情報処理学会論文誌 Vol. 52 No. 12 3239–3250 (Dec. 2011)

MPI-IO/Gfarm：分散ファイルシステムGfarmのためのMPI-IOの実装と評価

木村浩希[†1,†2] 建部修見[†1,†2]

本稿では，分散ファイルシステムGfarmのためのMPI-IO実装，MPI-IO/Gfarmについて述べる．Gfarmファイルシステムは広域にまたがる複数のクラスタ内の計算ノードのストレージをまとめ1つのネームスペースで管理することを可能にする．Gfarmファイルシステムは，計算ノードのローカルストレージの利用，ファイル複製の利用によりスケーラブルなファイルアクセス性能を実現している．しかし，MPI-IOにおける典型的なアクセスである複数のプロセスが単一のファイルに対して書き込みを行う場合においては，性能はスケールしない．本稿では以上の場合の性能改善を主な目的とし，最適化手法の提案と実装について述べる．性能評価では，IOR，HPIO，BTIOを用いた評価を行い，PVFS2，NFSとの比較を行った．提案手法を用いることでノード数に対してスケーラブルな結果が得られ，PVFS2との比較においても2倍から3倍程度MPI-IO/Gfarmが良い性能を示した．

MPI-IO/Gfarm: An Implementation and Evaluation of MPI-IO for the Gfarm File System

Hiroki Kimura[†1,†2] and Osamu Tatebe[†1,†2]

This paper proposes a design and implementation of an MPI-IO implementation of the Gfarm file system, called MPIO/Gfarm. The Gfarm file system is a global file system that federates the local storage of compute nodes among several clusters. It has a scale-out architecture designed to support distributed data-intensive computing. However Gfarm file system does not achieve scalable performance in the case of parallel writes to a single file, a typical file operation in MPI-IO. This paper proposes an optimization technique to improve the parallel write performance to a single file. In the evaluation using BT- IO, IOR and HPIO, MPI-IO/Gfarm achieves scalable parallel I/O performance. Compared with MPI-IO on PVFS2 and NFS, it achieves at most a 1.5x speedup of write performance and 2.0x to 3.0x speedup of read performance using seven storage nodes.

1. はじめに

近年，科学技術計算分野におけるデータの大規模化が進んでいる．扱うデータサイズはエクサバイト，さらにはゼタバイトスケールになっていくことが考えられる．このような大規模なデータを単一の拠点で管理することは信頼性の観点からも現実的ではなく，複数の拠点で分散し管理する必要がある．このような場合，広域にまたがる拠点間で効率的に大規模なデータ管理を行う必要があり，Gfarm[1)]のような分散ファイルシステムが必須のものである．これまでに，天文分野および素粒子物理学の分野においてGfarmを用いた大規模データ解析に関する研究が行われている[2),3)]．

科学技術計算における並列ファイルアクセスは標準インタフェースとしてMPI-IO[4)]が用いられている．現在までに，PVFS2，Lustre，PanFS，GPFSなどのファイルシステムへの実装があり，最適化に関する研究がなされている[5)–7)]．MPI-IOでは複数のプロセスが単一のファイルに対してアクセスを行う場合の最適化を主な目的としており，そのための重要な特徴としてFileViewがある．FileViewはプロセスがファイルに対してアクセス可能なアクセス範囲の定義である．MPI-IOではFileViewの定義を実際のファイルアクセスを行う前に行うことで，アクセス範囲の情報をプロセス間で交換し効率的なファイルアクセスを行うようなCollective I/Oと呼ばれる最適化を行うことができる．

Gfarm上で効率的にデータ解析をするために，Gfarmのための MPI-IOが必要であると考えられる．本稿ではMPI-IO/Gfarmと呼ばれるGfarmに最適化されたMPI-IOの設計と実装について述べる．Gfarmにおける並列ファイルアクセスは，複数のプロセスが個別のファイルに書き込みを行う場合においてはスケーラブルな書き込み性能を得ることができる．また読み込みについても，ファイルのローカリティを考慮したプロセスのマッピングを行い，ファイル複製を有効に活用することで複数のプロセスが単一のファイルにアクセスする場合と，またそれぞれ個別のファイルにアクセスする場合の両方においてスケーラブルな性能を実現できる．しかし，複数のプロセスが単一のファイルに書き込みを行う場合においてはすべてのプロセスが単一のストレージにアクセスし性能がスケールしない問題点がある．本稿では，以上の問題点を解決し複数のプロセスが単一のファイルに対してアクセスす

†1 筑波大学大学院システム情報工学研究科
Graduate School of Systems and Information Engineering, University of Tsukuba
†2 独立行政法人科学技術振興機構 CREST
Japan Science and Technology Agency, CREST

る場合においてもスケーラブルな書き込み性能を得られる手法を提案し，実装について述べる．そして，IOR，HPIO，BTIO の 3 つの並列ファイルアクセスのベンチマークを用いて性能評価を行う．また代表的なクラスタファイルシステムである PVFS2 と NFS との性能比較を行う．本稿の貢献を以下に示す．

- 本稿では，Gfarm に最適化された MPI-IO である MPI-IO/Gfarm の設計と実装について述べる．“N-1” パターンの書き込み性能を改善するため，提案手法では “N-1” パターンのアクセスを “N-N” パターンに置換えを行う．提案手法を用いることによって，“N-1” パターンの書き込みにおいてもスケーラブルな性能を実現することができ，Gfarm 上で効率的なデータ解析を可能にする．
- 提案手法を IOR，HPIO，BTIO のベンチマークを用いて評価を行った．“N-1” パターンの書き込みを行った結果，すべての評価でノード数に対してスケーラブルな性能を示し本手法の有効性を示した．また PVFS2，NFS との性能比較を行った結果，すべての評価で MPI-IO/Gfarm が良い性能を示した．

2. MPI-IO

MPI-IO は MPI-2[8] によって規定された並列ファイルアクセスの標準インタフェースである．MPI-IO の重要な特徴の 1 つは FileView である．FileView はプロセスがファイルに対してアクセス可能な範囲の定義を行うものである．MPI-IO ではファイルのオープンを行った後，実際の読み書きを行う前に FileView の定義が行われる．FileView を事前に定義することで Collective I/O と呼ばれる最適化が可能となる．Collective I/O はプロセス間で協調したファイルアクセスを行うことで効率化を図るものである．MPI-IO では，ファイルアクセスに先立ってまず FileView をプロセス間で共有し，それぞれのプロセスができるだけ連続した領域に対してファイルアクセスを行えるように I/O リクエストの最適化を行う．MPI-IO ではファイルのオープン（MPI_File_open）を行った後，FileView の定義（MPI_File_set_view）を行う．ここではそれぞれのプロセスがファイルに対してアクセス可能な範囲の定義を行う．FileView の定義は，変位と要素データ型とファイル型の 3 つを指定する．変位は読み書きを行う先頭部分を指定するもので，ヘッダ部分を飛ばしてアクセスを行う場合に用いられる．要素データ型はアクセスの基本単位を指定するもので，MPI-IO によるファイルアクセスでは要素データ型単位で読み書きが行われる．ファイル型は 1 つ以上の要素データ型からなっていて，ファイルに対してプロセスがアクセス可能な範囲を定義するものである．要素データ型，ファイル型は MPI における基本データ型，もしくは

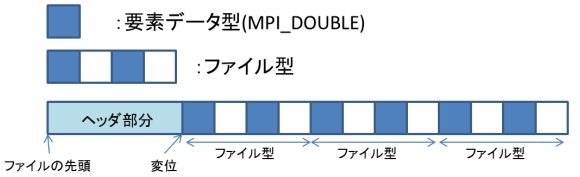


図 1　FileView の例
Fig. 1　FileView example.

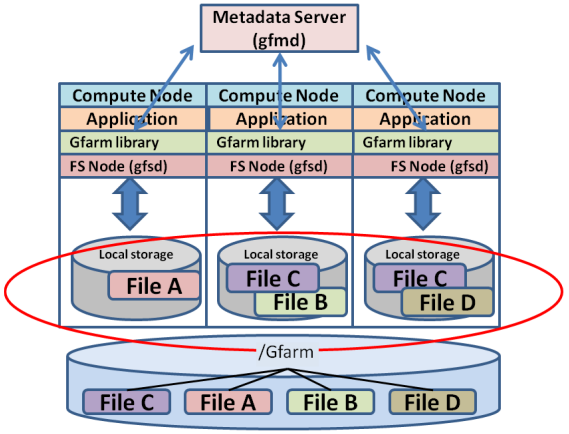
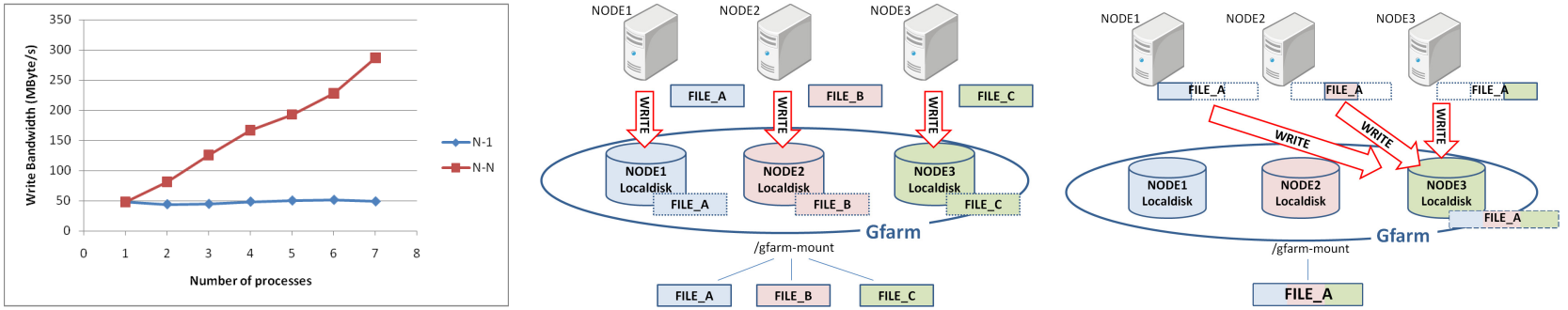


図 2　Gfarm の概要
Fig. 2　Outline of the Gfarm file system.

複数の基本データ型からなる派生データ型を用いて定義される．FileView の定義は，オープン時に指定されたコミュニケータ内のプロセスがすべて呼び出す必要があり，要素データ型はすべてのプロセス間で共通している必要がある．FileView の例を**図 1** に示す．例では MPI_DOUBLE 型を要素データ型として，MPI_DOUBLE 型のアクセス範囲を 2 つ持つファイル型を定義している．空白部分はアクセスを行わない範囲を示している．実際のファイルアクセス（MPI_file_write/read）が行われるとき，定義された FileView に従ってアクセス可能範囲に対して読み書きが行われる．

3. Gfarm ファイルシステム

Gfarm ファイルシステムは，筑波大学と産総研で開発が進められている OSS の分散ファ



(a) Gfarm における並列ファイルアクセス性能　(b) Gfarm における N-N パターンの書き込み　(c) Gfarm における N-1 パターンの書き込み

図 3　Gfarm における並列ファイルアクセス
Fig. 3　Parallel write performance of the Gfarm file system.

イルシステムである．Gfarm ファイルシステムは，複数の計算ノードのローカルストレージをまとめ，単一のネームスペースで管理することを可能にしている．**図 2** に Gfarm ファイルシステムの概要を示す．Gfarm ファイルシステムは 1 つのメタデータサーバと複数のファイルサーバからなる．gfmd と呼ばれるメタデータサーバは，ファイルの属性，ディレクトリ構造，ファイル複製の管理を行っている．gfsd と呼ばれるファイルシステムサーバは，基本的に計算ノード上で動作することを前提としておりローカルストレージに対するファイルアクセス，およびリモートノードの gfsd を介しリモートストレージに対してアクセスを行う．Gfarm ファイルシステムでは，ファイルアクセスを行うファイルシステムノードを RTT や負荷状況を用いて選択する．基本的に，自ノードのストレージに空き容量がない場合を除き，ローカルストレージが選択され，優先的に使用される．そのため新規ファイルの作成時には create を発行したプロセスが動作するノードのローカルストレージ上にファイルの実体が作成される．この場合，Gfarm ファイルシステムではファイルのストライピングを行っていないため 1 つのファイルは 1 つのファイルとしてストレージに保存されている．ファイル複製がある場合には，RTT と負荷状況から最適なファイル複製が選択されファイルアクセスが行われる．新規ファイル作成時と同様に，ローカルストレージにファイルの実体が存在する場合はローカルストレージが優先的に使用される．以上のような，ローカルストレージ，ファイル複製の活用によって Gfarm ファイルシステムはスケーラブルな性能を実現している．Gfarm ファイルシステムへのアクセスは Gfarm ライブラリを通したアクセス，もしくは gfarm2fs を用いて FUSE[9] を通したアクセスが可能である．

3.1　Gfarm における並列ファイルアクセス

並列ファイルアクセスは主に 2 つに分類できる．複数のプロセスが個別のファイルに対してアクセスを行う “N-N” パターンと複数のプロセスが 1 つのファイルに対してアクセスを行う “N-1” パターンの 2 つである．以上の 2 つのパターンで Gfarm ファイルシステムに対する評価を行った結果を**図 3** (a) に示す．1 プロセスあたり 512 MB のデータを 2 つのパターンで書き込みを行った結果である．1 ノードあたり 1 プロセスを動作させ，ノード数を増加させた．書き込みが終了した後 sync 処理を行っている．

“N-N” パターンの場合書き込み性能がスケールしている．この場合のアクセスについて図 3 (b) に概要を示す．Gfarm ファイルシステムではそれぞれのプロセスが動作するローカルストレージを優先的に使用するためファイルの実体はプロセスの動作するノードのローカルストレージ上に作成される．そのためそれぞれのプロセスの書き込み処理はローカルストレージに対して行われ，処理が分散され性能がスケールする．一方 “N-1” パターンの場合においては書き込み性能は一定となっている．この場合のアクセスについて図 3 (c) に概要を示す．Gfarm ファイルシステムではファイルの分割を行っていないため，ファイルの実体は create を発行した単一のノードのストレージ上に作成される．そのためすべてのプロセスはファイルの実体が存在するストレージに対しファイルアクセスを行い単一のストレージ性能に制限され性能は一定となっている．

“N-1” パターンは並列ファイルアクセスで一般的に行われるアクセスパターンであり，この場合でも性能がスケールするための最適化が必要であると考えられる．この “N-1” パターンの性能がスケールしないという問題は他の並列ファイルシステムでも報告されており，最適化に関する研究がなされている[10]．

4. MPI-IO/Gfarm の設計

基本的には MPI-IO/Gfarm は，“N-1” パターンのアクセスを “N-N” パターンに置き換えを行う．そしてアプリケーションからは透過的にアクセスを可能にするための仕組みを用意する．MPI-IO/Gfarm の設計の概要について**図 4** に示す．図の上部 (1) は，それぞれのプロセスの持つ FileView を現しており，色のついたブロックがプロセスのアクセス可能な範囲を示している．MPI-IO/Gfarm の層 (2) では 1 つのファイルとしてみたときのアクセスパターンが示されている．(3) のブロックはそれぞれ図上部 (1) の FileView のアクセス可能範囲と対応しており，ブロックの並びは 1 つのファイルを表している．個別に作成されたファイルは Gfarm ファイルシステムによってすべてのノードでアクセス可能である．

まず MPI-IO/Gfarm では，“N-1” パターンの書き込みが行われる場合に，プロセスごとにそれぞれ個別のファイルを作成する．このとき Gfarm ファイルシステムでは，それぞれのプロセスの動作するローカルストレージにファイルの実体が作成されることが期待できる．それぞれのプロセスは自身のアクセス可能範囲のデータを個別のファイルに対して書き込みを行う．

アプリケーション透過にするため，MPI-IO/Gfarm では FileView を用いる．FileView の情報を位置情報ファイルとして保存しておき，単一ファイルのオフセット（グローバルオフセット）から個別ファイルのオフセット（ローカルオフセット）へ自動的に変換を行う．位置情報ファイルは，FileView の定義を行うときにのみ作成する．Log-structured ファイルシステムのように，I/O オペレーションごとにオフセットと長さを保存しない．そのためメタデータ管理に関するコストを少なくできると考えられる．

MPI-IO/Gfarm では FileView の定義が行わることを前提としている．またデフォルトで設定されているようなすべてのプロセスが同一のアクセス可能範囲を持つ場合は想定していない．この場合，本手法は適応されないが，ファイルアクセスは可能である．複数のプロセスが同一のアクセス可能範囲を持っていた場合，本手法をそのまま適応した場合に重複箇所のファイルの実体が複数存在してしまう．そのため，MPI-IO/Gfarm では重複範囲のデータを 1 つに制限しファイル実体を代表プロセスが担当する．

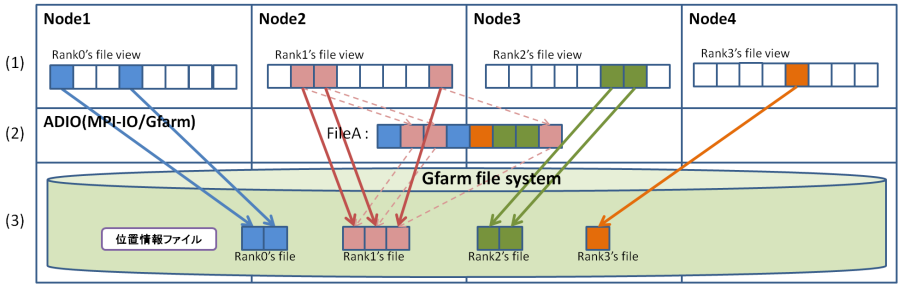


図 4　MPI-IO/Gfarm の設計概要
Fig. 4　Design of MPI-IO/Gfarm.

提案手法についてまとめる．

- “N-1” アクセスを “N-N” アクセスに置き換えることで，“N-N” アクセスと同様に書き込み処理がそれぞれのプロセスが動作するノードのローカルストレージに分散することが期待でき，スケーラブルな書き込み性能を得ることができると考えられる．
- MPI-IO/Gfarm では FileView の情報を用いて位置情報ファイルを作成する．位置情報ファイルを用いて本来のファイルと分散ファイルのオフセットの変換を行うことでアプリケーション透過なアクセスを可能にする．位置情報ファイルは MPI_File_set_view を用いて FileView の定義が行われるときに作成される．

5. MPI-IO/Gfarm の実装

我々は，MPI-IO/Gfarm を MPI-IO の実装の 1 つである ROMIO[11] の持つ ADIO[12] プラグインとして実装した．ROMIO では様々なファイルシステムに対応するため，ADIO と呼ばれるファイルシステムを抽象化する仕組みを用いている．MPI-IO/Gfarm の位置情報管理の実装，および実装したファイルアクセス関数の詳細について以下に示す．

5.1 位置情報ファイル

位置情報ファイルは，分散書き込みされたファイルが本来のファイルのどのオフセットに位置するデータであるのかを示すファイルである．MPI-IO/Gfarm ではファイルを新規に作成する場合の MPI_File_set_view 時に，指定された FileView の情報を元に位置情報ファイルを作成する．位置情報ファイルは，プロセスごとに 1 つのファイルを作成し，自プロセスの情報を書き込む．位置情報ファイルの例を**図 5** と**表 1** に示し，それぞれのパラメータについて以下で示す．図 5 では，3 つのプロセスがあり，1 つのファイルに対するアクセス

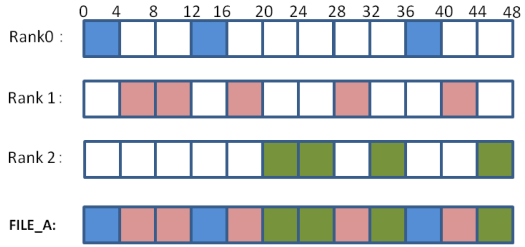


図 5　アクセスパターンの例
Fig. 5　Access pattern example.

表 1　位置情報ファイルの例
Table 1　Location information file example in the three processes case.

nprocs	3		
	Rank0	Rank1	Rank2
arr_len	4	6	4
disp	0	0	0
filetype_size	12	20	16
filetype_extent	48	48	48
blocklens[]	{4,4,4,0}	{0,8,4,4,4,0}	{0,8,4,4}
indices[]	{0,12,36,48}	{0,4,16,28,40,48}	{0,20,32,44}

範囲が示されている．それぞれ色のついた部分がアクセス可能範囲を示している．

- nprocs
  ノード内の分散書き込み時のプロセス数.
- arr_lens
  それぞれのプロセスの “blocklens[]”，“indices[]” のサイズ.
- disp
  FileView の開始位置を示す.
- filetype_size
  FileView の示すアクセス範囲のうち，実際にアクセス可能である部分のサイズを示す.
- filetype_extent
  FileView の示すアクセス範囲全体のサイズを示し，アクセスを行わない部分を含めたサイズを示す.
- blocklens
  アクセス可能部分の連続部分のサイズを示す．FileView の先頭もしくは終端がアクセスを行わない部分である場合 “0” が入る.
- indices
  “blocklen” で示す連続ブロックの開始位置を示す.

5.2　ファイルのオープン

MPI_File_open に対応する．新規ファイルにアクセスする場合には，MPI_info を用いて提案手法を用いて分散書き込みを行うかどうか指定することができる．もし提案手法を用いて分散書き込みされているファイルに対してアクセスを行う場合ここではファイルのオープンを行わず，分散ファイルに対してアクセスを行うフラグをセットするのみである．

5.3　FileView の設定

MPI_File_set_view に対応する．

5.3.1　新規ファイルに対するアクセス

まず分散ファイルを格納するためのディレクトリの作成を行う．ファイル名と同じ名前のディレクトリを作成する．このディレクトリが MPI-IO/Gfarm を用いて分散書き込みされたことを示すフラグを Gfarm の拡張属性を用いてディレクトリのメタデータに保存する．そして指定された FileView を元に位置情報ファイルの作成を行う．位置情報ファイルはノードあたり 1 つ作成され，ノード内のプロセスの FileView の情報を格納する．このとき，異なるプロセスが同じアクセス範囲を指定した場合，ファイルの同じオフセットに位置するデータが異なるファイルに同時に存在してしまう．これを回避するために，いったんすべての FileView の情報を MPI_Allgather を用いて書き込みを行うコミュニケータ内で共有し，同じアクセス範囲を指定しているプロセスがあるか判別を行う．同じアクセス範囲を含むプロセスが存在した場合，ファイルを 1 つに限定するためランク番号の一番小さいプロセスが代表しファイルの実体を持つ．代表プロセス以外のプロセスは自分の FileView から同一のアクセス範囲を除いて，位置情報ファイルに書き込みを行う.

その後，プロセスごとに個別のファイルを create する．Gfarm では，ストレージの空き容量がない場合を除いて，ほとんどの場合ローカルストレージにファイルの実体が作成される．そのため，そのプロセスが動作するノードのローカルストレージに対して書き込みを分散させることができる.

5.3.2　既存のファイルに対するアクセス

分散ファイルに対するアクセスの場合は，すべてのプロセスがすべての位置情報ファイル

を読み込む．現在指定されている FileView の情報と読み込みを行った位置情報ファイルの情報とを比較し，自プロセスがアクセスするファイルの判別を行い，必要な分散ファイルのオープンを行う．

5.4　書き込みと読み込み処理

MPI_File_write，MPI_File_write_at と MPI_File_read，MPI_File_read_at に対応する処理である．共有ファイルポインタを用いたアクセスの場合，すべてのプロセスが同じ FileView を指定する必要があるため，本手法は有効にならない．また，非同期 I/O と Collective I/O については未実装である．

読み書きを行うとき，まず現在指定されている FileView の情報から元々の 1 つのファイルでのオフセット（グローバルオフセット）を求める．この処理は ADIO 内ですでに用意されておりそれを用いている．その後，本手法ではグローバルオフセットから分散ファイルのオフセット（ローカルオフセット）を求める．

図 6 にグローバルオフセットからローカルオフセットを求める処理を示す．入力としてグローバルオフセットと *location_info* が渡される．*location_info* は位置情報ファイルの情報が格納されており，図 1 で示したパラメータが格納されている．出力として，ローカルオフセットと *count* を出力する．*count* は，求められたローカルオフセットから連続してアクセスできる長さを示しており，元ファイル上で，*count* の示す長さは連続していることを表している．

(1)　値の初期化を行う．*ft_count* は FileView を何度繰り返したかを表す値で，*offset* は，FileView でのオフセットを表す．

(2)　現在のグローバルオフセットが含まれるブロックの検索を行う．(1) で求めた *offset* を含むブロックを検索する．

(3)　アクセス範囲を含むブロックが存在する場合には，ローカルオフセットと *count* を求める．

新規ファイルに対して書き込みを行う場合，プロセス間で重複するアクセス範囲を持っている場合を除いて，それぞれのプロセスは自分が create したファイルに対して書き込みを行う．このとき，分散ファイルに対して，元ファイルでのオフセットが単調増加するように書き込みを行う．プロセスが非連続なアクセスを行う場合においても，分散ファイルに対して連続して書き込みを行うことができる．

5.5　ファイルのクローズとシンク

現在オープンしているすべての分散ファイルに対して，クローズ，シンクを行う．

```
INPUT : global_offset, location_info
OUTPUT : local_offset, count
1) initialize variables
ft_count = global_offset / filetype_extent
offset = global_offset - ft_count * filetype_extent
bl_sum = 0
flag = 0
2) search the block including the offset
for i = 0 to arr_lens do
  if blocklens[i] = 0 then
    continue
  else if offset = indices[i] then
    flag = 1
    break
  else if offset > indices[i] and offset < indices[i] + blocklens[i] then
    bl_sum += offset - indices[i]
    flag = 1
    break
  else
    bl_sum += blocklens[i]
  end if
end for
3) calculate the local offset and count
if flag = 1 then
  local_offset = filetype_size * ft_count + bl_sum
  count = blocklens[i] - (offset - indices[i])
else
  local_offset = -1
  count = 0
end if
```

図 6　グローバルオフセットからローカルオフセットへの変換
Fig. 6　Converting a global offset into a local offset.

6. mpiio-gfarm2fs

提案手法を用いて出力されたデータは，POSIX 準拠のアプリケーションでは 1 つのファイルとしてアクセスできないという問題がある．そこで gfarm2fs を改変し，提案手法で出力された分散ファイルに対して POSIX/API でのアクセスを可能にする mpiio-gfarm2fs を作成した．mpiio-gfarm2fs では，set_view 時に設定された拡張属性を目印にし，分散ファイルが格納されているディレクトリを単一ファイルとして扱うことを可能にする．ファイルのオープン時に位置情報ファイルを読み込み自動的にグローバルオフセットをローカルオフセットに変換しファイルアクセスを行う．mpiio-gfarm2fs を用いてマウントすることで，既存の POSIX 準拠のアプリケーションでアクセスが可能となる．

7. 性能評価

7.1 評価環境

MPI-IO/Gfarm の性能評価を Intrigger プラットフォームのつくばサイトを用いて行う．また，代表的なクラスタファイルシステムである PVFS2 と NFS との性能比較を行う．評価環境について**表 2** に示す．PVFS2 のストライプサイズは 64 KB に設定した．また PVFS2 のメタデータサーバはファイルサーバと同じ数，同じノード上で動作させる．すべての評価で，PVFS2 に対しては Collective-I/O を用いてアクセスを行っている．NFS は，表 2 に示されたハードウェアを NFS サーバとして用い，ストレージは RAID キャッシュ512 MB，RAID6，SATA II 3 Gbps で構成された RAID ストレージを用いている．

7.2 IOR

IOR [13] は LLNL によって開発された並列ファイルアクセスのベンチマークソフトウェアである．IOR では**図 7** に示すような 3 つのパラメータ（Segment，BlockSize，TransferSize）を用いてアクセスパターンの定義を行うことができる．今回の評価では MPI_File_close を行う前に 1 度だけ MPI_File_sync を行うように変更を加えた．IOR におけるバンド幅は，すべてのプロセスのうち MPI_File_open が開始された最小の時間から，MPI_File_close が終了した最大の時間までを用いてバンド幅の計算を行っている．

7.2.1 書き込み性能

この評価では，Segment = 128，BlockSize = 4 MByte，TransferSize = 4 MByte に設定した．それぞれのプロセスは，合計 512 MByte のデータを 1 つのファイルに対して書き込みを行う．MPI-IO/Gfarm の評価では，Gfarm のファイルシステムノードを 7 ノード用い，同じ 7 ノードを計算ノードとしても動作させ評価を行った．プロセス数が 7 に満たない場合，計算プロセスが使うファイルサーバノードはプロセス数と同じ数となる．PVFS2 では，全体で 14 ノードを用い，うち 7 ノードを計算ノード，他の 7 ノードをファイルサーバノードとして動作させた．PVFS2 では計算プロセスが 7 ノードに満たない場合においても，7 ノードのファイルシステムサーバに対してストライピングを行って書き込み処理を行っている．書き込みの評価結果を**図 8** (a) に示す．

表 2　評価環境
Table 2　Hardware and software environment.

Hardware	
CPU	Xeon E5410 2.33 GHz * 2
Memory	32 GB
Kernel	2.6.18-6-amd64
NIC	10 GigE
Local Disk	SATA
bonnie++	
Seq Output	Ave:52.3 (Max:57.4 Min:47.6) [MByte/s]
Seq Input	Ave:63.5 (Max:69.8 Min:53.3) [MByte/s]
Software	
Gfarm	2.3.0
MPI	mpich2-1.2.1
PVFS2	2.8.1
NFS	v3
IOR	2.10.2
BTIO	NPB3.3
HPIO	1.55

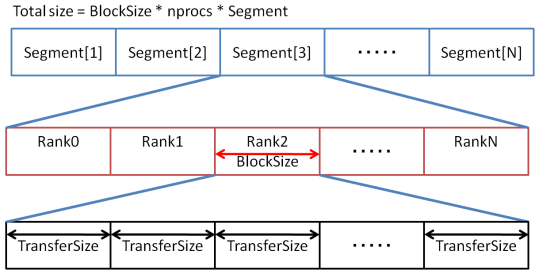
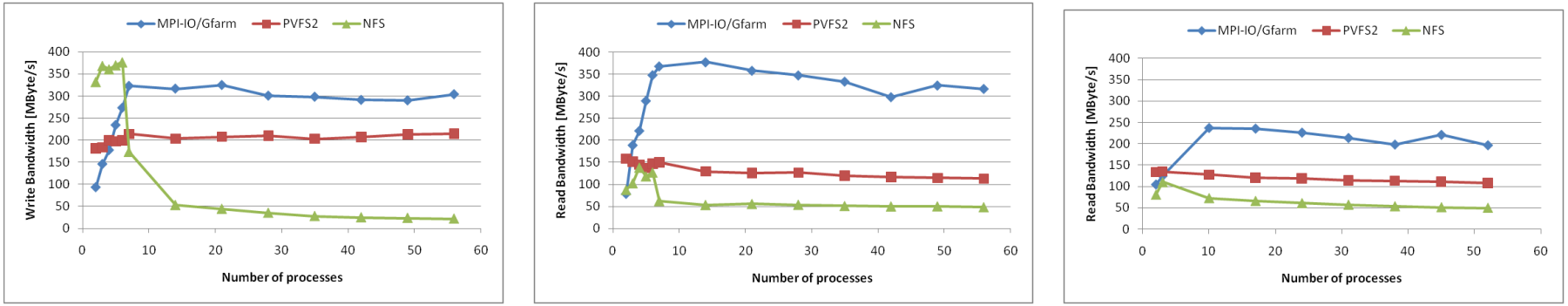


図 7　IOR のアクセスパターンの概要
Fig. 7　Access pattern of IOR benchmark.



(a) IOR: Write performance　(b) IOR: Read performance using same file view　(c) IOR: Read performance using different file view

図 8　IOR による評価結果
Fig. 8　Result of IOR benchmark.

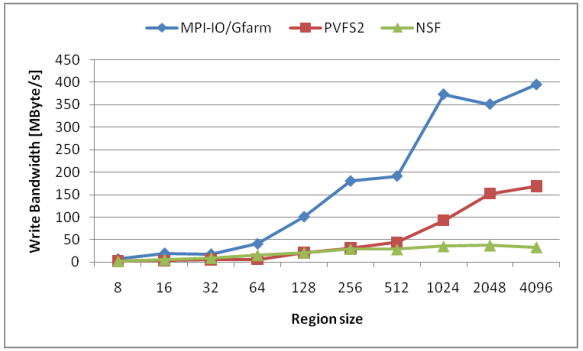
MPI-IO/Gfarm は 2 から 7 プロセスの間はスケールしている．この間は 1 ノードあたり 1 プロセスが動作しておりノード数が増えている状態である．その後は 1 ノードあたりのプロセス数が増加しており，ローカルストレージの性能に抑えられ 300–350 MByte/s となっている．それぞれのプロセスが動作するノードに書き込み処理が分散されているため，性能はノード数に対してスケールしていることが分かる．MPI-IO/PVFS2 の場合は約 200 MByte/s で一定となっている．PVFS2 ではプロセス数が 7 より少ない場合においても最初から 7 ノードのファイルシステムノードを用いて書き込みを行っているため，プロセス数が 2–4 の間では MPI-IO/Gfarm よりも良い性能を示しているが，その後はつねに MPI-IO/Gfarm が良い性能を示している．NFS ではプロセス数が 2–6 の間は最大 370 MByte/s の書き込み性能を示した．しかしその後プロセス数が増加するに従って性能が低下し，7 ノード 56 プロセスでの書き込みで MPI-IO/Gfarm が 15 倍の性能を示した．

7.2.2　読み込み性能

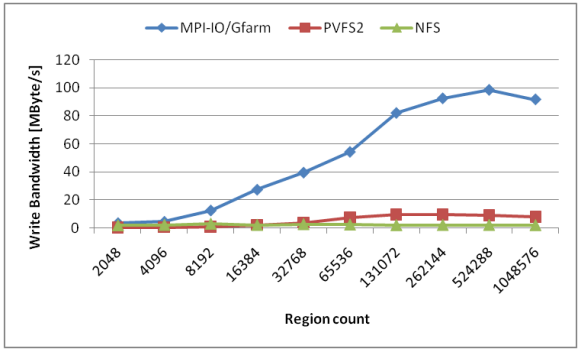
読み込み性能の評価では 2 つのパターンでの評価を行った．Gfarm，PVFS2 の設定は書き込み時と同じである．読み込みを行う前にバッファキャッシュのフラッシュを行っている．

1 つ目は，上述の書き込み評価で生成されたファイルに対して，同じプロセス数かつ同じ FileView を用いて読み込みを行った場合の評価である．書き込み時と同じランク番号を持つ MPI プロセスが同じ FileView を指定して読み込みを行う．この場合の評価結果を図 8 (b) に示す．MPI-IO/Gfarm の評価結果では，書き込みの性能と同様に，2–7 プロセスの間ではスケールしている．その後は，ストレージの性能に抑えられ 300–380 MByte/s となっている．書き込み時と同じ FileView を指定して読み込みを行った場合，それぞれのプロセスはローカルストレージ上にある，1 つのファイルに対してのみ読み込みを行う．そのためノード数に対してスケーラブルな読み込み性能を得られている．プロセス数が増加した場合に，ファイルアクセスの競合によって若干性能が低下しているが，MPI-IO/PVFS2 では 120 から 160 MByte/s となり，MPI-IO/Gfarm の方が約 2.5 倍良い性能を示した．NFS では，4 プロセスで最大 140 MByte/s を示し，7 プロセス以降は約 50 MByte/s となっている．

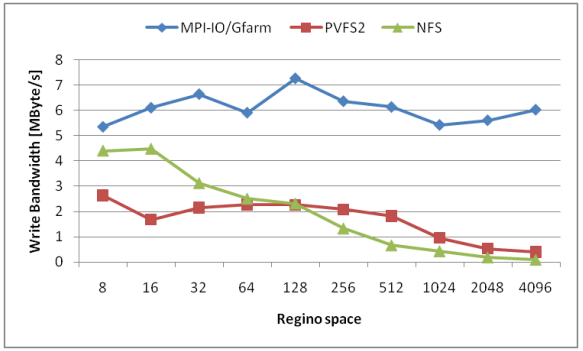
2 つ目は，異なるプロセス数を用いて読み込みを行った場合である．この評価では，書き込み時より 4 プロセス少ないプロセス数で読み込みを行う．アクセスパターンに関するパラメータは，BlockSiize = 4 MByte，TransferSize = 4 MByte で書き込み時と同じ値を指定し，Segment = 128 + 512/nprocs，を指定した．評価結果を図 8 (c) に示す．MPI-IO/Gfarm では，250–200 MByte/s となった．この場合，プロセスはローカルストレージだけではなく，リモートのストレージにもアクセスする可能性があり，また複数のファイルに対してランダムアクセスを行っているため，同じ FileView を指定した場合に比べ性能は低くなっている．MPI-IO/PVFS2 では，同じ FileView を指定した場合とほぼ同じ性能となっているが，異なるプロセス数で読み込みを行った場合においても MPI-IO/Gfarm が良い性能を示した．MPI-IO/PVFS2 と同様に，MPI-IO/NFS においてもプロセス数が異なった場合でも性能に変化はない．



(a) HPIO: Vary region size



(b) HPIO: Vary region count



(c) HPIO: Vary region spaceing

図 9　HPIO による評価結果
Fig. 9　Result of HPIO benchmark.

7.3　HPIO

HPIO ベンチマーク[14] は Vorthwestern University で開発された並列 I/O のベンチマークソフトウェアである．HPIO は region count，region size，region spaceing の 3 つのパラメータを用いて，様々なアクセスパターンの評価が行える．またメモリ上のデータ配置が連続か非連続か，ファイル上のデータ配置が連続か非連続かによって 4 つのアクセスパターン（c-c，c-nc，nc-c，nc-nc）を指定することができる．**図 10** にプロセス数を 2，region count を 3，region size と region spaceing を任意の値に指定し “nc-nc” を用いた場合のアクセスパターンの例を示す．本評価では，デフォルトのパラメータを region count = 4,096，region size = 8 byte，region spacing = 128 byte に設定し，“nc-nc” パターンを用いて書き込みを行う．それぞれ 3 つのパラメータのうち 1 つを変化させ，評価を行う．region size を科学技術計算におけるデータの最小単位と考えられる 8 byte とすることで，最も厳しい条件下での評価となっている．

Gfarm のノード構成は，7 ノードを用いファイルサーバと計算ノードを同じノード上に動作させた．PVFS2 のノード構成は，合計 14 ノードを用いて，うち 7 ノードをファイルサーバノード，他の 7 ノードを計算ノードとして動作させた．1 ノードあたり 1 プロセスを動作させ，合計 7 プロセスを用いて評価を行った．

7.3.1　region size を変化させた場合

region size を変化させた場合の結果を**図 9** (a) に示す．本評価では，region size を 8 byte から 4,096 byte まで増加させた．MPI-IO/Gfarm，MPI-IO/PVFS2，MPI-IO/NFS はともに，region size が増加するに従って性能は向上している．しかし，4,096 byte を指定した場合において，MPI-IO/Gfarm では 400 MByte/s，MPI-IO/PVFS2 では 160 MByte/s となり約 2.5 倍の性能差，MPI-IO/NFS では 35 MByte/s となり 10 倍以上の性能差を示した．

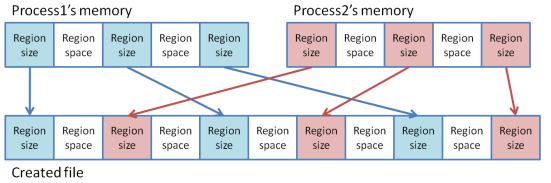


図 10　HPIO のアクセスパターンの概要
Fig. 10　Access pattern of HPIO benchmark.

7.3.2　region count を変化させた場合

region count を変化させた場合の結果を図 9 (b) に示す．本評価では，region count を 2,048 から 1,048,576 まで増加させ評価を行った．それぞれのプロセスは，合計 region count * 8 byte のデータを非連続な状態でメモリ上に持ち，それをファイルに対して非連続に書き込みを行う．MPI-IO/Gfarm は，region count が増加するに従って性能が向上し，最大

100 MByte/s となった．対して MPI-IO/PVFS2 は，最大 10 MByte/s，MPI-IO/NFS では，約 2 MByte/s で一定となっている．

7.3.3　region spaceing 変化させた場合

region spaceing を変化させた場合の結果を図 9 (c) に示す．本評価では，region spacing を 8 byte から 4,096 byte まで変化させた．それぞれのプロセスは，region size * region count = 32,768 byte のデータを非連続な状態でメモリ上に持ち，それをファイルに対して非連続に書き込みを行っている．MPI-IO/Gfarm では最大約 6 MByte/s，MPI-IO/PVFS2 では最大約 2 MByte/s，MPI-IO/NFS では最大約 4.5 MByte/s となった．書き込みデータの合計値が小さいため性能は低いものとなっているが，region space が大きくなった場合に MPI-IO/PVFS2 と MPI-IO/NFS の性能は低下しているのに対して，MPI-IO/Gfarm の性能は 5 から 7 MByte/s の間で一定となっている．

7.3.4　考　　察

MPI-IO/Gfarm は，ファイルに対して非連続なアクセスを行う場合においても分散ファイルに対して連続して書き込みを行うことができる．そのため，メモリ上にデータが非連続に格納されている場合に非連続なデータを連続領域にコピーしそれを 1 度の I/O リクエストで処理することができる．異なる region size，region count での評価で MPI-IO/Gfarm が非連続なデータアクセスにおいても全体のデータ量が増加するに従って，性能が向上しているのはこのためである．また異なる region spacing の評価で性能が一定となっているのも以上の理由からである．本手法を用いることで非連続なアクセスを効率的に扱うことができると考えられる．

7.4　BTIO

BTIO ベンチマーク[15] は NAS parallel ベンチマークに含まれる並列 I/O のベンチマークソフトウェアである．今回の評価では，A，B，C の 3 つの問題サイズを用いて評価を行った．それぞれ A は 419.43 MByte，B は 1,697.93 MByte，C は 6,802.44 MByte のデータを 1 つ作成する．Gfarm，PVFS2 ともに 14 ノードを用いて，14 ノードすべてをファイルシステムノード，計算ノードとして用いた．本評価では Sync 処理を含んでいない．

MPI-IO/Gfarm の評価結果を**図 11** に，MPI-IO/PVFS2 の評価結果を**図 12** に示す．MPI-IO/Gfarm においてはプロセス数が増加するに従って，書き込み性能が伸びているのに対して，MPI-IO/PVFS2 の場合プロセス数が増加するに従って書き込み性能は低下している．MPI-IO/Gfarm については，プロセス数が大きくなった場合に 1 プロセスあたりの書き込みデータ量が小さくなり，性能が伸びずその後低下しているが，書き込みデータ量が十分ある場合においてはプロセス数に対してスケーラブルな書き込み性能を示している．14 ノード上で 64 プロセスを動作させた場合，クラス C で MPI-IO/Gfarm では 6 GByte/s，MPI-IO/PVFS2 では 150 MByte/s となっている．MPI-IO/NFS では，実行時間が非常に長く終了しなかった．

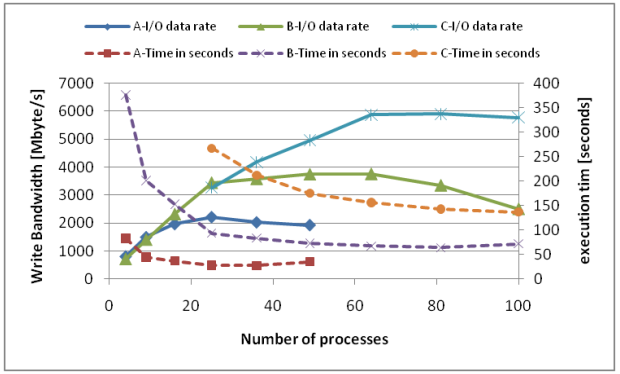


図 11　BTIO：MPI-IO/Gfarm の性能
Fig. 11　BTIO: Performance of MPI-IO/Gfarm.

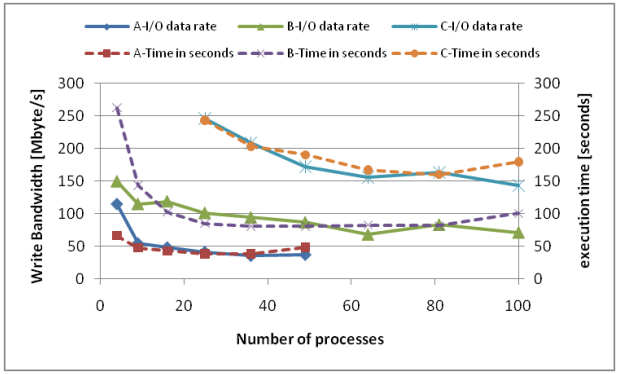


図 12　BTIO：MPI-IO/PVFS2 の性能
Fig. 12　BTIO: Performance of MPI-IO/PVFS2.

8. 関連研究

これまでに Lustre，GPFS，PVFS などの並列ファイルシステムに対する MPI-IO の最適化に関する研究がなされている．Lustre における最適化では[6]，ストライピングによるオーバヘッドを軽減しファイルアクセス性能を向上させるために，1 つのファイルを複数のサブファイルに分割し階層的なストライピングを行う手法を用いることで性能改善を行っている．ファイルのクローズ時にサブファイルをまとめ 1 つのファイルにまとめる操作を行う．また GPFS における最適化では[7]，Collective-I/O における通信量を減らし，通信と I/O 処理をオーバラップさせることで最適化を行っている．PVFS における最適化[5] では，Collective-I/O の高速化を行うため，ストライピングされたデータの位置を考慮したアクセスを行う．プロセスが動作しているノードのローカルディスクに存在するデータに対してアクセスを行うことで，通信量を減らしている．これらの研究においてはそれぞれのファイルシステムに特化した最適化となっている．またストライピングを行うファイルシステムであり本研究の対象とは異なっている．

Parallel Log-structured File System（PLFS）[10] は，チェックポイントの書き出しにおいて複数のプロセスが 1 つのファイルに対して書き込みを行う場合に，性能がスケールしないという問題を Lustre，PanFS，GPFS に焦点をあて最適化を行っている．本研究と同様に “N-1” パターンのアクセスを “N-N” パターンに置換えを行い性能改善を行っている．しかし，PLFS では FUSE を用いて実装を行っており実装レイヤが異なる．また PLFS は，書き込みごとに分散されたデータのアクセスパターンに関するログをとる必要があるのに対して，MPI-IO/Gfarm では，FileView を用いたメタデータ管理を行っている点が大きく異なる．

Catwalk-ROMIO [16] は，オンデマンドファイルステージングシステムである Catwalk [17] を MPI-IO の層で ADIO を用いて実装したものである．単一のファイルサーバを仮定している点において本研究と大きく違うが，書き込み処理を一時的にジャーナルファイルとして計算ノードのローカルディスクに対して行い，クローズ時にファイルサーバに対し書き込みを行っている．また I/O Delegate Cache System（IODC）[18] では，計算ノードの一部を IOD ノードとして動作させすべての I/O リクエストを IOD ノードに中継させる．IOD ノードでは複数のリクエストを集約するなどの最適化，データのキャッシュが行われ，並列 I/O の最適化を行っている．計算ノードに対して 10%程度の IOD ノードを用意することで最大 500%の性能改善を示している．Lustre や GPFS などのファイルシステムを対象にしており，Gfarm のような計算ノードのローカルストレージを活用する場合とは異なる．

9. まとめ

本稿ではまず Gfarm に最適化した MPI-IO 実装である MPI-IO/Gfarm の設計，実装について述べた．“N-1” パターンの書き込み性能の改善を行うため，“N-1” パターンのアクセスをそれぞれのプロセスが個別のファイルを作成し，個別のファイルに対して書き込みを行う “N-N” パターンのアクセスに置換えを行う．アプリケーション透過にするために，自動的にグローバルオフセットとローカルオフセットの変換を行う．

また性能評価では，IOR，HPIO，BTIO の 3 つのベンチマークソフトウェアを用いて評価を行った．“N-1” パターンの書き込みを行った結果，いずれの評価においてもノード数に対してスケーラブルな性能を示し，本手法の有効性を示した．さらに PVFS2，NFS との性能比較ではつねに MPI-IO/Gfarm が良い性能を示し，7 ノードのファイルシステムサーバを用いた場合に最大 2.5 倍から 10 倍，MPI-IO/Gfarm が良い性能を示した．

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」，文科省次世代 IT 基盤構築のための研究開発「研究コミュニティ形成のための資源連携技術に関する研究」（データ共有技術に関する研究）および情報爆発時代に向けた新しい IT 基盤技術の研究，文部科学省科学研究費補助金「特定領域研究」（課題番号 21013005）による．

参考文献

1) Tatebe, O., Hiraga, K. and Soda, N.: Gfarm Grid File System, *New Generation Computing*, Vol.28, No.3, pp.257–275, Ohmsha, Ltd. and Springer, DOI:10.1007/s00354-009-0089-5, Vol.2007, No.122, pp.7–12 (2010) (online), available from ⟨http://ci.nii.ac.jp/naid/110006549609/en/⟩.
2) Yamamoto, N., Tatebe, O. and Sekiguchi, S.: Parallel and Distributed Astronomical Data Analysis on Grid Datafarm, *Proc. 5th IEEE/ACM International Workshop on Grid Computing* (*Grid 2004*), pp.461–466 (2004).
3) Nishida, S., Katayama, N., Adachi, I., Tatebe, O., Sato, M., Boku, T. and Ukawa, A.: High Performance Data Analysis for Particle Physics using the Gfarm file system, *Journal of Physics: Conference Series*, 119, 062039, DOI:10.1088/1742-6596/119/6/062039 (2008).
4) Rajeev, T., William, G. and Ewing, L.: On implementing MPI-IO portably and with high performance, *IOPADS '99: Proc. 6th workshop on I/O in par-*

allel and distributed systems, New York, NY, USA, ACM, pp.23–32 (online), DOI:http://doi.acm.org/10.1145/301816.301826 (1999).
5) Jonathan, I., Cyril, R. and Gil, U.: Improving MPI-I/O Performance on PVFS, *Euro-Par '01: Proc. 7th International Euro-Par Conference Manchester on Parallel Processing*, London, UK, pp.911–915, Springer-Verlag (2001).
6) Weikuan, Y., Jeffrey, V., Shane, C.R. and Song, J.: Exploiting Lustre File Joining for Effective Collective IO, *Proc. 7th IEEE International Symposium on Cluster Computing and the Grid, CCGRID '07*, Washington, DC, USA, IEEE Computer Society, pp.267–274 (online), DOI:http://dx.doi.org/10.1109/CCGRID.2007.51 (2007).
7) Jean-Pierre, P., Richard, T., Richard, H., Bin, J. and Alice, K.: MPI-IO/GPFS, an optimized implementation of MPI-IO on top of GPFS, *Supercomputing '01: Proc. 2001 ACM/IEEE conference on Supercomputing* (*CDROM*), New York, NY, USA, ACM, p.17 (online), DOI:http://doi.acm.org/10.1145/582034.582051 (2001).
8) MPI Forum: MPI-2: Extensions to the Message-Passing Interface (1997).
9) FUSE: Filesystem in Userspace, available from ⟨http://fuse.sourceforge.net/⟩.
10) 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 (online), DOI:http://doi.acm.org/10.1145/1654059.1654081 (2009).
11) Rajeev, T., William, G. and Ewing, L.: Data Sieving and Collective I/O in ROMIO, *FRONTIERS '99: Proc. 7th Symposium on the Frontiers of Massively Parallel Computation*, Washington, DC, USA, IEEE Computer Society, p.182 (1999).
12) Rajeev, T. and Ewing, L.: An Abstract-Device Interface for Implementing Portable Parallel-I/O Interfaces, *Proc. 6th Symposium on the Frontiers of Massively Parallel Computation*, pp.180–187, IEEE Computer Society Press (1996).
13) IOR HPC Benchmark, available from ⟨http://sourceforge.net/projects/ior-sio/⟩.
14) Ching, A., Choudhary, A., Liao, W.-K. and Pundit, N.: Evaluating I/O characteristics and methods for storing structured scientific data, *Proc. International Parallel & Distributed Processing Symposium* (2006).
15) Wong, P. and Van der Wijngaart, R.F.: NAS Parallel Benchmarks I/O Version 2.4, NAS Technical Report NAS-03-002 (2003).
16) Hori, A., Kamoshida, Y., Matsuba, H., Yasui, T., Sumimoto, S. and Ishikawa, Y.: An Implemenation of an MPI-IO Using Catwalk File Staging System, IPSJ SIG Technical Report, Vol.2009-HPC-121, No.14 (2009).
17) Hori, A., Kamoshida, Y., Matsuba, H., Ohta, K., Yasui, T., Sumimoto, S. and Ishikawa, Y.: On-demand file staging system for Linux clusters, *CLUSTER '09: IEEE International Conference on Cluster Computing and Workshops*, pp.1–10 (online), DOI:10.1109/CLUSTR.2009.5289189 (2009).
18) Nisar, A., Liao, W.-K. and Choudhary, A.: Scaling Parallel I/O Performance through I/O, *Proc. 2008 ACM/IEEE Conference on Supercomputing* (2008).

(平成 23 年 4 月 12 日受付)
(平成 23 年 7 月 8 日採録)

木村　浩希（学生会員）
昭和 62 年生．平成 22 年筑波大学第三学群情報学類卒業．現在，同大学大学院システム情報工学研究科在学中．並列ファイルアクセスに関する研究に従事．

建部　修見（正会員）
昭和 44 年生．平成 4 年東京大学理学部情報科学科卒業．平成 9 年同大学大学院理学系研究科情報科学専攻博士課程修了．同年電子技術総合研究所入所．平成 17 年独立行政法人産業技術総合研究所主任研究員．平成 18 年筑波大学大学院システム情報工学研究科准教授．博士（理学）（東京大学）．超高速計算システム，グリッドコンピューティング，並列分散システムソフトウェアの研究に従事．日本応用数理学会，ACM 各会員．