Vol.2010-HPC-126 No.21
2010/8/4

広域分散ファイルシステムGfarmのMPI-IOの実装と評価

木村浩希[†1]　建部修見[†1]

本稿では，Gfarm のための MPI-IO 実装である MPI-IO/Gfarm の実装と性能評価について述べる．Gfarm ファイルシステムは，広域に分散する計算ノードのローカルディスクをまとめ 1 つのネームスペースで管理を行える広域分散ファイルシステムである．Gfarm は計算ノードのローカルディスクを用いることで，スケーラブルな I/O 性能を実現しているが，複数のプロセスが 1 つのファイルに書き込みを行う場合書き込み性能がスケールしない．MPI-IO/Gfarm は複数のプロセスが 1 つのファイルに対して書き込みを行う場合，プロセスごとに個別のファイルを作成し書き込みを行うことで最適化を行っている．本稿では MPI-IO/Gfarm の書き込み，読み込み性能の性能評価を行い PVFS2 と性能比較を行った．7 ノードを用いて MPI-IO/Gfarm が，書き込み性能では最大 2 倍，読み込み性能では 2 倍から 4 倍の性能が得られた．

An Implementation and evaluation of MPI-IO for Gfarm file system

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

This paper describes an implementation of MPI-IO/Gfarm that is an MPI-IO implementation for the Gfarm file system. The Gfarm file system is a global file system that federates a local storage of compute nodes among several clusters. It has a scale-out architecture designed using a local storage of compute node. However, in case of N-1 parallel write pattern that writes to a single file, the write performance does not improve. MPI-IO/Gfarm optimizes it by transparently translating to N-N parallel write pattern. The evaluation results show scalable I/O performance up to number of physical I/O nodes. It is at most 2.0x write performance and 2.0x to 4.0x read performance compared with MPI-IO on PVFS2 using seven I/O nodes.

1. はじめに

現在，並列計算におけるファイルアクセスは，標準インターフェイスとして MPI-IO が使われている．MPI-IO を用いた並列ファイルアクセス性能を改善するために，PVFS2 , GPFS, Lustre，PanFS などの並列ファイルシステムのための MPI-IO が研究されてきた[1)–3)]．近年，解析データの大規模化が進んでおり，ヘクサバイト，ゼタバイトスケールのデータを扱う必要性がある．このようなデータは信頼性の観点から地理的に離れた複数の拠点に分散して存在している．分散しているデータを管理するため，Gfarm ファイルシステム[4),5)] のような広域分散ファイルシステムが必要となっている．よって Gfarm を用いた広域分散環境において並列アプリケーションが効率的にファイルアクセスを行えるための MPI-IO が必要である．

以上のような背景から我々は Gfarm 用に最適化された MPI-IO である，”MPI-IO/Gfarm” の開発を進めている．Gfarm は計算ノードのローカルディスクをまとめ，1 つのネームスペースでアクセスを行うことができる．Gfarm では複数のノードに分散するプロセスが個別のファイルに対して並列に読み書きを行う場合には，書き込み処理がそれぞれのノードのローカルディスクに分散し，スケーラブルなアクセス性能を得ることができる．また読み込みの場合においては，ファイル複製を用意することで 1 つのファイルに対する読み込み処理を分散させ，スケーラブルなアクセス性能を得ることができる．しかし，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ではスケーラブルな書き込み性能を得ることができない．MPI-IO/Gfarm は，複数のプロセスが 1 つのファイルに書き込みを行う場合の並列ファイルアクセス性能の改善を主な目的としており，単一ファイルへの書き込みをプロセスごとに個別のファイルを作成し書き込みを行うことで，書き込み処理をノード毎に分散させ，ノード数に対してスケーラブルなファイルアクセス性能を実現している．

本稿では，MPI-IO/Gfarm の設計と実装について述べ，書き込み性能と分散ファイルに対する読み込みの性能評価を行う．読み込み処理については，書き込み時とプロセス数が同じ場合と，異なる場合について性能評価を行う．

†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. MPI-IO/Gfarm の設計

複数のプロセスから Gfarm 上の 1 つのファイルに対して書き込みを行う場合，Gfarm ファイルシステムでは，create を呼んだプロセスが動作している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．他のノードで動作しているプロセスは，ファイルの実体が存在するノードに対して書き込みを行う．そのため実体の存在するノードのディスク性能に制限され，またシーク処理の競合により性能低下が引き起こる．このような，1 つのファイルに対する複数プロセスからの書き込み処理は，Lustre，PanFS ，GPFS 等の並列ファイルシステムにおいてもスケーラブルなファイルアクセス性能を得ることができず，最適化に関する研究がなされている．[8)]

以上の問題を解決するために MPI-IO/Gfarm 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を，プロセスごとに個別のファイルとして書き込みを行う．個別のファイルとして書き込みを行うことで，それぞれのプロセスが動作しているノードのローカルディスクに対する書き込みとなり，ノード数に対してスケーラブルな書き込み性能を得られる．また MPI-IO/Gfarm では，プロセスが書き込むデータの位置が非連続な場合でも，個別のファイルに連続して書き込みを行うことで，書き込み性能を向上することができると考えられる．

MPI-IO によるファイルアクセスは VIEW の定義を行う．VIEW とはプロセスがアクセスを行うファイル内の範囲であり，MPI-IO ではファイルの open が行われたあとに VIEW の定義を行う．VIEW を予め定義することで，Seek 処理を MPI-IO 内部に任せることができ，Collective I/O[6)] を用いた I/O 処理の最適化を行うことができる．MPI-IO/Gfarm は分散ファイルとして書き出したファイルをアプリケーション側からは，1 つのファイルとしてアクセス可能にするため，VIEW の情報を元にプロセスごとに書き出した個別ファイルと本来のファイルとのオフセットの変換をおこなう．読み込み時にアプリケーション側からあたえられる本来のファイルのオフセットを書き込み時に保存された VIEW の情報を元に分散ファイルのオフセットに変換を行う．

VIEW の情報を用いない場合，書き込んだオフセットとサイズを保存する必要があるが，VIEW の情報を用いることで，VIEW の定義を行った時点でオフセットの変換に関する情報を得ることができ，書き込み時にオフセットの対応を保存しておく必要がないため書き込み時のオーバーヘッドを削減できる．

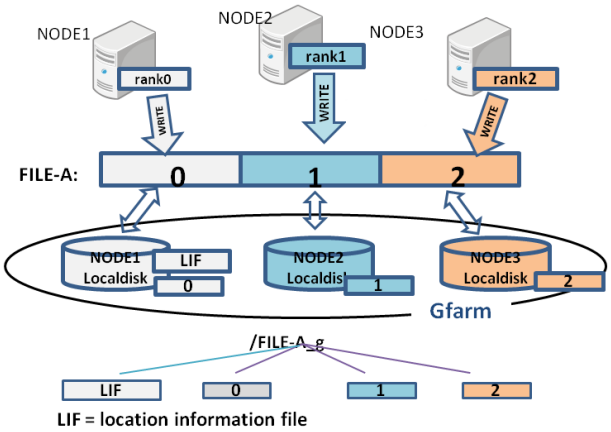

図 1　提案手法

3. MPI-IO/Gfarm の実装

3.1 ADIO

本研究では，ADIO[7)] を用いて実装を行った．ADIO は MPI-IO の実装の一つである ROMIO が用いているファイルシステムへのインターフェイスである．ADIO 内の定められた関数を定義することで，ファイルシステム個別の MPI-IO 実装が可能となる．実行時に MPI_File_open でファイルシステムの判別が行われ，適切な実装が選択される．

MPI-IO/Gfarm では，以下の 5 つの関数を定義した．

ADIOI_GFARM_open

MPI_File_open に対応する関数である．MPI_Info を用いて通常の書き込みと分散書き込みとを指定することができる．分散書き込みを行う場合この関数では実際の open 処理は行っていない．

ADIOI_GFARM_set_view

MPI_File_set_view に対応する関数である．新規ファイルに対する書き込みの場合，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い，指定された VIEW 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当たり 1 つ作成され，ノード内のプロセスの VIEW の情報を格納する．この時，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，一旦すべての VIEW 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 VIEW 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．他のプロセスと同じ範囲を持っている場合については．代表プロセスの create したファイルを open する．

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位置情報ファイルから書き込み時に保存されたすべての VIEW の情報を読み込む．現在指定されている VIEW の情報と読み込みを行った VIEW の情報とを比較し，アクセス範囲を含む分散ファイルを判別する．

ADIOI_GFARM_write

MPI_File_write に対応する関数である．与えられたデータを，MPI_File_set_view の定義時に open されたファイルに対してシーケンシャルに書き込みを行う．

ADIOI_GFARM_read

MPI_File_read に対応する関数である．この関数ではまず，現在の VIEW から本来のファイルのオフセットの計算を行う．そして分散書き込み時の VIEW の情報から，分散ファイルの選択，分散ファイル上のオフセットの計算を行う．

ADIOI_GFARM_close

open されたすべてのファイルを close する．

ADIOI_GFARM_sync

open されたすべてのファイルにたいして Sync をする．

3.2 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．

ファイルを新規に作成する場合の VIEW の定義時に，指定された VIEW の情報を元に作成する．位置情報ファイルは，ノードごとに 1 つのファイルが作成され，ノード内のプロセスが書き込む分散ファイルについての位置情報を書き込む．位置情報ファイルの書き込み処理がノードごとに分散するため，作成のためのオーバーヘッドを削減することができる．位置情報ファイルの例を図 2 と表 3.2 に示し，それぞれのパラメータについて以下で示す．

- 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
  VIEW の開始位置を示す．
- filetype_size
  VIEW 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
  VIEW 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．VIEW 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”で示す連続部分の開始位置を示す．

nprocs	3		
Arr_lens[]	{4, 6, 4}		
	Rank0	Rank1	Rank2
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}

4. 性能評価

4.1 実験環境

性能評価には InTrigger の Tsukuba サイトを使用した．スペックを表 1 に示す．PVFS2

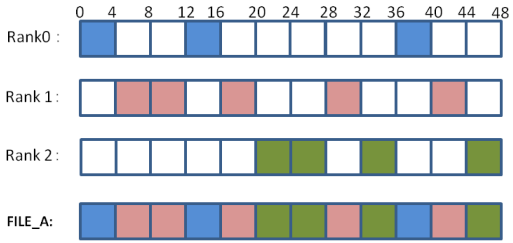

図 2 File view example in three processes case

を構築し，MPI-IO/Gfarm との性能比較を行う．用いたソフトウェアについて表 3 に示し，PVFS2 の設定について表 2 に示す．

Gfarm と PVFS2 の構成として，Tsukuba サイトの 7 ノードを用いて，7 ノードを計算ノードとファイルサーバノードの両方で動作させた場合を”Gfarm-7”，”PVFS2-7”とした．Tsukuba サイトの 14 ノードを用いて，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場合を”PVFS2-14”とした．

プロセス数は，1 ノード 1 プロセスを動作させノード数を増加させ，同様にノード当たりのプロセス数を順に増加させ，最大 1 ノード 8 プロセスを動作させた．

CPU	Xeon E5410 2.33GHz * 2
Memory	32GB
Kernel	2.6.18-6-amd64
Compiler	GCC version 4.1.2
NIC	10GigE

表 1 Intrigger

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

表 2 PVFS2 の設定

Gfarm	2.3.0
MPI	mpich2-1.2.1
pvfs	2.8.1

表 3 software environment

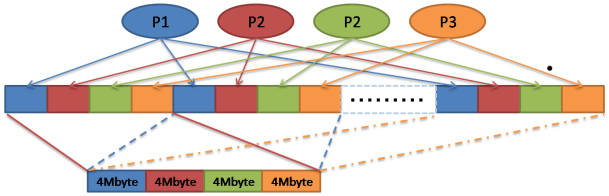

図 3 アクセスパターン

4.2 ベンチマークソフトウェア

アクセスパターンの概要について図 3 に示す．このベンチマークソフトでは，1 プロセス当たり 4MB のアクセス範囲をもつ VIEW を定義し，1 プロセス当たり合計 512MB のファイルアクセスを行う．書き込み処理は MPI_File_write の開始から MPI_File_sync が返ってくるまでの時間を計測し，書き込み性能を求めている．読み込み処理については MPI_File_read 関数の時間を計測している．書き込み時と異なる VIEW での読み込みについては，1 プロセス当たりのアクセス範囲を多くし，合計書き込み時と同じサイズ読み込みを行う．読み込み性能では，ディスクキャッシュの影響をなくすため，一旦メモリキャッシュをクリアした後読み込み処理を行っている．

4.3 書き込み性能

書き込み性能を図 4 に示す．MPI-IO/Gfarm では 1 ノード当たり 1 プロセスが動作している 2 − 7 プロセス間では書き込み性能がスケールし，7 ノードで最大 400MB/ s の書き込み性能が得られた．その後，1 ノード当たりのプロセス数が増加するに従って，同一ストレージへの書き込みの競合が発生し性能が低下している．PVFS2 での書き込み性能は，7 ノードで最大 300MB/ s で抑えられている．PVFS2 ではファイルサーバノードに均等にファイルをストライピングしており，リモートのディスクにアクセスを行っているため MPI-IO/Gfarm に対して低い性能であると考えられる．

4.4 読み込み性能

読み込み性能の評価は，書き込み時と同じ VIEW を指定した場合と書き込み時と異なる VIEW を指定した場合の 2 パターンで評価を行った．

書き込み時と同じ VIEW を指定した場合

結果を図 5 に示す．プロセスの割り当ては，ノード内のプロセスが書き込み時と同じランク番号もつように実行し，同じ VIEW を指定して読み込みを行っている．それぞれのプロセスが読み込みに必要なファイルは 1 つだけであり，すべてローカルディスクに

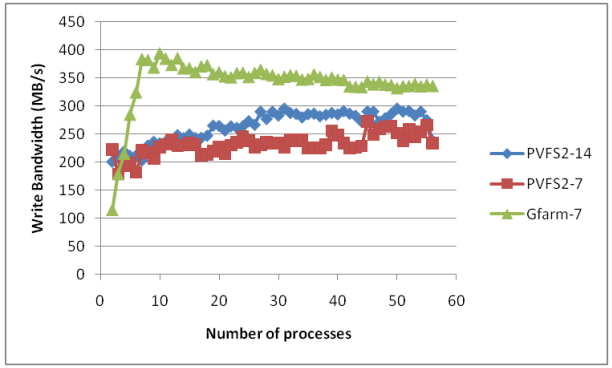

図 4 書き込み性能

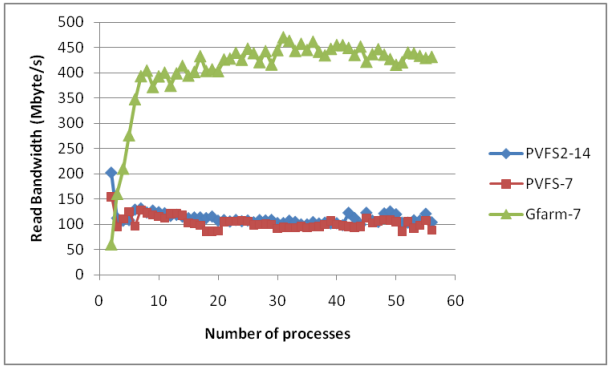

図 5 書き込み時と VIEW が同じ場合の読み込み性能

存在する場合である．そのため読み込み処理はプロセスが動作しているノードのローカルディスクに分散し，1 ノード当たり 1 プロセスが動作している 2 － 7 プロセス間でスケーラブルな読み込み性能を示している．その後，1 ノードのディスク性能に抑えられているが，最大約 450MB/ s の読み込み性能が得られた．

PVFS2 の場合，読み込み性能はスケールせず 2 プロセスの場合に最大の 200MB/ s の性能を示した後，約 100MB/ s で抑えられている．MPI-IO/Gfarm に対して性能が低いのは，PVFS2 ではファイルのファイルサーバノードに対して均等にストライピングしており，リモートのディスクに対して頻繁にアクセスを行っているためであると考えられる．

書き込み時と異なる VIEW を指定した場合

結果を図 6 に示す．この場合では，書き込み時のプロセス数より 4 プロセス少ないプロセス数で読み込みを行っている．そのため指定する VIEW は書き込み時と異なるものになり，MPI-IO/Gfarm では必要なデータがノード間に分散している状態である．リモートのディスクに対してアクセスを行う必要があり，書き込み時と同じ VIEW を指定した場合に比べ性能は約半分に低下し，約 200MB/ s となっている．PVFS2 では同じ VIEW を指定した場合とほぼ同程度の性能を示している．これはストライピングによってファイルサーバノードに均等にデータが分散され，VIEW が異なった場合の影響がでにくいためと考えられる．

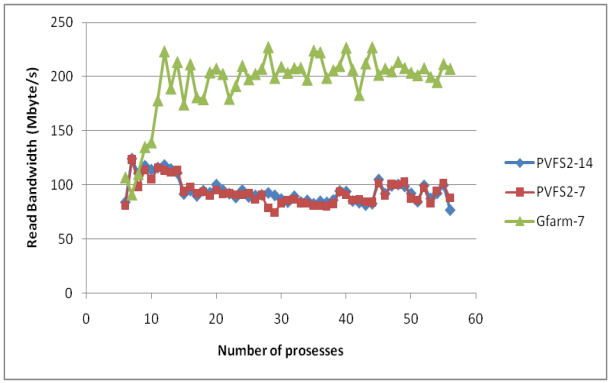

図 6 書き込み時と VIEW が異なる場合の読み込み性能

5. 関 連 研 究

5.1 PLFS

PLFS[8] は，チェックポイントの書き出しに焦点をあて，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，アクセス性能が低下する問題点を最適化している．本研究と同様に 1 つのファイルへの書き込みを，プロセスごとに個別のファイルとして書き出す手法を提案しているが，FUSE を用いて実装を行っている．FUSE を用いた実装では，MPI-IO

での実装に比べより汎用的なアクセスが可能となるがそのオーバーヘッドについては本研究の場合と比較検討が必要であると考えられる．

5.2 Catwalk-ROMIO

Catwalk-ROMIO[9] 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 MPI-IO を用いた実装である．単一のファイルサーバを仮定しているが，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ない，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．

6. おわりに

本稿では，Gfarm に対して MPI-IO を用いた並列ファイルアクセスの最適化を行った MPI-IO/Gfarm の設計と実装について述べ，性能評価を行った．MPI-IO/Gfarm は複数のプロセスから 1 つのファイルに書き込みを行う場合の性能を向上するため，プロセスごとに個別のファイルを作成し，ローカルディスクに書き込み処理を分散させる．VIEW の情報を利用し，分散ファイルと本来のファイルとの変換を行いアプリケーション側からは 1 つのファイルとして扱うことができる．

性能評価では，PVFS2 との比較を行った．書き込み性能では，MPI-IO/Gfarm はノード数に対してスケーラブルな性能を得ることができ，7 ノードで最大 400MB/ s となった．PVFS2 と比較して，常に良い性能を得たが，ノード内のプロセス数が増加するに従って書き込み処理の衝突により性能が低下している．

読み込み処理性能では，書き込み時と同じ VIEW の場合では書き込みと同様にノード数に対してスケーラブルなアクセス性能が得られた．書き込み時と VIEW が異なる場合の性能は，VIEW が同じ場合と比べ半分程の性能であった．しかし PVFS2 と比較した場合にはどちらも MPI-IO/Gfarm の方がよい性能を得た．

今後の課題としては，書き込み時に示したノード内でプロセス数が増加した場合の書き込み処理の競合による性能低下を防ぐ手法が必要であると考えられる．1 つのプロセスが代表して書き込みを行うか，排他制御を行い同時に 1 つのプロセスしか書き込みを行わない仕組みを用意しノード内のプロセス間で協調した書き込み処理を行う必要があると考えられる．

また，現在分散ファイルに対して MPI-IO を用いなければアクセス出来ない．より汎用的にアクセスを行えるようにするために FUSE を用いて分散ファイルに対して POSIX/API を用いたアクセスを可能にする必要があると考えられる．

参 考 文 献

1) PhillipM. Dickens and Jeremy Logan. Towards a high performance implementation of MPI-IO on the Lustre file system. In *Proceedings of GADA'08*, 2008.
2) Jean-Pierre Prost, Richard Treumann, Richard Hedges, Bin Jia, and Alice Koniges. MPI-IO/GPFS, an optimized implementation of MPI-IO on top of GPFS. In *Proceedings of Supercomputing '01*, pages 17–17, 2001.
3) Jonathan Ilroy, Cyril Randriamaro, and Gil Utard. Improving MPI-I/O performance on PVFS. In *Proceedings of Euro-Par '01*, pages 911–915, 2001.
4) Osamu Tatebe, Kohei Hiraga and Noriyuki Soda Gfarm Grid File System. *New Generation Computing*, Ohmsha, Ltd. and Springer, Vol. 28, 2010. (to appear)
5) Gfarm. http://sourceforge.net/projects/gfarm/
6) Rajeev Thakur, William Gropp, and Ewing Lusk. Data Sieving and Collective I/O in ROMIO. In *Proceedings of FRONTIERS '99*, page 182, 1999.
7) Rajeev Thakur and Ewing Lusk. An Abstract-Device Interface for Implementing Portable Parallel-I/O Interfaces. In *Proceedings of Frontiers '96*, pp.180–187, 1996.
8) John Bent, Garth Gibson, Gary Grider, Ben McClelland, Paul Nowoczynski, James Nunez, Milo Polte, and Meghan Wingate. PLFS: a checkpoint filesystem for parallel applications. In *Proceedings of SC '09*, pages 1–12, 2009.
9) Atsushi Hori, et al. An implemenation of an MPI-IO using Catwalk. *IPSJ SIG Technical Report Vol.2009-HPC-121 No.14*, 2009.