 Mop/s）を最大で約 25 倍上回り，Class B においても sort 方式ではソートバッファのサイズがある程度大きい場合に，PVFS2 の性能を最大で約 17 倍上回っている（Catwalk-ROMIO の最大 7,609.99 Mop/s，PVFS2 は 430.09 Mop/s）．結合なしの journal 方式と sort 方式の性能が悪いのは，1 回の書き込み量がかなり小さいことによるソートバッファの利用率の低下が原因と考えられる．この状況は，結合処理を導入することでかなり改善されていることが分かる．

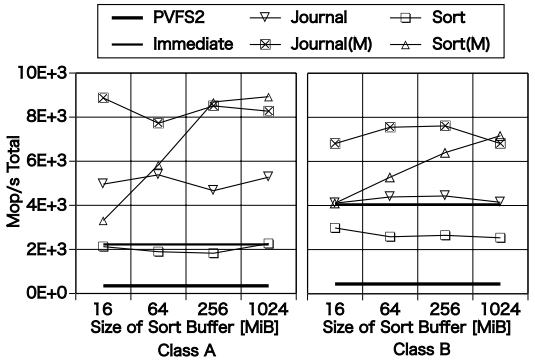


図 8　BT-IO Simple（Ethernet）
Fig. 8　BT-IO Simple (Ethernet).

4.3.2 BT-IO Full

BT-IO full の場合は，ファイルのオープン時に `MPI_Type_struct()` などを用いてストライドパターンを設定した後，timestep ごとに collective I/O の `MPI_File_write_at_all()` 関数を 1 回の呼び出しで書き込んでいる．ROMIO を経由して Catwalk-ROMIO に発行される I/O 要求は，一定の大きさの比較的大きなサイズであり，simple に比べ単純なアクセスパターンになる．

図 9 は先の BT-IO simple と同じ条件で BT-IO full を用いて評価した結果である．今回は PVFS2 の性能がすべてのケースで Catwalk-ROMIO を上回った．Catwalk-ROMIO でソートバッファが小さいときに sort 方式の性能が悪いのは，ソートバッファの溢れが生じ，不連続な書き込みが発生したためである．Class A でソートバッファのサイズが 64 MiB 以上，Class B の 256 MiB 以上では，不連続な書き込みがないことが確認されている．BT-IO full では 1 回の書き込み量が Catwalk-ROMIO の `DATA_MAX` を超えているため，結合処理の有無でも結果はほとんど変わらない．

BT-IO simple の PVFS2 は Catwalk-ROMIO のベストの値に比べ，20 倍ほど遅い結果となっている一方で，BT-IO full の場合は PVFS2 の性能が Catwalk-ROMIO のベストを上回るものの，その差は 2 倍に届かない（Class A の Catwalk-ROMIO の最大は 13,066.3 Mop/s，PVFS2 は 22,032.1 Mop/s，Class B の Catwalk-ROMIO の最大は 15,737.9 Mop/s，PVFS2 は 23,245.7 Mop/s）．

[*1] `README` に記述があり，この値の変更はベンチマークとして許容されている．

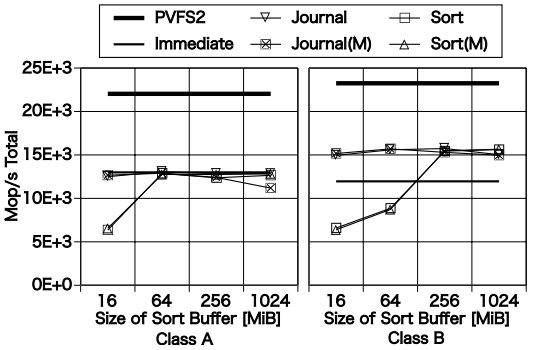


図 9　BT-IO Full（Ethernet）
Fig. 9　BT-IO Full (Ethernet).

4.4　Lustre をファイルサーバとした場合

これまでの評価では Catwalk-ROMIO のファイルサーバは計算ノードのローカルディスクを用いてきた．IOR による計測では，Catwalk-ROMIO のバンド幅が 30〜40 MiB/s と，表 1 に示すディスクの性能に近い値となっている．このことから，ファイルサーバのディスクの性能がボトルネックになっていると考えられる．

ここでは，並列ファイルシステム Lustre[7] を Catwalk-ROMIO のファイルサーバとして用いることで，性能がどのように変化するかを調べる．Catwalk-ROMIO からの書き込みは，Catwalk-ROMIO のサーバ上で逐次的に Lustre ファイルシステムに書き込まれる．このため Lustre は並列ファイルシステムとしてではなく，単純に高性能なファイルシステムという扱いになる．Catwalk-ROMIO が用いるネットワークは Myri10G（IP over Myrinet）を使用する．ここでの Catwalk-ROMIO の評価条件は，結合処理ありの sort 方式，ソートバッファの大きさは 1,024 MiB である．

実験に用いた Lustre は，専用のメタデータサーバ（MDS）1 台，ファイルサーバ（OSS）4 台，それぞれの OSS はディスク 4 台を RAID0 で接続し，計算ノードおよび Lustre のサーバ間を結ぶネットワークは PVFS2 と同じく Myri10G（ただし IP over Myrinet）を使う．ストライピング数の設定は OSS の数に合わせて 4 とした．

図 10 は Catwalk-ROMIO のファイルサーバを Lustre としたときの IOR の性能（図中 “Catwalk-Lustre”）と，比較のために ROMIO の ufs ADIO デバイス経由で Lustre を用いたときのバンド幅（図中 “Lustre”）を示したものである．IOR のパラメータは，図 5 から図 7 と同じに設定した．図中 `TxSz` は TransferSize を，`BlkSz` は BlockSize，`#Seg` は Segment 数をそれぞれ表している．

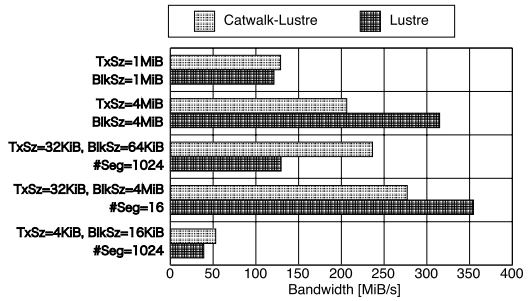


図 10　IOR—Lustre
Fig. 10　IOR – Lustre.

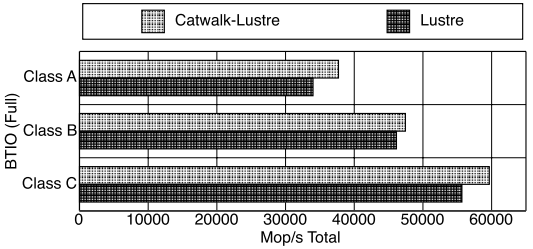


図 11　BT-IO—Lustre
Fig. 11　BT-IO – Lustre.

この図から，Catwalk-ROMIO の最大バンド幅は 277 MiB/s とディスク単体を用いた場合を大きく上回る性能を示していることが分かる．また Lustre を直接用いた場合との比較では，PVSF2 との比較で判明したのと同様の傾向，細粒度の場合に Lustre をサーバとした Catwalk-ROMIO の方が有利，という状況が見られる．

図 11 は BT-IO full において Lustre を Catwalk-ROMIO のファイルサーバとしたときの性能である．IOR の場合と同様，Lustre を BT-IO から直接用いた場合の性能も比較のためにあわせて示す．なお，BT-IO simple の Lustre による計測も試みたが，ROMIO レベルでのエラーが生じたため，BT-IO full の結果のみを示す．

BT-IO full は collective I/O を用いていることもあり，比較的大きな粒度での単純な書

き込みなので Catwalk-ROMIO に不利な状況と考えられるが，図 11 においてはすべてのクラスにおいて Catwalk-ROMIO が Lustre を直接用いた場合を上回る結果となった．

Catwalk-ROMIO の動作は，見方を変えれば，細粒度で複雑なファイル書き込みパターンを，より大きな粒度かつ単純なパターンに変換している，と考えることができる．図 10 および図 11 の結果から，Catwalk-ROMIO は既存のファイルシステムに対し，ファイル書き込みパターンを最適化するフロントエンドとして機能したとも考えられる．

5. 考　　察

本稿における評価の結果から，1 台のファイルサーバしか持たない Catwalk-ROMIO は，複数のファイルサーバから構成される PVFS2 や Lustre に対し，細粒度で複雑な書き込みパターンの場合で優位性を示すことができた．特に BT-IO simple のような細粒度で複雑な書き込みパターンにおいては，Catwalk-ROMIO の性能が大きく上回る結果となった．総じて，Catwalk-ROMIO は書き込みパターンによらず，安定した性能を発揮したが，並列ファイルシステムではその性能は書き込みパターンに大きく依存していることが判明した．

Catwalk-ROMIO の結合処理ありの journal 方式と，ソートバッファサイズを 1,024 MiB としたときの結合処理ありの sort 方式が，様々なファイル書き込みパターンで高い性能を発揮した．この結果から，結合処理が並列ファイルシステムの書き込み性能に与える効果は大きいと考えることができる．また結合処理を効率良く行うためにレコードのソートが必要である．リング通信トポロジを用いることで，書き込み要求を効率良く結合し，シーク頻度を減らし，結果として多数のユーザプロセスからの同時書き込みがあっても，ディスクの単体性能に近い性能を引き出すことができた．換言すれば，すべてのレコードを収めることのできない有限長のソートバッファを用いた不完全なソートであっても，シークの頻度を十分に減らすことができた，と考えることができる．

MPI-IO には atomic mode と呼ばれる，MPI のジョブ全体で 1 つの file pointer をプロセス間で共有し，かつ，その更新処理がアトミックである，というモードがある．残念ながら Catwalk-ROMIO は atomic mode をサポートしていないが，たとえば文献 10) にあるような方法で，Catwalk-ROMIO とは独立に atomic mode を実装する方式が考えられる．

Catwalk-ROMIO では書き込みレコードをソートして書き込み順序を入れ替える．このため書き込みの対象となるファイル領域がオーバラップした場合に，処理順序により，ファイルに書き込まれる結果が異なる可能性がある．基本的に MPI-IO では，このような場合には `MPI_File_flush()` 関数を呼び，オーバラップした複数の書き込みの順序を規定する．Catwalk-ROMIO には `MPI_File_flush()` がすでに実装されている[18]．

しかしながら MPI-IO では，同じプロセスからの書き込みに関しては，書き込み順序が自然に規定されるため，`MPI_File_flush()` を陽に呼ぶ必要がない．Catwalk-ROMIO は，この同一プロセスによるファイルの読み書きに対し，一貫性を保証していない．これに対処するには Catwalk-ROMIO がプロセスからの書き込み要求を受け取った時点で，ローカルなファイルにその内容を書き込んでおく方法が考えられるが，この動作は文献 18) と基本的に同じであり，同一ノード内の複数プロセスによるローカルディスクへの同時書き込みオーバヘッドが回避できない．

本稿では，Catwalk-ROMIO の書き込み性能に焦点をあてて議論したが，Catwalk-ROMIO 自体はそのベースとなった Catwalk[5] はリングトポロジによる読み込み機能も備えているため，リングトポロジとする必要があった．これはリングによるパイプライン転送が効率的であるとの判断による[9]．Catwalk-ROMIO の書き込みにおいては，マージソート的に書き込みレコードをソートし，同時に細かいレコードを結合することが本質であり，ネットワークトポロジをリングではなく分散木構造とすることも可能であると考える．

しかしながら，4.1 節で Catwalk-ROMIO が発揮したバンド幅，30〜40 MiB/s，は表 1 で示したディスクの単体性能に近い値である．さらに，4.4 節で示したように，ファイルサーバとネットワークの性能を上げれば，Catwalk-ROMIO の性能も向上することが確認されている．これらから，単一ファイルサーバの性能ボトルネックとなっていることが分かる．Catwalk-ROMIO のネットワークトポロジを，たとえば分散木構造としても，単一のファイルサーバがボトルネックであるという状況に変わりはなく，結果に根本的な差は生じないものと考える．したがって，大規模なクラスタなどで，単一ファイルサーバの性能を大きく上回る I/O 性能が要求される場合には，複数サーバへの対応が不可欠である．

Catwalk-ROMIO は単一ファイルサーバというボトルネックがあるにもかかわらず，そのようなボトルネックがないはずの並列ファイルシステムに比肩できる性能を発揮することが確認された．Catwalk-ROMIO が並列プログラムが生成する並列アクセスパターンを，ディスクの特性に適したアクセスパターンに変換したと考えることもできる．従来の並列ファイルシステムには，アクセスパターンの変換という視点が欠けていると考えられる．

6. 関 連 研 究

Data Sieving[15] は並列 I/O 特有の大量の細かい不連続なファイルアクセス要求を，それらを包含するような大きな単位でファイルにアクセスし，要求があった部分のデータのみ

をユーザプロセスとやりとりする技法である．この方法は読み込みに際しては有効な方法であるが，書き込みにおいては read-modify-write の処理や，他のプロセスの書き込みとの調停が必要となる．

Two-Phase I/O[4] は書き込みを 2 つのフェーズに分ける．最初のフェーズで I/O 要求を collective の対象となるプロセス数よりも少ないプロセス群に集約し，そこで I/O 要求を整理し，次のフェーズで整理された要求をディスクに発行する方式である．

ROMIO の collective な I/O にはすでに Two-Phase I/O[4] および Data Sieving[15] による最適化が施されている[15]．しかしながら，たとえば図 11 の結果から，Catwalk-ROMIO の技法がこれらに匹敵する効果をあげていると考えることができる．Catwalk-ROMIO で提案する技法は，I/O 要求を整理するにあたり，明確なフェーズに分割せず，パイプライン的に処理するという点で Two-Phase I/O と大きく異なっている．また Catwalk-ROMIO の技法は collective，non-collective の区別なく適用可能であり，collective I/O を行う BT-IO full による評価結果（図 9 および図 11）から，collective I/O でも効果があると考えられる．

FORTRAN で記述されている BT-IO の書き込みファイルをオープンする `setup_btio` サブルーチンは，simple の場合で 35 行しかないのに対し，full の場合では MPI-IO の file view の設定など，それなりに複雑な記述が必要であり 160 行もある．このことから，性能を引き出すために最適化が容易な collective I/O を使えばよいという主張は，プログラミングの容易さという観点が抜けていると考える．

ファイルの書き込みをローカルなディスクにジャーナルとして記録するという方式は PLFS[1] でも提案されており，その有効性が確認されている．しかしながら，ローカルなジャーナルファイルの生成は，ノード内の複数のプロセスがそれぞれのジャーナルファイルに同時に書き込むため，結果的にファイル間を行き交うシークが発生し，ノード内のプロセス数が増えるに従い性能が低下する可能性がある．この現象は，最初の Catwalk-ROMIO[18] ですでに確認されている．本稿で提案するようにローカルでプロセスごとに複数のファイルを生成せずに，1 つのファイルにまとめる方法，あるいは Catwalk-ROMIO の sort 方式のようにファイルを生成しない方式を用いることで，ローカルディスクに対するシークの問題を回避できる．

リングに沿ったパイプライン転送という方式としては Dolly[9] がある．Dolly の目的はファイルサーバにあるファイルを複数の計算ノードのローカルディスクにパイプライン転送でコピーすることであり，これは Catwalk[5),19] および Catwalk-ROMIO[18] の読み込みでも採用されている方式である．しかしながら本稿の主題である並列ファイル書き込みでは，ファイルサーバのファイルに書き込むため，データの流れの向きが逆である．また Catwalk-ROMIO のパイプライン処理はデータ転送よりもレコードのソートや結合処理により大きな意味があると考える．

ま と め

並列ファイルのアクセスは細粒度で不連続な場合があり，それが並列ファイルアクセスの性能低下を招くことは比較的古くから知られている問題である．この問題に対処するために，本稿では通信トポロジに基づいたパイプライン処理により，この問題を回避する技法を提案した．提案技法は MPI-IO ROMIO の ADIO デバイスとして実装され，並列ファイルベンチマークプログラム IOR と BT-IO を用いて評価され，代表的な並列ファイルシステムである PVFS2 と Lustre により性能が比較された．

評価結果から，細粒度かつ複雑なファイル書き込みパターンにおいては，10 Gb/s の Myri10 G で接続された複数のファイルサーバを用いた並列ファイルシステムに対し，1 Gb/s の Ethernet で接続された 1 台のファイルサーバしか用いない提案技法の方が，高い性能を発揮することが示された．また，より高速なネットワークおよび高速なファイルシステムを用いることで，Catwalk-ROMIO の性能も並列ファイルシステムに比肩できる性能を発揮することが判明した．同時に Catwalk-ROMIO が並列ファイルシステムに対するフロントエンドとしても有効であることが確認された．

これらの結果から，並列アクセスパターンを適切に変換することでより高速な並列 I/O が可能になることが判明した．この成果をもとに，今後は通信トポロジを改良し，単一ファイルサーバのボトルネックを回避すると同時に，よりスケーラブルな並列 I/O を実現する手法の研究を進めたいと考える．

小規模のクラスタにおいて，Catwalk-ROMIO は費用対効果の高い MPI-IO システムが構築可能であることが示された．特に専用の並列ファイルシステムを持つことが少ない小規模のクラスタにおいては，Catwalk-ROMIO は有効と考える．Catwalk-ROMIO は既存の（並列）ファイルシステムと共存可能であり，アプリケーションの IO パターンに応じて，並列ファイルシステムと Catwalk-ROMIO を使い分けることも可能である．

Catwalk-ROMIO は SCore[8] の最新バージョンに組み込まれオープンソースとして公開されている．

謝辞　本研究の一部は文部科学省「e-サイエンス実現のためのシステム統合・連携ソフトウェアの研究開発」からの支援を受けている．

参 考 文 献

1) Bent, J., Gibson, G., Grider, G., McClelland, B., Nowoczynski, P., Nunez, J., Polte, M. and Wingate, M.: PLFS: A checkpoint filesystem for parallel applications, *SC '09: Proc. Conference on High Performance Computing Networking, Storage and Analysis*, New York, NY, USA, ACM, pp.1–12 (2009).
2) Carns, P., Lang, S., Ross, R., Vilayannur, M., Kunkel, J. and Ludwig, T.: Small File Access in Parallel File Systems, IEEE Computer Society (2009).
3) Carns, P.H., III, W.B.L., Ross, R.B. and Thakur, R.: PVFS: A parallel file system for linux clusters, *Proc. 4th Annual Linux Showcase and Conference*, USENIX Association, pp.317–327 (2000).
4) del Rosario, J.M., Bordawekar, R. and Choudhary, A.: Improved parallel I/O via a two-phase run-time access strategy, *SIGARCH Comput. Archit. News*, Vol.21, No.5, pp.31–38 (1993).
5) Hori, A., Kamoshida, Y., Matsuba, H., Ohta, K., Yasui, T., Sumimoto, S. and Ishikawa, Y.: On-Demand File Staging System for Linux Clusters, *Proc. IEEE Cluster Conference* (2009).
6) Kotz, D. and Nieuwejaar, N.: Dynamic File-Access Characteristics of a Production Parallel Scientific Workload, Technical Report PCS-TR94-211, Dartmouth College (1994).
7) ORACLE: Lustre. http://www.lustre.org/
8) PC Cluster Consortium: SCore. http://www.pccluster.org/
9) Rauch, F., Kurmann, C. and Stricker, T.: Partition Cast - Modelling and Optimizing the Distribution of Large Data Sets in PC Clusters, *Euro-Par 2000 – Parallel Processing*, Bode, A. and Ludwig, T. (Eds.), Munich, Germany, Springer (2000).
10) Ross, R., Latham, R., Gropp, W., Thakur, R. and Toonen, B.: Implementing MPI-IO atomic mode without file system support, *Cluster Computing and the Grid, IEEE International Symposium on*, Vol.2, pp.1135–1142 (2005).
11) Russell Coker: Bonnie++. http://www.coker.com.au/bonnie++/
12) Shan, H. and Shalf, J.: Using IOR to analyze the I/O Performance for HPC Platforms, Technical Report LBNL-62647, Lawrence Berkeley National Laboratory (2007).
13) T2K Open Supercomputer Alliance: T2K. http://www.open-supercomputer.org/
14) Thakur, R., Gropp, W. and Lusk, E.: An Abstract-Device Interface for Implementing Portable Parallel-I/O Interfaces, Technical report (1996).
15) Thakur, R., Gropp, W. and Lusk, E.: Data Sieving and Collective I/O in ROMIO, *Symposium on the Frontiers of Massively Parallel Computation*, p.182 (1999).
16) Thakur, R., Lusk, E. and Gropp, W.: Users Guide for ROMIO: A High-Performance, Portable MPI-IO Implementation, Technical Memorandum ANL/MCS-TM-234, Argonne National Laboratory (2004).
17) Wong, P. and der Wijngaart, R.F.V.: NAS Parallel Benchmarks I/O Version 2.4, Technical Report NAS-03-002, NASA Ames Research Center (2003).
18) 堀　敦史，鴨志田良和，松葉浩也，安井　隆，住元真司，石川　裕：ファイルステージングシステム Catwalk の MPI-IO 実装，研究報告「ハイパフォーマンスコンピューティング（HPC）」，No.2009-HPC-121, 情報処理学会 (2009).
19) 堀　敦史，鴨志田良和，松葉浩也，安井　隆，住元真司，石川　裕：ファイルステージング再考：オンデマンド化と高速化に向けたプロトタイプ実装の評価，研究報告「ハイパフォーマンスコンピューティング（HPC）」，No.2009-HPC-119, pp.38–42, 情報処理学会 (2009).

(平成 22 年 5 月 7 日受付)
(平成 22 年 9 月 6 日採録)

堀　　敦史（正会員）

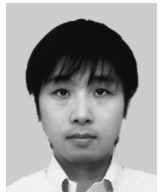
1979 年早稲田大学理工学部電気工学科卒業．1981 年早稲田大学大学院理工学研究科修士課程修了．同年株式会社三菱総合研究所に入社．1992 年技術研究組合新情報処理開発機構に出向．1999 年東京大学より博士（工学）の学位を取得．2001 年株式会社スイミーソフトウェア設立．2004 年英国 Allinea Software 社に移籍．2008 年東京大学情報基盤センター特任教授．2010 年より理化学研究所計算科学研究機構．研究員．並列システムソフトウェアの研究に興味を持つ．

太田　一樹

2008 年東京大学理学部情報工学科卒業．2010 年東京大学大学院情報理工学系研究科コンピューター科学修士課程修了．現在は Preferred Infrastructure, Inc.，最高技術責任者．大規模データ処理のための分散システム等に興味を持つ．

安井　　隆（正会員）

2001 年岡山大学工学部情報工学科卒業．2003 年岡山大学大学院自然科学研究科博士前期課程電子情報システム工学専攻修了．同年 4 月株式会社日立製作所中央研究所入社．エンタプライズサーバのチップセット開発等に従事．



鴨志田良和（正会員）

1997 年東京大学理学部情報科学科を卒業．同年日本オラクル株式会社に入社．2003 年東京大学大学院情報理工学系研究科電子情報学専攻修士課程に入学，2008 年同専攻博士課程を退学．同年より東京大学情報基盤センター特任助教．高性能コンピュータシステム，クラスタ・グリッド・クラウドシステムソフトウエアに興味を持つ．

松葉　浩也（正会員）

2003 年東京大学理学部情報科学科卒業．2005 年東京大学大学院情報理工学系研究科コンピュータ科学専攻修士課程修了，同博士課程進学．2006 年東京大学情報基盤センター助手．2010 年東京大学大学院情報理工学系研究科にて博士（情報理工学）の学位を取得．2010 年 4 月より株式会社日立製作所中央研究所．研究員．並列計算機のシステムソフトウェアに興味を持つ．

住元　真司（正会員）

1986 年同志社大学工学部電子工学科卒業．同年株式会社富士通入社．株式会社富士通研究所にて並列オペレーティングシステム，並列分散システムソフトウェアの研究開発に従事．1997 年より新情報処理開発機構に出向．高性能クラスタ向け高速通信機構の研究開発に従事．2002 年より株式会社富士通研究所にて高性能通信機構の研究開発，大規模 PC クラスタ開発に従事．超大規模並列分散システムのアーキテクチャ，システムソフトウェア等に興味を持つ．平成 12 年度情報処理学会論文賞受賞．工学博士（慶応義塾大学大学院理工学研究科）．

石川　　裕（正会員）

1982 年慶応義塾大学工学部電気工学科卒業．1987 年慶応義塾大学大学院工学研究科電気工学専攻博士課程修了．工学博士．同年電子技術総合研究所入所．1993 年技術研究組合新情報処理開発機構出向．2002 年より東京大学大学院情報理工学系研究科コンピュータ科学専攻．教授．次世代高性能コンピュータシステム，クラスタ・グリッドシステムソフトウェア，高信頼システムソフトウェア開発技術，実時間処理等に興味を持つ．