Vol.2011-ARC-197 No.34
Vol.2011-HPC-132 No.34
2011/11/29

並列ジョブのファイルI/Oをひとつのファイルに集約する方式の提案と予備評価

大 野 善 之[†1] 堀 敦 史[†1] 石 川 裕[†1,†2]

並列ジョブにおける多数ファイルに対する I/O をひとつのファイルに対する I/O に集約することによりファイル I/O を高速化する方式を提案する．各プロセスごとにファイルを作成し，データを書き出すというファイル I/O パターンをとるアプリケーションが多くある．しかし，現在普及している並列ファイルシステムは，少数の大きなデータ I/O で高い性能がでるように設計されており，プロセスごとにファイルを作成するという I/O パターンでは高い性能がでない．そこで，並列ジョブにおけるファイル I/O をひとつのファイルに集約し，少数の大きなデータ I/O にする方式を提案する．Lustre 上で予備評価を行った結果，並列プロセスがそれぞれ 1MB のファイルを 128 個 I/O する場合，2-3 倍の性能向上を確認した．

Proposal and preliminary evaluation of a mechanism for file I/O aggration to one file in a parallel job

YOSHIYUKI OHNO,[†1] ATSUSHI HORI[†1]
and YUTAKA ISHIKAWA[†1,†2]

In many parallel applications, each process creates process independent files. However, many existing parallel file systems are designed for large data I/O operations on a few files, and exhibit low performance for small data I/O operations while accessing lots of files. In our proposed approach, small file I/O operations are aggregated to have large I/O operations and converted to access one large file. Preliminary evaluation using Lustre shows that the parallel write/read of 1MB x128files per each process is 2-3 times faster than the original I/O.

†1 理化学研究所計算科学研究機構
RIKEN Advanced Institure for Computational Science
†2 東京大学
The University of Tokyo

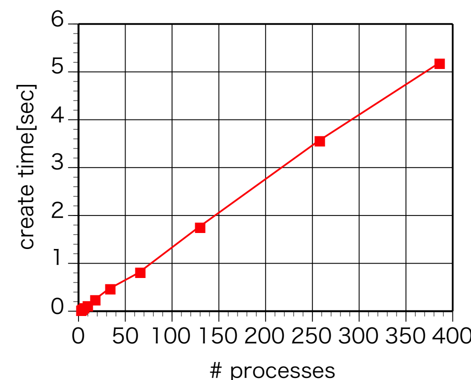

図 1 同時ファイル生成にかかる時間

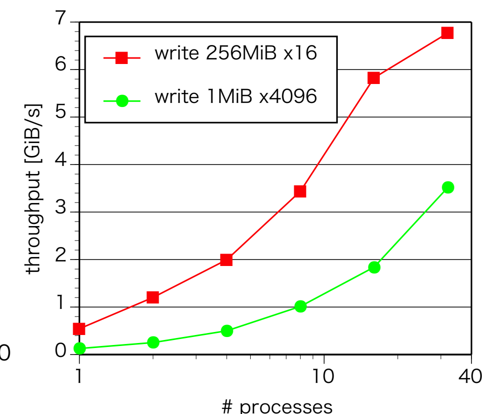

図 2 large I/O と small I/O のスループット差

1. はじめに

これからの HPC システムでは，演算コア数を増やしてトータルの計算性能を向上させる方向に進むことが予想されている[1)]．したがって，増大するノード数，およびコア数からのファイルアクセスに耐えうるような I/O システムの実現が望まれている．

現在の多くの並列アプリケーションの I/O パターンは，各プロセスごとにファイルを作成しデータを書き出すという I/O パターンである．しかし，一度に多数ファイルの I/O を行うようなファイル I/O パターンは，現状の並列ファイルシステムは高い性能を得られない．多くの HPC システムで採用されている Lustre[2)] や，Paralell Virtual File System(PVFS)[3)] は，大きなデータを I/O する場合に高い性能がでるように設計されているが，多くの小さなファイルを I/O する場合は，大きなデータを I/O する場合ほど性能が出ない．その理由は 2 点挙げられる，Metadata Server への負荷集中の問題と，Small I/O の問題である．

Lustre や PVFS は，1 つまたは少数の Metadata Server と多数の Storage Server からなる構成である．多数のファイルを I/O する場合は，ファイル数分のメタデータ操作が 1 つまたは少数の Metadata Server に集中し，性能が低下する．**図 1** は，同時に複数プロセスが，それぞれ個別の 1 個のファイル生成をするのに要する時間である．実験は 3 章の評価実験に

© 2011 Information Processing Society of Japan

用いたクラスタおよび Lustre 上で行っている．同時にするファイル生成を行うプロセス数に比例して，時間が増加していることが分かる．文献[4] では，実際に大規模な HPC システムで同様の実験をしており，Lustre 上で 12K プロセスによるファイル生成に約 5 分かかったと報告されている．また，複数の Metadata Server を持つ構成である General Parallel File System(GPFS)[5] 上でも計測しており，256K プロセスによるファイル生成で約 33 分かかったと報告されている．このように，既存の並列ファイルシステムにおいては，同時に多数のファイルを作成するのに，O(N) の処理時間がかかる．

既存の並列ファイルシステムは 1 度に大きなファイル I/O をする場合に，高い性能を達成するように設計されている．**図 2** は，Lustre 上で並列プロセスがプロセスごとに 4GB のファイルを書き出した場合のスループットである．1MiB の write を 4096 回した場合と，256MiB の write を 16 回した場合を比較している．大きなファイル I/O を並列に行った場合，スループットが大きくなることが分かる

本稿では，並列アプリケーションの各プロセスがそれぞれ個別の多くの小さなファイル I/O を行う場合，高い I/O 性能を得るためには，アクセスするファイル数を少なくすること，1 回あたりのファイル I/O 操作のデータサイズを大きくすることの，2 点が必要であると考える．そして，並列アプリケーション側では，プロセス個別に多くのファイル I/O を行っているように見せながら，並列ファイルシステム側には，1 つのファイルに対する大きな I/O とするファイル集約方式を提案する．Lustre 上で予備評価では，並列プロセスがそれぞれ 1MiB のファイルを 128 個 I/O する場合，提案方式により 2-3 倍の性能向上することを確認し，有効性を示した．

2. 提 案 方 式

2.1 ファイル集約方式の概要

我々が提案するファイル集約方式は，**図 3** のような集約ライブラリとして実現する．各プロセスは，集約ライブラリを経由してファイル I/O を行う．集約ライブラリでは，各プロセスからのそれぞれ異なるファイルに対する I/O 要求を，1 個の集約ファイルに対するファイル操作に変換する．また，集約ライブラリでは，metadata や，プロセスの write data をバッファリングし，バッファしたデータをまとめて，集約ファイルに対して書き出す．これにより，並列アプリケーションが I/O するファイル数を 1 つに削減でき，同時にファイルシステムに対する I/O の単位を大きくすることができる．

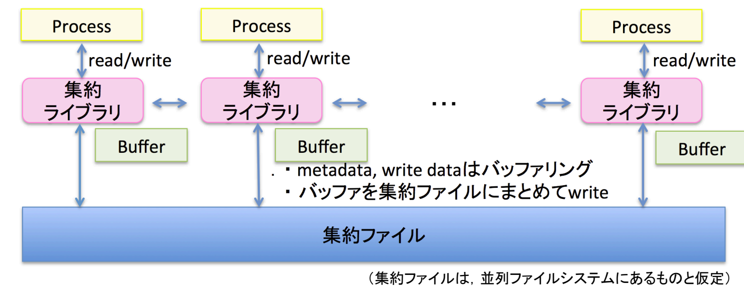

図 3 ファイル集約方式の全体像

2.2 集約ファイルレイアウト

集約ライブラリと集約ファイル間の I/O 単位を “Object Block” と定義し，集約ファイルは，Object Block の列で構成する．Object Block のサイズを，並列ファイルシステムのストライプサイズにすることで，集約ライブラリは，ストライプごとに I/O を行うため効率がよくなると考えられる．

図 4 は，提案する集約ファイルのデータレイアウトである．1 つの Header Object Block(図中 “H”) が先頭にあり，後ろに Metadata Object Block(“M”)，Data Object Block(“D”) が続く．

Header Object Block には，集約ファイルそのものに関する情報が格納される．格納する情報は，Object Block のサイズ，後ろに続く Metadata Object Block や Data Object Block の数とレイアウトである．

並列ジョブのプロセス P_i が作成するファイルに関する全ての情報は，Metadata Object Block M_i と Data Object Block D_i に格納される．各ファイルの実体は，Data Object Block に格納される．Metadata Object Block には，ファイル名を index として参照できるメタデータを格納している．メタデータには，ファイルのサイズ，集約ファイル内のオフセット位置を記録している．また，Metadata Object Block では，各ファイルのメタデータだけでなく，Data Object Block の利用状況も保持している．

2.3 実 装

ファイル集約ライブラリでは，ファイル操作に最低限必要な create, open, write, read, lseek, close, sync，および，ライブラリの初期化・終了処理のための，initailize と finalize を実装した．

ⓒ 2011 Information Processing Society of Japan

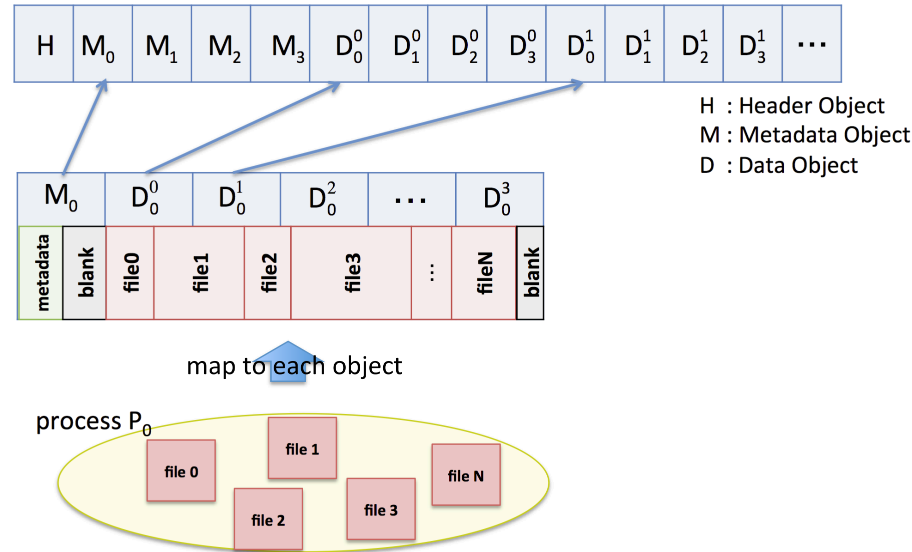

図 4 集約ファイルのデータレイアウト

各プロセスで，Metadata Object Block と Data Object Block 用のバッファをメモリ上に 1 つずつ持つようにした．initalize の際には，集約ファイルの Metadata Object Block を read してバッファにコピーしておき，metadata を扱う操作は，メモリ上のバッファにアクセスするようにし，並列ファイルシステムに対して毎回アクセスをしないようにした．

ファイルの実体においては，バッファより小さいサイズの write であれば，並列ファイルシステムに write せずに，バッファにストアするようにし，バッファが Object Block のサイズまでたまってから，まとめて並列ファイルシステムに書き出す．

3. 評価実験

前章で提案した，ジョブ内の全てのファイル I/O を 1 つファイルに集約する方式の効果を確認するために，Lustre ファイルシステム上で評価実験を行った．

3.1 実験環境

実験環境を**図 5** に示す．32 ノードの計算ノードからなる計算ノード群，および，合計 10PB

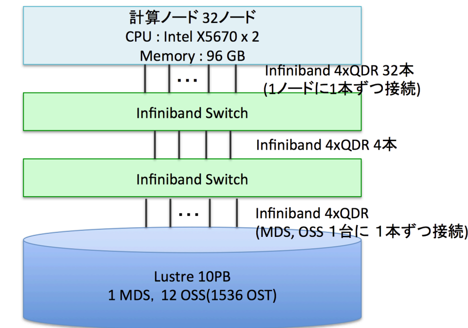

図 5 実験環境

の Lustre ファイルシステムで構成される．計算ノード群は，各ノードが Infiniband 4xQDR のケーブルが 1 本ずつ Infiniband Switch に接続する構成である．各計算ノードは， Intel Xeon Processor (X5670，2.93GHz，6cores) を 2 ソケット持ち，96 GB のメモリを有している．Lustre ファイルシステムは，1 台 MDS，および 12 台の OSS が 1 つの Infiniband Switch に接続する構成である．各 OSS は，128 台の OST が接続しており，系全体で 1536 台の OST が利用できる．また各 OSS は，192GB のメモリを有し，メモリにデータがキャッシュされる．Lustre ファイルシステムを構成するサーバ群が接続する Infiniband Switch と，計算ノード群が接続する接続する Infiniband Switch は，4 本の Infiniband 4xQDR で接続されている．

3.2 実験内容

Create 時間 および write と read のスループットの測定を，同時にファイル I/O を行うプロセス数を 1 から 32 まで変更して実施した．**表 1** は，測定にあたり変更したパラメタである．ファイル集約をせずに各プロセスが個別のファイルにアクセスする場合は，ストライプ数を増やしても並列性を確保できないため小さめの値 4 および 64 をとった．また，ファイル集約して 1 つのファイルにまとめる場合は，ストライプ数を増やして並列性を確保できるように大きい値 160 および 64 をとった．160 という値は，Lustre の仕様による最大値である．ストライプサイズは 16MiB で固定とした．1 ファイルのファイルサイズは，ストライプサイズより小さい 1MiB，ストライプサイズと同じ 16MiB の 2 種類とし，1 プロセスあたりのファイル数は 128 とした．

図 6 は，実験で用いた測定コードである．1 回のジョブで，10 回の測定をし，最大と最小

© 2011 Information Processing Society of Japan

表 1 実験パラメタ

	集約なし (normal)	集約あり (aggregate)
並列プロセス数	1〜32	
ストライプサイズ	16MiB	
ストライプ数	4, 64	64, 160
1 ファイルのサイズ	0B(create), 1MiB, 16MiB	
1 プロセスあたりのファイル数	128	

をとりのぞいた 8 回の試行の平均を結果とした．ジョブごとに異なるディレクトリを作り，create や write の実験では，対象のディレクトリ内にフラットにファイルを作成した．read の実験では，write の実験で作成したファイルを read している．なお，ファイル集約を行う場合は，1 回の試行ごとに別の集約ファイルを利用しており，測定時間には，集約ファイルの作成や，initialize 処理および finalize 処理も含むようにした．実験にあたっては，ページキャッシュによる影響がでないようにするために，全ての write の実験を完了させたのちに，計算ノード，MDS，OSS のキャッシュをクリアしてから，read の実験を行っている．

```
for( i = 0; i < 10; i++ ) {
    // Barrier (測定開始)
    for( j = 0; j < filenum_per_proc; j++ ) {
        open()  ;  // filename is "file_rank_i_j"
        write() ;  // or read()
        close() ;
    }
    // Barrier (測定完了)
}
```

図 6 測定用プログラムの疑似コード

3.3 実験結果

3.3.1 Create

図 7 は，create の結果である．横軸にプロセス数，縦軸にプロセス数 × 128 のファイルを create するのに要した時間をとっている．normal がファイル集約しなかった場合で，aggregate がファイル集約した場合である．ファイル集約の効果によって，ストライプ数 4 でファイル集約しない場合に比べて約 10 倍，ストライプ数 64 でファイル集約しない場合に比べて約 500 倍，速くなっている．ファイル集約の場合のファイル create にかかるコストは十分小さいと言える．

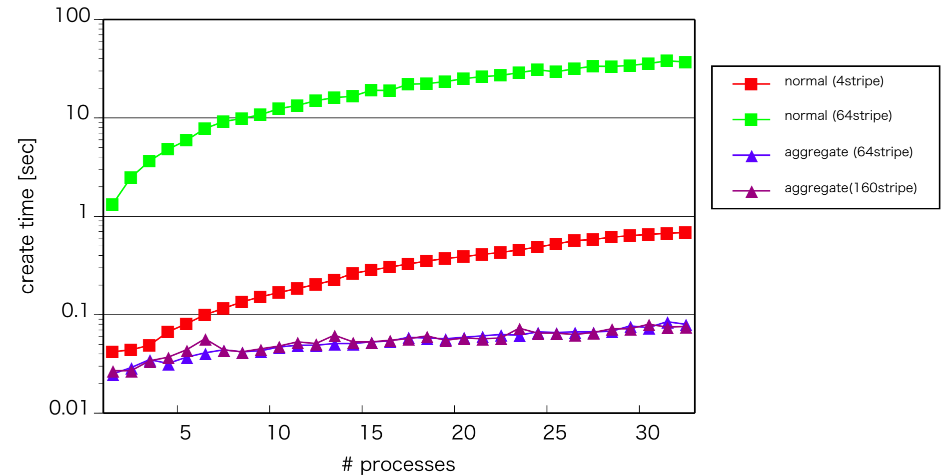

図 7 create 時間

3.4 Write

図 8 は，1MiB のファイルをプロセス数 × 128 個 write した場合のスループットである．ファイル集約の効果によって，ファイル集約しない場合のうち性能が良いほうのストライプ数 4 と比べても，約 2 倍スループットが向上している．この性能向上は，MDS に対する create 操作の負荷の削減，および，ストレージに対する I/O の集約の，両方の効果があらわれているからであると考えられる．

図 9 は，16MiB のファイルをプロセス数 × 128 個 write した場合のスループットである．ファイル集約をしない場合は，ファイルの create コストが大きいことと，ストライピングの効果が得られないことにより，大きなスループットを得ることができない．ファイル集約をすることでファイル集約をしない場合と比較して約 2 倍スループットが向上している．これは，ファイル作成のコストが削減できているためであると考えられる．しかし，ファイル集約をしていても，プロセス数が増えるにつれてストライプ数が 64 の場合は，性能が頭打ちになっている．これは，1 つのファイルに全ての I/O を集約することにより，書き出す OST を 64 個に限定することになってしまい，64OST の性能の上限に達してしまっているためであると考えられる．

ⓒ 2011 Information Processing Society of Japan

Vol.2011-ARC-197 No.34
Vol.2011-HPC-132 No.34
2011/11/29

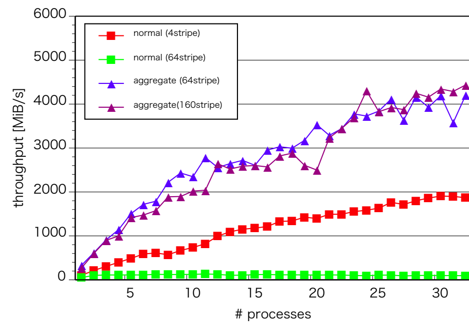

図 8 write スループット (1MiB/file)

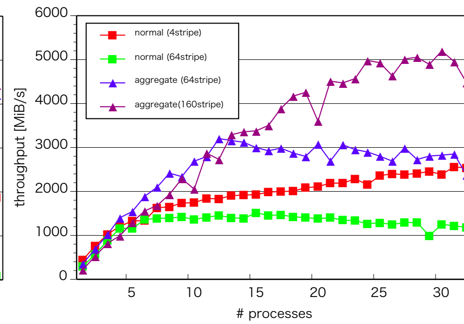

図 9 write スループット (16MiB/file)

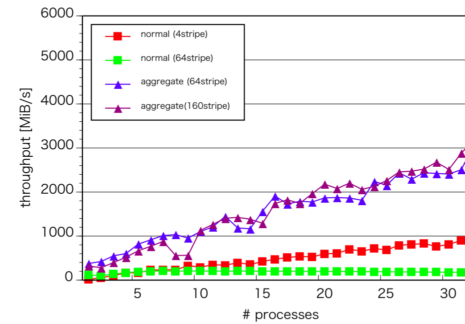

図 10 read スループット (1MiB/file)

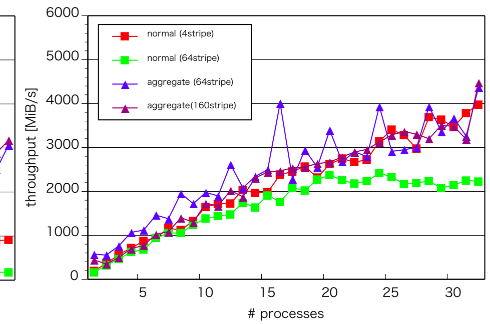

図 11 read スループット (16MiB/file)

3.5 Read

図 10 は，1MiB のファイルをプロセス数 × 128 個 read した場合のスループットである．ファイル集約をすることにより，ファイル集約しない場合のうち性能が良いほうのストライプ数 4 と比べても，約 3 倍スループットが向上している．**図 11** は，16MiB のファイルをプロセス数 × 128 個 read した場合のスループットである．こちらは，ファイル集約の有無の違いがほとんど見られない．

Write はバッファリングによってスループット性能が向上していたが，read はつねにファイルシステムにアクセスして data を読む実装にしている．それでもファイルサイズが 1MiB の場合のスループットが向上しているのは，object block 内で連続に配置されるファイルを読むときに，ストレージに対して連続アクセスできているためであると考えられる．

3.6 実験結果のまとめ

我々が提案するファイル集約方式では，多数の小さいファイルを I/O するような I/O パターンにおいて，I/O 性能が向上した．この理由は，1 つのファイルに対するアクセスとすることで metadata server への負荷が集中するというボトルネックを回避できたことと，write するデータをバッファリングすることで Storage Server に対して大きなデータ単位で I/O を行うことができたためである．

しかし，集約ファイルのストライプ数が小さいと，ファイルを格納する OST の数を少なく制限することになり，プロセス数が増えるにしたがって性能が頭打ちになってしまうという欠点があることが分かった．大規模な並列ファイルシステムにおいては，多数の OST を保有しながらも，ストライプ数の最大値は OST の数より少なく設定されており，現状の方式では性能が向上しない可能性がある．

上述の問題を解決するには，1 つのファイルだけに集約するのではなく少数のファイルに分割し，利用できる OST を増やせばよいと考えられる．しかし，集約ファイルを分割するにあたって，集約ファイルの分割の仕方から，分割の個数など検討をする必要がある．これらの検討が今後の課題となる．

4. 関 連 研 究

文献[6] では，並列アプリケーションが 1 つのファイルに対する I/O にしないのは，並列ファイルシステムのストライピングによる並列 I/O 性能が得難いこと，データの一貫性のために各プロセスがロックをとる必要があるということを問題視しているためであると述べている．上記問題を，アプリケーションプログラマが気にせずとも，1 つのファイルに対する I/O 操作を記述できる PFA(Parallel File Aggregation) Mechanism を提案している．プロセスごとにファイルを作成する I/O パターンであったプログラムを，PFA Mechanism の API を用いて 1 つのファイルに対するアクセスパターンをとるプログラムに記述しなおしたところ，3.8 倍の I/O 性能向上を達成したと報告している．

文献[4] は，我々の提案と同じようにプロセスごとにファイルを作成する I/O パターンのまま，1 つのファイルに集約させるアプローチをとり，ファイル集約ライブラリである SIONlib を実装している．SIONlib は，並列アプリケーションの各プロセスが collective に個別のファイルを作成する I/O パターンを想定している．

文献[7] では，並列アプリケーションの各プロセスがそれぞれ個別のファイルに対する操作

© 2011 Information Processing Society of Japan

を，1 つの group file に対する操作として記述する group file operation を提唱し，スケーラブルに group file operation を行うことができる 分散ファイルシステム TBON-FS を提案している．複数のファイルをまとめて操作する場合の有効性を示している．しかし，それぞれ個別のファイルに別々の操作を行うためには，個別のファイルを個々に操作する必要がある．

本研究の貢献は，並列アプリケーションの各プロセスがそれぞれ個別の多くの小さなファイル I/O を行う場合を想定し，並列アプリケーション側では，プロセス個別に多くのファイル I/O を行っているように見せながら，並列ファイルシステム側には，1 つのファイルに対する大きな I/O とするファイル集約方式を提案し，有効性を示したことにある．

5. ま と め

本稿では，並列ジョブのファイル I/O を 1 つのファイルに対する I/O に集約する方式を提案し，Lustre 上で有効性の評価を行った．

提案方式では，各プロセスが 1MiB のファイルを 128 個 I/O する場合において，集約を行わない場合と比べ write で 2 倍，read で 3 倍の性能向上することを確認した．ストライプサイズ程度の 16MiB ファイルを 128 個 I/O する場合でも，read の性能は変わらなかったが，write の性能が 2 倍向上した．多数の小さいファイルを I/O するような I/O パターンにおいて，提案するファイル集約方式が有効であると言える．

しかしながら，大規模な並列ファイルシステムにおいては，ストライプ数の最大値は OST の数より少なく設定されており，プロセス数が増えると，限られた OST 数の性能の上限に達してしまい，性能が低下する欠点も分かった．今後は，集約することによる欠点を解決する仕組みを検討し，ストライプサイズ以上のファイルの I/O に対しても性能向上をはかる．

謝辞 本研究は，科学技術振興機構 (JST) の戦略的創造研究推進事業「CREST」における研究領域「ポストペタスケール高性能計算に資するシステムソフトウェア技術の創出」によるものである．本研究の評価実験には，東京大学情報基盤センター若手・女性研究者支援プログラムで提供される並列コンピュータおよびストレージを利用した．

参 考 文 献

1) Dongarra, J. et al.: The International Exascale Software Project roadmap, *Int. J. High Perform. Comput. Appl.*, Vol.25, pp.3–60 (online), DOI:http://dx.doi.org/10.1177/1094342010391989 (2011).
2) Lustre File System (online), available from ⟨http://wiki.lustre.org/index.php⟩.
3) Parallel Virtual File System (online), available from ⟨http://www.pvfs.org/⟩.
4) Frings, W., Wolf, F. and Petkov, V.: Scalable massively parallel I/O to task-local files, *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, SC '09, New York, NY, USA, ACM, pp.17:1–17:11 (online), DOI:http://doi.acm.org/10.1145/1654059.1654077 (2009).
5) General Parallel File System (online), available from ⟨http://www-03.ibm.com/systems/software/gpfs/index.html⟩.
6) Kato, J. and Ishikawa, Y.: Design and implementation of parallel file aggregation mechanism, *Proceedings of the 1st International Workshop on Runtime and Operating Systems for Supercomputers*, ROSS '11, New York, NY, USA, ACM, pp.33–40 (online), DOI:http://doi.acm.org/10.1145/1988796.1988802 (2011).
7) Brim, M. and Miller, B.: Group file operations for scalable tools and middleware, *High Performance Computing (HiPC), 2009 International Conference on*, pp.69–78 (online), DOI:http://dx.doi.org/10.1109/HIPC.2009.5433223 (2009).

© 2011 Information Processing Society of Japan