情報処理学会研究報告
IPSJ SIG Technical Report

Vol.2010-HPC-124 No.15
2010/2/23

広域分散ファイルシステム Gfarm の MPI-IO の実装

木村浩希[†1]　建部修見[†2]

本稿では，MPI-IO によって複数のプロセスから 1 つのファイルに対するアクセスを Gfarm ファイルシステム上においてスケーラブルなアクセス性能を実現するための手法を提案する．Gfarm ファイルシステムでは 1 つのファイルに対して複数のプロセスが書き込みを行うとファイルの実体が存在するノードにアクセスが集中し性能低下が起こる．そこでそれぞれのプロセスが個別のファイルとしてローカルディスクに書き込むことでアクセスの集中を回避する．ADIO を用いて実装を行い IOR，BTIO の 2 つのベンチマークを行った．IOR では最大 20GB/ s の性能が得られ，BTIO では最大 12GB/s の書き込み性能を得ることができた．

An Implementation of MPI-IO for Gfarm Global File System

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

This paper proposes design and implementation of MPI-IO/Gfarm to achieve scalable file I/O performance for Gfarm global file system. Accessing a single file from multiple processes tends to decrease file I/O performance. Avoiding the access concentration is a key for scalable performance. MPI-IO/Gfarm transforms parallel access to a single file to parallel access to each individual file transparently. Performance evaluation of IOR and BTIO shows scalable I/O bandwidth, and achieves the I/O bandwidth of 20 GB/sec and 12 GB/sec, respectively, using 28 client nodes in two PC clusters located in different sites.

†1 筑波大学第三学群情報学類
College of Information Science, Third Clusters of Colleages University of Tsukuba
†2 筑波大学大学院システム情報工学研究科
University of Tsukuba, Graduate School of Systems and Information Engineering

1. はじめに

近年，CPU 演算性能の向上に伴ない物理学，天文学などさまざまな分野の科学技術計算で扱うデータの大規模化が進んでいる．このようなデータは地理的に離れたさまざまな拠点に置かれ広域に分散され存在しており，広域に分散されたデータを効率的に扱うため，広域分散ファイルシステムである Gfarm[1] が使われている．Gfarm は広域に分散するノードのローカルディスクを束ねて一元管理することができ，現在 Gfarm 上で科学技術計算をおこなう研究がされてきている．

科学技術計算におけるファイルアクセスは複数のプロセスが並列にファイルアクセスを行う．並列入出力ライブラリとしては MPI-IO が利用されている．Gfarm 上でこのような並列ファイルアクセスが行われたとき，複数のプロセスが個別のファイル対して書き込みを行う場合にはスケーラブルなアクセス性能を得られるが，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のファイルアクセス性能は非常に悪くなってしまう．

そこで本研究では，MPI-IO による複数のプロセスから 1 つのファイルに対する並列ファイルアクセスであっても Gfarm ファイルシステムに対してスケーラブルなファイルアクセス性能を実現するための手法を提案し，ADIO[2] を用いて実装した．

ADIO は並列ファイルアクセスの標準インターフェイスとして MPI-2 によって規定された MPI-IO の実装である ROMIO[3] がファイルシステムへのインターフェイスとして用いており，ADIO を実装することでさまざまなファイルシステムに対応した効率的なアクセスが可能となる．また MPI-IO は並列ファイルアクセスを柔軟に効率的に行うための仕組みを持っており ADIO を用いて実装することは有効な手法であると考えられる．

本論文の構成は，第 2 章で既存の並列ファイルアクセスの最適化に関する関連研究をあげる．第 3 章では既存の MPI-IO の実装の問題点を明らかにし，設計について述べる．第 4 章では実装について述べる．第 5 章では IOR[4]，BTIO[5] の 2 つのベンチマークプログラムを用いて性能評価を行った結果について述べ，最後にまとめと今後の課題を述べる．

2. 提案手法

2.1 予備実験

Gfarm 上で複数のプロセスからの書き込みアクセスについて以下の 2 つのアクセスパターンについて事前評価を行った．評価環境はで結果を **図 1** に示す．1 ノード 1 プロセスを動作させノード数を増加させた．

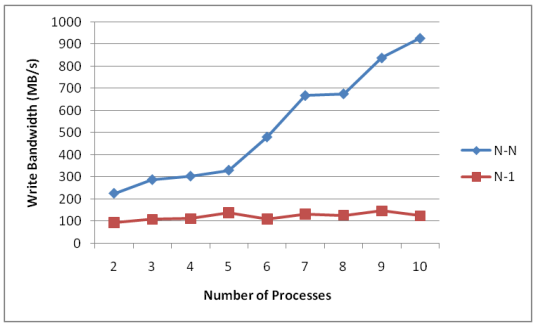


図 1 IOR による事前評価

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（N-N）

このパターンにおける Gfarm 上での書き込み処理は，プロセスが動いているノードのローカルディスクに対して書き込み処理が行われる．そのため書き込み処理が分散され 1 つのノードのファイルアクセス性能に制限されることがないためノード数に比例したファイルアクセス性能を示している．

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（N-1）

このパターンにおける Gfarm 上での書き込み処理は，Gfarm がファイルのストライピングをおこなっていないため，1 つのノードのローカルディスクにファイルの実体が作成される．そのため複数のプロセスから 1 つのノードのローカルディスクに対してアクセスが集中しファイルアクセス性能の低下が起こっている．

本研究では以上のような複数のプロセスから並列に 1 つファイルに対するアクセス性能が 1 つのノードのディスクアクセス性能に制限されてしまう問題を解決し，ノード数に比例したアクセス性能が得られる手法の提案を行う．

2.2 提案手法の概要

予備実験の結果をふまえ，Gfarm 上で行われる科学技術計算において複数のプロセスから 1 つのファイルに対する書き込みを行う場合にファイルの実体が存在する 1 つのノードにアクセスが集中しファイルアクセス性能が低下する問題を解決するために，複数プロセスから 1 つのファイルへの書き込みを 1 つのファイルではなくそれぞれのプロセスが個別のファイルとして書き込みを行うことでアクセスの集中を避ける手法を提案する．

提案手法の概要を**図 2** に示す．1 つのファイルを 1 つのディレクトリに対応させそれぞれのプロセスが個別のファイルを同一ディレクトリ内に保存する．個別に書き込まれたファイルが本来 1 つのファイルであったときにどの位置のデータであったかという情報を header ファイルとして保存することで，本提案手法で分散書き込みされたファイルに対する汎用的なアクセスを可能とする仕組みを用意する．本提案手法を用いることでファイルの実体が存在する 1 つのノードへのアクセス集中を避けることができる．これにより，複数のプロセスが個別のファイルを書き込む場合と同様にノード数に比例したファイルアクセス性能が得られると考えられる．

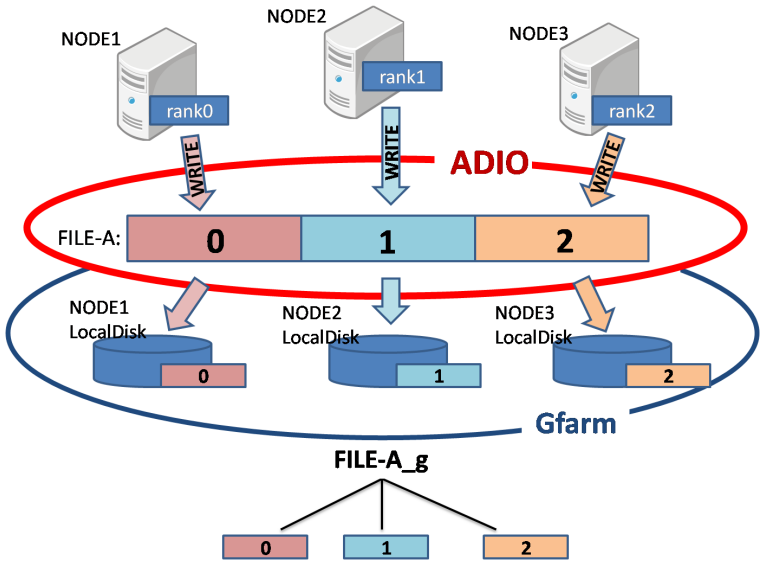


図 2 提案手法

3. 実　　装

3.1 ADIO

本研究では，提案手法を ADIO を用いて実装を行った．ADIO は MPI-IO の実装である ROMIO が用いるファイルシステムへのインターフェイスである．ADIO は定められた関数を定義する事で実装できる．またデフォルトで POSIX 準拠の関数が用意されているので，特別な動作を行わないものについてはデフォルトで用意されている関数のファイルアクセス

を行う部分を GfarmAPI を用いて書き換えた．

3.2 VIEW

MPI-IO では複数のプロセスが 1 つのファイルにアクセスを行うとき VIEW と呼ばれるそれぞれのプロセスがアクセスする範囲の定義を行うことができる．VIEW を用いて定義された範囲は非連続な範囲であってもアプリケーション側には連続した 1 つのファイルとして扱える仕組みが用意されており，MPI-IO では VIEW を予め定義することでそれぞれのプロセスのアクセス範囲を把握し，複数プロセス間で協調したファイルアクセスの仕組みを用意している．また本提案手法でも予めプロセスのアクセス範囲を把握することで分散書き込みされたデータの位置情報を効率的に扱うことができると考えられる．そこで本提案手法では，複数のプロセスから 1 つのファイルに対する書き込みが行われる時，VIEW をもちいてプロセスがファイルへ行うアクセスパターンの定義を行うものとし，VIEW の定義が行われた時に本提案手法を用いた分散書き込みを行う．VIEW によるアクセス範囲の定義を行わなかった場合は，デフォルトの書き込み処理が行われる．

3.3 ファイルアクセス処理の流れ

書き込み処理

(1) オープン (ADIOI_GFARM_Open())

(a) 新規書き込みオープン
この場合は通常通りあたえられたファイル名に対してオープンを行う．

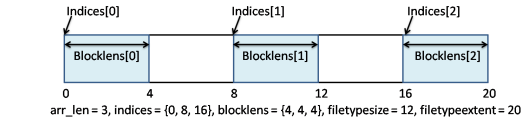
(b) 既存ファイルに対するオープン
通常のファイルに対するアクセスであれば通常通りアクセス対象のファイルをオープンする．本提案手法で分散書き込みされたファイルに対するアクセスの場合ここではファイルハンドラに分散書き込みされたファイルを扱うフラグを立てるのみである．

(2) VIEW の設定 (ADIOI_GFARM_Set_view())
まず通常通り VIEW の設定を行った後，新規書き込みの場合には以下の処理を行う．

(a) ディレクトリの作成
分散書き込みするデータと header ファイルを格納するディレクトリを作成する．ディレクトリ名は「⟨filename⟩_g」とした．

(b) ファイルのオープン
まず (1) でオープンされたファイルをクローズする．そしてプロセスごとに別々のファイルをオープンする．Gfarm では基本的に Create を行ったプロセスのローカルディスクにファイルの実体が作成されるため個別のファイルをオープンするだけでよい．ファイル名はランク番号とした．

(c) header ファイルの作成
それぞれのプロセスが設定した VIEW の情報を 1 つの header ファイルにまとめて保存する．header ファイルに保存されるデータを **図 3** に示す．



headerファイル	
nprocs	プロセス数
arr_len[]	indices, blocklens配列のサイズ
disp	VIEWの設定自にあたえられた変位
indices[]	blockの先頭変位
blocklens[]	連続するデータの長さ
filetypesize	VIEWのサイズ
filetypeextent	ファイル全体のサイズ

プロセス数分書き込む

図 3 headerfile の説明

(3) 書き込み (ADIOI_GFARM_WriteContig())
VIEW におけるローカル変位をグローバル変位に変換する処理が行われ，ADIOI_GFARM_WriteContig 関数に渡される．あたえられたグローバル変位をそれぞれのプロセスの VIEW と比較し分散書き込みされるファイルの変位に変換し書き込みを行う．

(4) クローズ (ADIOI_GFARM_Close())
分散書き込みされたファイルをクローズする．

読み込み処理

(1) オープン (ADIOI_GFARM_Open())
あたえられたファイル名から分散書き込み時に使われる「⟨filename⟩_g」が存在するかどうかチェックを行う．存在した場合には分散書き込みされたデータに対する読み込み処理と判断しファイルハンドラにフラグを立てる処理のみを行い，実際のオープン処理は VIEW の設定で行う．通常のファイルが存在した場合は通常の動作を行う．

(2) VIEW の設定 (ADIOI_GFARM_Set_view())

まず，header ファイルから分散書き込みされたデータの VIEW を読み込み，読み込んだ VIEW と現在指定されている VIEW との比較を行う．まず自プロセスと同じランク番号（存在しない場合は 0）の VIEW と比較を行い現在指定している VIEW と全く同じであればそのファイルのみをオープンする．一致しなかった場合は複数のファイルにまたがっているため，すべての VIEW と比較を行い自分の必要としているデータがあるファイルを探し，オープンする．

（3） 読み込み処理 (ADIOI_GFARM_ReadContig())
読み込み処理ではまずあたえられたグローバル変位から VIEW の設定でオープンしたどのファイルに位置するデータであるかを計算する．その後，グローバル変位を読み込むファイルのローカル変位に変換し読み込みを行う．

（4） クローズ (ADIOI_GFARM_Close())
オープンされたファイルをクローズする．

4. 性能評価

4.1 実験環境

実験には，InTrigger を利用した．InTrigger は日本国内の様々な教育・研究機関にまたがる広域分散計算環境である．実験に使用した InTrigger ノードのスペックと拠点を表 1 に示す．

MPI の実装は MPICH2 のバージョン 1.2.1 を用いた．既存の mpdboot はドメイン名を省略して扱っていたため，広域環境で MPI を使用するために省略を行わない用に修正を加えた．

Gfarm はバージョン 2 を用いた．また比較のため PVFS2 を tsukuba ノード上に構築した．バージョンは PVFS2-8.1 を用いた．tsukuba サイトの 13 台のノードすべてをファイルサーバー，メタデータサーバ，計算ノードとした．PVFS2 の設定について表 2 に示す．

表 1 実験に使用した Intrigger のスペック

tsukuba, kyutech	
CPU	Xeon E5410 2.33GHz
Memory	32GB
Kernel	2.6.18-6-amd64
Compiler	GCC version 4.1.2

表 2 PVFS2 の設定

pvfs2-fs.conf	
TroveSyncMeta	yes
TroveSyncData	no
TroveMethod	alt-aio
stripe size	64KB

4.2 IOR

IOR は LLNL によって開発され，MPI をもちいて POSIX，MPI-IO，HPDF5 などさまざまなインターフェイスに対応した並列ファイルアクセス性能の測定ができる．今回の実験では 1 プロセス 512MB のデータを 4MB ずつ 1 つのファイルに対して書き込む場合の性能を測定した．ファイルアクセスパターンを **図 4** に示す．実験には InTrigger の tsukuba サイト 13 台，kyutech サイト 15 台を用いた．1 ノード当たりのプロセス数は 1 プロセスから 4 プロセスまで変化させた．

提案手法を用いた Gfarm での書き込み性能の結果を **図 5** に PVFS2 を用いた書き込み性能の結果を **図 6** 示す．書き込み性能はメモリ容量が十分に大きいためメモリへの書き込み性能となっている．2 つのサイトで 28 ノードに 112 プロセスを配置した場合に最大で 20GB/s であった．予備評価 (図 1) ではノード数を増加させた場合でも性能の向上はなかったが，提案手法を用いることで大幅な性能向上を示すことができた．PVFS2 の場合 13 プロセスまではノード数にスケール下書き込み性能がえられているが，1 ノード当たりのプロセス数の増加に伴って性能が低下している．

次に，提案手法を用いた読み込み性能の結果を **図 7** に PVFS2 を用いた読み込み性能結果を **図 8** 示す．書き込み処理と同様に 1 プロセス 4MB ずつ合計 512MB のデータを読み込む．書き込み時と同じプロセスが同じアクセス範囲を指定して読み込みを行っており，書き込みが行われた直後に読み込みを行うためメモリキャッシュからの読み込み速度となっている．

4.3 BTIO

BTIO は NAS Parallel Benchmarks に含まれる BT (block tridiagonal) ベンチマークで並列ファイルアクセス性能の測定に使われている．バージョンは 3.3 を用い，SUBTYPE には full を用いた．BTIO は問題サイズでクラス分けされておりそれぞれクラス A で 0.4GB，クラス B で 1.6GB，クラス C で 6.8GB のファイルを 1 つ生成する．

図 9 に本提案手法を用いた Gfarm 上での結果を示し **図 10** に PVFS2 上で行った結果を

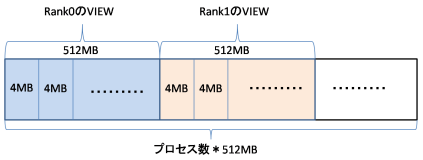


図 4 IOR のアクセスパターン

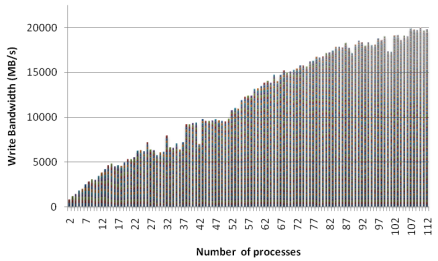


図 5 Gfarm 上における IOR を用いた Write 性能

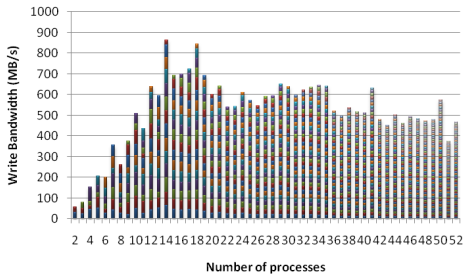


図 6 PVFS 上における IOR を用いた Write 性能

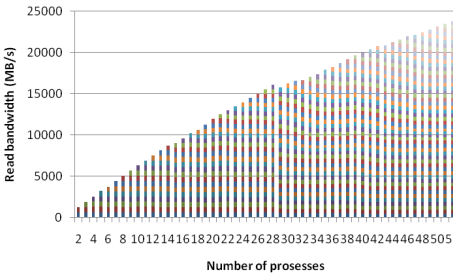


図 7 Gfarm 上における IOR を用いた Read 性能

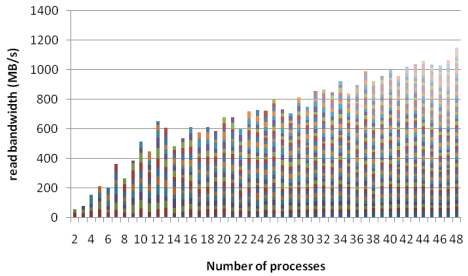


図 8 PVFS 上における IOR を用いた Read 性能

示す．縦軸右側が実行時間，左側が書き込みのバンド幅，横軸がプロセス数を表している．

tsukuba サイトで 1 ノード最大 8 プロセスを動作させ測定を行った．クラス C の時，プロセス数 121 以降は 2 つのサイトにまたがって性能を測定した．メモリのサイズが書き込むデータのサイズより十分大きいためメモリへの書き込みとなっている．PVFS2 においてはプロセス数が増加に伴なう書き込み性能の向上は確認できなかったが，Gfarm 上では 28 ノード 144 プロセスを動作させた場合クラス C において最大 12GB/ s の書き込み性能が得られた．しかしサイトをまたぐ場合には通信にかかる時間が増加し全体の実行時間が非常に遅くなっている．

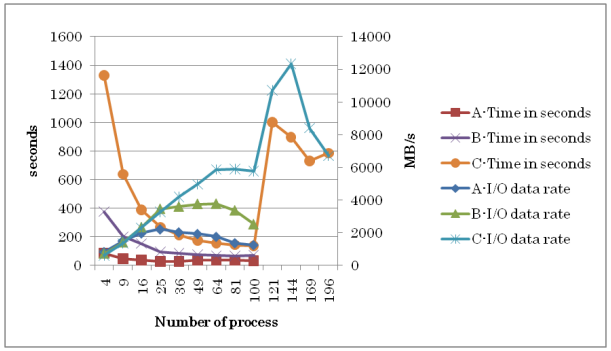


図 9 Gfarm 上における BTIO を用いた提案手法での性能評価結果

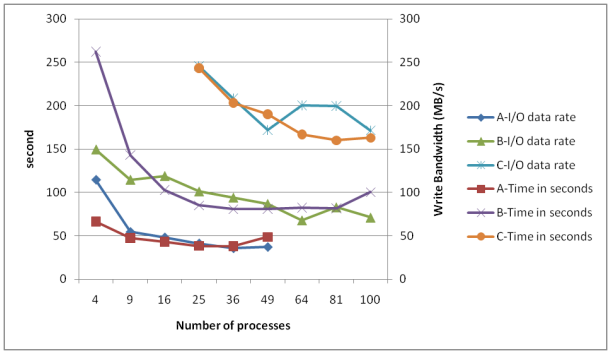


図 10 PVFS2 上における BTIO を用いた性能評価結果

5. 関 連 研 究

5.1 PLFS[6)]

PLFS は科学技術計算におけるチェックポイントの書き込みに焦点をあて PanFS，GPFS，Luster の 3 つのファイルシステムについて最適化を行った並列ログファイルシステムである．PLFS ではチェックポイントの書き込みにおけるアクセスパターンを複数のプロセスが個別のファイルを出力する N-N アクセスと複数のプロセスが 1 つのファイルに対して出力をする N-1 アクセスに分類しており，N-1 アクセスの場合ファイルアクセス性能が低下する問題点をあげ最適化を行っている．

PLFS では N-1 パターンのアクセスが行われる時にそれぞれのプロセスが書き込むデータを個別のファイルとして書き出し N-N パターンのアクセスに置換えを行う，個別に書き込まれたデータをアプリケーション側からは 1 つのファイルであるようにアクセスできる仕組みを用意している．N-1 パターンのアクセスを N-N パターンのアクセスに置き換えることで N-1 パターンの書き込みであっても N-N パターンと同等の性能を実現している．

5.2 Catwalk-ROMIO[7)]

Catwalk-ROMIO 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 の仕組みを ADIO を用いて実装したものである．Catwalk システムは NFS などによる 1 つのファイルサーバと複数の計算ノードで構成されるクラスタを対象にし必要なファイルをオンデマンドにステージングを行うシステムである．ADIO を用いた実装では複数のプロセスから 1 つのファイルに対するアクセスの場合ローカルディスクに一時ジャーナルファイルとして書き込みを行い，クローズが呼ばれた時点でファイルサーバーノードに書き込みを行う．それぞれのプロセスがローカルディスクに対するアクセスをおこなうためノード数に対してスケーラブルなアクセス性能を実演している．

6. おわりに

本研究では，MPI-IO による複数のプロセスから 1 つのファイルに対する並列ファイルアクセスに関して，Gfarm 上においてスケーラブルなファイルアクセス性能を実現する手法を提案し，実装を行った．提案手法では複数プロセスから 1 つのファイルに対する書き込み処理をプロセスごとに分割し個別のファイルとして書きだすことでローカルディスクを生かしたファイルアクセスが可能となった．InTrigger を用いて IOR，BTIO の 2 つのベンチマークを行った．IOR の書き込み性能では広域に分散する 2 つのサイトの 28 ノード 112 プロセスで最大 20GB/ s の性能が得られた．BTIO を用いた性能評価では最大 12GB/s の書き込み性能を得ることができた．

今後の課題としては，以下の点があげられる．

(1) 読み込み性能評価
今回の IOR を用いた読み込み処理の性能評価では，書き込みと読み込みが同じプロセスで同じアクセス範囲に対して行われている．そのため読み込みはローカルのディスクに対してのみアクセスが起こる．ローカルのディスクに必要なデータがない場合など，さまざまなアクセスパターンについて詳しく性能評価を行う必要がある．

(2) Collective-I/O[8)]
MPI-IO では複数のプロセス間で協調してファイルアクセスを行う仕組みとして Collective-I/O がある．今回の実装では Collective-I/O については考慮しなかったが，特に読み込み処理においては複数プロセスでの協調したファイルアクセスがが有効である可能性があり，検討する必要があると考えられる．

(3) アプリケーションによる性能評価
本提案手法が実アプリケーションにおいても有効な手法であることを示すために，アプリケーションによる性能評価を行う必要がある．

謝辞 本研究の一部は，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) および文部科学省次世代 IT 基盤構築のための研究開発「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」，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) による．

参 考 文 献

1) 建部修見, 曽田哲之. 広域分散ファイルシステム gfarm v2 の実装と評価. 情報処理学会研究報告 2007-HPC-113, 2007.
2) Rajeev Thakur and Ewing Lusk. An Abstract-Device Interface for Implementing Portable Parallel-I/O Interfaces. In *in Proceedings of The 6th Symposium on the Frontiers of Massively Parallel Computation*, pp. 180–187. IEEE Computer Society Press, 1996.
3) Rajeev Thakur, William Gropp, and Ewing Lusk. On implementing MPI-IO portably and with high performance. In *IOPADS '99: Proceedings of the sixth workshop on I/O in parallel and distributed systems*, pp. 23–32, New York, NY, USA, 1999. ACM.
4) IOR HPC Benchmark.
5) Parkson Wong and Rob F. Van der Wijngaart. NAS Parallel Benchmarks I/O Version 2.4. *NAS Technical Report NAS-03-002*, 2003.
6) John Bent, Garth A. Gibson, Gary Grider, Ben McClelland, Paul Nowoczynski, James Nunez, Milo Polte, and Meghan Wingate. PLFS: a checkpoint filesystem for parallel applications. In *SC*. ACM, 2009.
7) 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. 情報処理学会研究報告，2009-HPC-121, 2009.
8) Rajeev Thakur, William Gropp, and Ewing Lusk. Data Sieving and Collective I/O in ROMIO. In *FRONTIERS '99: Proceedings of the The 7th Symposium on the Frontiers of Massively Parallel Computation*, p. 182, Washington, DC, USA, 1999. IEEE Computer Society.