社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2009－ARC－182 (8)
2009－HPC－119 (8)
2009／2／26

ファイルサーバー独立な並列ファイルキャッシュ機構

太田一樹[†] 石川裕[†,††]

CPU の計算能力の増大に伴って、大規模なデータを扱う科学計算アプリケーションが増加している。しかしハードディスクのバンド幅が計算速度に追い付いておらず、ディスク I/O がアプリケーション全体のボトルネックになってしまうケースがある。

この計算と I/O のギャップを埋めるために、並列ファイルキャッシュシステム pdCache を提案する。pdCache は計算プロセスから発行された I/O 要求を受け取り、それを元に並列ファイルシステムに対する操作を行う。書き込みに対してはメモリ上に一度キャッシュされ遅延されて書きこまれ、読み込みに関しては二度目に同じ領域に対する要求が発生した場合は並列ファイルシステムに対する操作は発生しない。pdCache はバックエンドの並列ファイルシステム・ネットワークレイヤーと独立に設計され、様々な環境で使用できる。

File-Server Independent Parallel File-Cache System

Kazuki Ohta[†] and Yutaka Ishikawa[†,††]

By an increase of the CPU speed, scientific applications become to handle large dataset. However, the bandwidth of the disk is too narrow compared to the compute speed, the I/O becomes the major bottleneck of the entire application.

To solve this gap between parallel computation and I/O performance, we propose a parallel file-caching system, called pdCache. pdCache receives the I/O requests from the compute processes, and does the actual operations to the backend parallel file systems. For a write request, the data is temporarily cached in the memory, and lazily written to the parallel file system. For a read request, the data is cached. If another process requests the same region, no requests are issued to the parallel file system. pdCache is designed independently from the backend parallel file system and the underlying network, so it can be used in various cluster environment.

1. はじめに

CPU の演算能力の増大に伴って、アプリケーションが扱うデータ量が年々増加しており、テラバイト級のデータを扱うこともある。しかし、そのようなサイズのデータを扱うには単一ディスクの I/O バンド幅は狭すぎるため、複数のディスクを単一のファイルシステムとして扱うための研究が数多くなされてきた[2),6),14),15)]。また、集団的 I/O のようにクライアントプロセス側で I/O リクエストを最適化する手法も数多く提案されている[9),18)]。

最近ではマルチコア CPU が一般的になったことで計算ノード上で実行される計算プロセスの数が増大しており、クラスタ全体で数十万コアを持つようなケースもある。もし仮にこれら全てのプロセスが一斉に I/O を行った場合、I/O ノードは大量のクライアントからのリクエストを同時に受け取ってしまう。結果的にディスクのシーク遅延や並列ファイルシステム内でのロック回数, 通信回数が増大し、バンド幅が十分に出ない[12)]。またクライアントと並列ファイルシステム間でネットワーク輻輳が発生し、十分に性能が出ないという問題もある。全てのプロセスが I/O ノードに要求を送信した場合、I/O ノード側のスイッチのバッファが溢れてしまうからである[13)]。更に、各プロセスが同時に異なるファイルを生成あるいは読み込む場合、メタデータサーバーへの負荷が集中し、その部分がボトルネックになるケースも有る。以上のような問題が原因で計算速度に見合った I/O バンド幅が得られず、I/O がアプリケーション全体のボトルネックとなってしまう。

このような問題を解決するため、本研究ではファイルサーバー独立な並列ファイルキャッシュ機構 pdCache を提案する。このシステムでは計算プロセスと並列ファイルシステムの間に複数のキャッシュサーバーを

† 東京大学大学院情報理工学系研究科
Graduate School of Information Science and Technology, The University of Tokyo
†† 東京大学情報基盤センター
Information Technology Center, The University of Tokyo

配置することで、並列ファイルシステムへのアクセスを緩衝し、I/O の最適化を行う (図 1)。

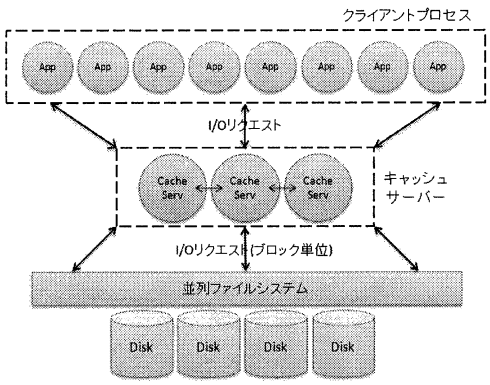

図 1 全体システム構成

pdCache では書き込み要求については一端オンメモリの領域に書かれる。書かれたデータはバックグラウンドスレッドによって並列ファイルシステムに書き出される。ここではブロック単位での入出力になるため、クライアントが直接書き込み要求を行うのに比べると、要求数の削減が期待できる。また全てのプロセスが同じファイルに対して書きこむような場合は、open, close 時に発生するメタデータリクエストの回数も削減する事が出来る。通常は各プロセスが open や close を呼び出していた所が、各キャッシュサーバーが発行するのみになるからである。

一度読み込まれた箇所についてはメモリ上にキャッシュされる。同じ領域を他のプロセスが読む場合には、キャッシュされたデータが読み込まれるため、アプリケーション起動時に全プロセスが同じファイルを読み込む場合などでは効果を発揮する。

pdCache では読み込み、書き込み両方に対してキャッシュの一貫性を保つことで、一部のプロセスが同じブロックを読み書きする場合にもキャッシュの効果が働くようにする。このようにすることで科学計算アプリケーションだけではなく、ホームディレクトリに置かれたファイルへのアクセス等にも対応できる。

さらに様々なクラスタ環境で動作するように、インターコネクトネットワークの種類, バックエンドの並列ファイルシステムの種類に依らないような設計/実装を行う。

2. 設　　計

pdCache は、クライアントライブラリとキャッシュサーバーの 2 つから構成される。クライアントライブラリは POSIX I/O と同じインターフェースを持っており、read/write などの要求をキャッシュサーバーに対して発行する。キャッシュサーバーは発行された I/O 要求を実際に処理する。

キャッシュサーバーのソフトウェア構造を図 2 に示した。pdCache では下位のネットワーク、並列ファイルシステムについて独立させるために、BufferedMessage Interface (BMI[5]) と Abstract-Device Interface for I/O(ADIO[16]) という 2 つのソフトウェアレヤーを用いる。

BMI は PVFS[6] に含まれる並列 I/O 向けのネットワークアブストラクションレイヤーで、TCP/IP, Myrinet[4], Infiniband[1] など、各種ネットワーク上で透過的に通信を行うことが出来る。BMI には以下のような特徴がある。

(1) 単純なメッセージベース API
(2) 全てノンブロッキングで動作
(3) MPI に非依存

ADIO は ROMIO[17](MPI-IO の実装) に含まれるもので、各種並列ファイルシステムに対して透過的にアクセスするためのレイヤーである。ADIO 以下には PVFS2, PGFS, Lustre など主要なファイルシステムについてのモジュールが一通り実装されている。ROMIO を使用した MPI 実装 (MPICH2[11], MVAPICH[3], 等) ではアプリケーションが MPI-IO の関数を呼ぶと、下位のファイルシステムを判別して適切な ADIO モジュールが使用されるようになっている。

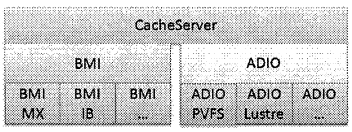

図 2 キャッシュサーバーのソフトウェアスタック

Myrinet と PVFS2 を使用した場合、システム全体の構成は図 3 のようになる。BMI はクライアント, キャッシューサーバー間の通信とキャッシューサーバ間同士の通信の両方に用いられる。

2.1 Cache Page Access

クライアントがキャッシュにアクセスする際は、まずいずれかのキャッシュサーバーにアクセスする。この最初にアクセスされるキャッシュサーバーを Coordinator と呼ぶことにする。どの Coordinator に接続するかは、完全にランダムか、またはクライアントノードの IP やランクから決定する。Coordinator はクライアントの要求を受け取り、返事を生成するまでの処理を引き受ける。クライアント側の処理を出来る限り Coordinator に任せているのは、出来る限り計算プロセスへのパフォーマンス的影響を少なくするためである。計算プロセスから見ると、自分が行おうとしている I/O リクエストを Coordinator に委譲するような形になっている。

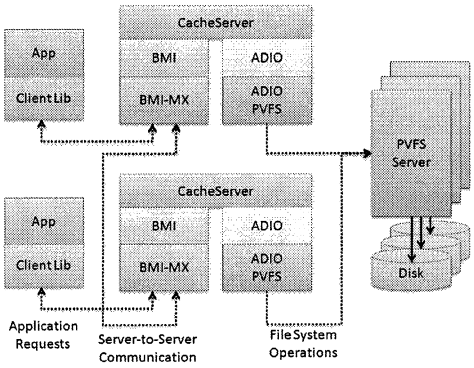

図 3　システム構成 (Myrinet + PVFS)

次に、Coordinator は要求されたページに関するメタデータを保持するキャッシュサーバー (MetadataHolder) に対する問い合わせを行う。pdCache ではファイルは同じサイズのページに分割されており、各ページのメタデータはファイルパスのハッシュ値とオフセットを基に、ラウンドロビン形式で全キャッシュサーバーに対して分配される。あるページのデータは複数のサーバーに保持される可能性が有るが、メタデータは必ず 1 つのサーバーだけに存在する事を保証する。

メタデータにはファイルパス、オフセット、キャッシュを保持しているノード、キャッシュの状態、アクセス用のロックが含まれる。キャッシュページは NotCached, Modified, Clean の 3 種類の状態を遷移する。NotCached はページがキャッシュ内に存在しない状態、Modified はキャッシュされたページの内容と並列ファイルシステム上のページ内容が異なる状態、Clean は内容が一致している状態である。図 4 は各状態間の遷移を示している。

最後に、Coordinator は MetadataHolder からページの情報を取得した後、そのページのデータを実際に保持しているキャッシュサーバー (DataHolder) にアクセスする。データが存在しない場合は並列ファイルシステムに対して直接操作を行う。

以上がページに対する基本的なアクセス方法となる。次に、読み込み、書き込みの操作が起こった場合のより詳しいシステムの動きを説明する。

2.1.1 Read Access

キャッシュに対して読み込みアクセスを行う際は図 5 のようになる。Coordinator, MetadataHolder, DataHolder をそれぞれ C, M, D で示している。

(1) Coordinator は MetadataHolder にアクセスし、ReadLock を獲得する。

(2) MetadataHolder は返事として、要求されたページのデータを保持するサーバー (DataHolder) の位置を返す。複数のサーバーがデータを保持している場合、MetadataHolder がその内の 1 つを選択する。DataHolder の選択基準としてはランダム、ラウンドロビン、LRU などが考えられる。

(3) Coordinator は該当 DataHolder に対してアクセスを行い、ページのデータを受け取って、自サーバーにキャッシュする。DataHolder がいなければ、Coordinator はバックエンドの並列ファイルシステムからデータの読み込みを行い、自サーバーにキャッシュする。

(4) MetadataHolder に再度アクセスして ReadLock を解放した後、データをクライアントに返す。

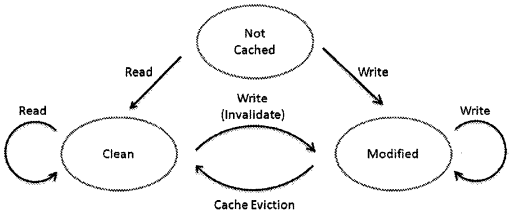

図 4　キャッシュページの状態遷移図

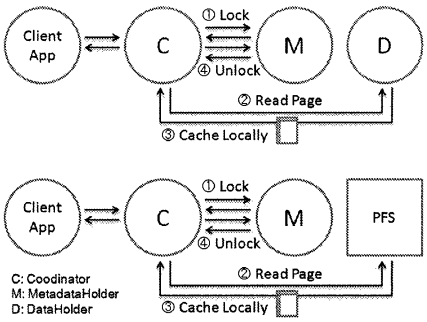

図 5　読み込み (上図: キャッシュにヒットした場合, 下図: ミスヒットの場合)

2.1.2 Write Access

pdCache に対して書き込みアクセスを行う際は図 6 のようになる。

(1) 読み込み時と同じように、Coordinator はまず MetadataHolder にアクセスし、WriteLock を獲得する。

(2) MetadataServer は返事として、要求されたページを保持する全ての DataHolder の位置を返す。

(3) DataHolder が存在すれば、どれかの DataHolder からページの内容を受け取った後、各 DataHolder に対してキャッシュの Invalidate 命令を発行する。そして受け取ったページに対して変更を施し、自サーバーにキャッシュする。DataHolder がいなければ、並列ファイルシステムからデータを読み込んでページを取得し、

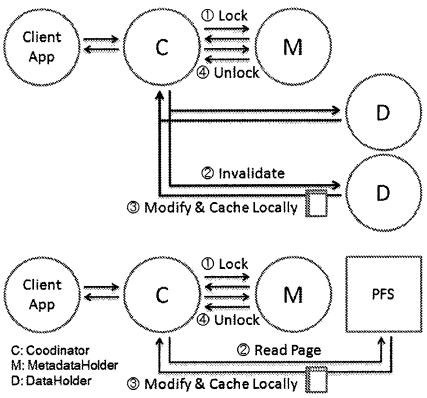

図 6 書き込み (上図: キャッシュにヒットした場合, 下図: ミスヒットの場合)

変更を加えて自サーバーにキャッシュする。

(4) MetadataHolder のロックを解放し、データをクライアントに返す。

2.2 Cache Page Eviction

pdCache ではキャッシュサイズについて、LowWaterMark と HighWaterMark を設定する。

各サーバーはキャッシュ破棄用のスレッドを持っており、LowWaterMark を超えるとそのスレッドは自分の保持するページの破棄を行い始める。破棄する際は、該当ページの MetadataHolder のロックを取ってから破棄を行う。

破棄スレッドがロックを確保する際には、ロック要求を送ってからパケットが届くまでの間に他のプロセスが該当ページの内容を Invalidate してしまう可能性が有るので、その点については留意する必要がある。

もし破棄対象のページの状態が Modified ならば、並列ファイルシステムに対して書き込みを行ってから破棄を行う。破棄するページを決定するアルゴリズムには、LRU などが考えられる。

HighWaterMark を超えると、その DataHolder に対する新しいページのアクセスは拒否される。クライアントはキャッシュ破棄スレッドがキャッシュページを破棄するまで待たされる事になる。

キャッシュアクセスが来た際にキャッシュを破棄するのではなく、キャッシュ破棄用スレッドで破棄を行っているのには理由がある。キャッシュアクセス時に破棄を行うと、アクセスしたいページ (P1) に関するロックを取得した後、破棄をする対象のページ (P2) に関するロックも獲得する必要がある。この時、別のサーバーにおいて P2 がアクセスされて P1 を破棄しようとすると、P2, P1 という順番でロックが取得される。ここでデッドロックが起こる場合があるので、破棄操作は別のスレッドで行っている。

2.3 議　　論

2.3.1 メッセージ配信方式の最適化

現在の設計では全ての操作の最初と最後で lock, unlock を行っているが、ユーザーに対してデータを送ってから unlock メッセージを送るなどの最適化が必要であると思われる。特にノード数が多くなってくるとロックのコンテンションが多くなるので、ロック時間を出来るだけ短くする必要がある。

2.3.2 メタデータのメモリ使用量

まず、メタデータが消費するメモリオーバーヘッドの問題を考える。ページサイズを P, HighWaterMark を H, 1 ページあたりのメタデータが消費するメモリ使用量を MPM とすると、最大で $(H/P)*MPM$ のメモリをキャッシュサーバーが必要とする事になる。

特に小さいサイズのページサイズを選択した場合、非常に多くのメモリを消費してしまう事があるので注意が必要であると考えられる。キャッシュを保持しているノードに関しては、ノード数 bit のビットベクターで表現出来る。キャッシュの状態については 1bit で表現する。その他ファイルパス、オフセット、ロックに関しても持つ必要がある。

また、ページに対して何か付随情報を追加する場合はさらにメモリ使用量が増加する。例えば、複数のノードが同じデータを保持している場合、MetadataHolder はその内の 1 つを選択する必要がある。この選択基準についてアクセスされた順番や回数等を記憶するとなると、それ相応のメモリが必要となる。

3. 関 連 研 究

3.1 IO Delegate Cache System

Wei ら[12] は、I/O delegate ノードという I/O キャッシュ専用のノードを追加で使用する事で、I/O 性能を向上させるシステム IODC (I/O Delegate Cache System) を提案した。I/O delegate ノード群は、アプリケーションからは MPI のコミュニケーターとして認識される。

アプリケーションが MPI-IO の関数を利用して I/O を行うと、自動的に I/O delegate ノードを介して並列ファイルシステムにアクセスされる。これは ADIO レイヤーに IODC にアクセスするためのモジュールを追加する事で実現されている。

I/O delegate ノードではデータのキャッシングが行われる。評価では BTIO, FLASH I/O の 2 つのベンチマークの他に S3D というアプリケーションを用い、アプリケーションプロセス数の 10%を追加で I/O delegate ノードとして使用した場合、最大で約 500%の書き込みバンド幅を出す事に成功している。

IODC は MPI に依存したシステムであり、MPI-IO のインターフェースを通してのみアクセスできる一方、pdCache では MPI から独立したデーモンとして実装

する事で、各種既存のアプリケーションからも利用できる汎用的なキャッシュシステムとして使用される事を想定している。

また、IODC では同じデータは必ず 1 ノードにしか置けないような形になっており、pdCache のように複数ノード間で同じデータをコヒーレンスを保ちながら共有するという事は行われていない。pdCache ではこれを行う事でステージインのように同じファイルを全員で読むような操作について、IODC に対する優位性を持つ事が出来ると考えている。

3.2 ZOID

ZOID[10] はファイル操作とソケット操作を計算ノードから分離させるための I/O 基盤である。IBM BlueGene/L ではメモリ使用量, システム複雑度, システムノイズ (ジッター) の削減のために、計算ノードでは I/O 機能を持っていない軽量 OS が使用されている。そのため、全ての I/O リクエストは I/O ノードに転送され、そこで実際の I/O 操作が行われる。

I/O ノードではカスタマイズされた Linux が起動しており、計算ノードから受け取った I/O リクエストを並列ファイルシステムに対して発行する。実際には I/O ノードでは ZOID デーモンと呼ばれるプロセスが起動しており、このデーモンがリクエストの処理を受け持つ。以上のような仕組みにより、高性能な計算能力と I/O を実現している。

評価では IBM から提供されている同等品との性能比較を行いパフォーマンスの向上が見られたほか、LOFAR と呼ばれる天体望遠鏡のリアルタイム観測システムを構築して、システムのパフォーマンスと柔軟性が確認されている。

ZOID デーモンでは、受け取ったリクエストに対するスケジューリングやデータのキャッシングのようなことはしておらず、そのまま並列ファイルシステムに対して操作を行うような形になっている。また、pdCache では ZOID と同じように計算ノードでは複雑な操作は行わず、Coordinator にデータ操作を委譲するような形になっているため、計算プロセスへの介入が少なくアプリケーションへの影響を最小限に出来ると考えている。

3.3 DWC^2

DWC^2 は Object Storage Device 向けの協調的キャッシュシステムである[19]。オブジェクトに対して設定されたウェイト値を利用する事で、どのオブジェクトをキャッシュから破棄するかを決定する。キャッシュシステムは I/O ノードで動作し、各ノードでは Object Caching Tree と呼ばれるツリー構造でキャッシュされているオブジェクトのデータと重みを管理している。自ノードのキャッシュを破棄する際には、他に空いているノードが有ればそちらにデータを移譲する事で複数のノード間で協調的にキャッシングを行う。

評価では Auspec ファイルシステムトレース[7] を用い、多人数で共有されている NFS ストレージの使用パターンについて、LRU や LFU よりもキャッシュヒット率が格段に向上する事が確認されている。

DWC^2 ではキャッシュの一貫性についての考慮が少なく、現状では read アクセスの最適化のみを対象にしている。pdCache でもあるノードにのみデータが偏った際には他のノードにデータを移すなどの操作を行う事でキャッシュヒット率を上げられる可能性がある。特にシステム全体で 1 つしかコピーが無いようなデータを破棄するよりは、複数のノードで持たれているようなデータを破棄した方がキャッシュのヒット率が向上すると思われる。これを実現するためのアルゴリズムとして、N-Chance Forwarding[8] などが提案されている。

4. おわりに

本稿では下層のファイルシステムレイヤー・ネットワークレイヤーに独立な並列ファイルシステムキャッシュシステム pdCache の設計を述べた。主に BMI・ADIO を使用した設計、キャッシュの一貫性を保つためのページの状態遷移、それを引き起こすメッセージの通達方式を述べた。

今後の課題としては、実際のシステムとして動作させ、各種ベンチマークを使用してシステムを評価する事である。評価の方法としては I/O ベンチマークと実際の科学計算アプリケーションの性能測定の他に、ファイルシステムのトレースを利用してキャッシュヒット率を測る等の方法も考えられる。

また、現在は pdCache からバックエンドの並列ファイルシステムに対して操作が行われる形になっているが、pdCache を階層的に配置する事でより効率的なキャッシュシステムを構築する可能性も模索したい。つまり、pdCache のバックエンドとして pdCache を使用できるような形で設計を行う。各階層ではメモリの他に、計算ノードのローカルディスクを使用したり、SSD のようなフラッシュドライブを使用する事が考えられる。その際は上層のキャッシュと下層のキャッシュ間での一貫性制御も必要となる事が予想される。

参考文献

1) Infiniband. http://www.infinibandta.org/.
2) Lustre file system. http://lustre.org/.
3) Mvapich: Mpi over infiniband and iwarp. http://mvapich.cse.ohio-state.edu/.
4) Myrinet. http://www.myri.com/.
5) Philip Carns, Robert Ross, Walter Ligon III, and Pete Wyckoff. Bmi: A network abstraction layer for parallel i/o. In *IPDPS '05: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Workshop 9*, page 213.1, Wash-

ington, DC, USA, 2005. IEEE Computer Society.

6) Philip H. Carns, Walter B. Ligon III, Robert B. Ross, and Rajeev Thakur. Pvfs: a parallel file system for linux clusters. In *ALS'00: Proceedings of the 4th conference on 4th Annual Linux Showcase & Conference, Atlanta*, pages 28–28, Berkeley, CA, USA, 2000. USENIX Association.

7) Michael D. Dahlin, Clifford J. Mather, Randolph Y. Wang, Thomas E. Anderson, and David A. Patterson. A quantitative analysis of cache policies for scalable network file systems. In *SIGMETRICS '94: Proceedings of the 1994 ACM SIGMETRICS conference on Measurement and modeling of computer systems*, pages 150–160, New York, NY, USA, 1994. ACM.

8) Michael D. Dahlin, Randolph Y. Wang, Thomas E. Anderson, and David A. Patterson. Cooperative caching: using remote client memory to improve file system performance. In *OSDI '94: Proceedings of the 1st USENIX conference on Operating Systems Design and Implementation*, page 19, Berkeley, CA, USA, 1994. USENIX Association.

9) Juan Miguel del Rosario, Rajesh Bordawekar, and Alok Choudhary. Improved parallel i/o via a two-phase run-time access strategy. *SIGARCH Comput. Archit. News*, 21(5):31–38, 1993.

10) Kamil Iskra, John W. Romein, Kazutomo Yoshii, and Pete Beckman. Zoid: I/o-forwarding infrastructure for petascale architectures. In *PPoPP '08: Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming*, pages 153–162, New York, NY, USA, 2008. ACM.

11) Argonne National Laboratory. Mpich2 : High-performance and widely portable mpi. http://www.mcs.anl.gov/research/projects/mpich2/.

12) Arifa Nisar, Wei keng Liao, and Alok Choudhary. Scaling parallel i/o performance through i/o delegate and caching system. In *SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, pages 1–12, Piscataway, NJ, USA, 2008. IEEE Press.

13) Amar Phanishayee, Elie Krevat, Vijay Vasudevan, David G. Andersen, Gregory R. Ganger, Garth A. Gibson, and Srinivasan Seshan. Measurement and analysis of tcp throughput collapse in cluster-based storage systems. In *FAST'08: Proceedings of the 6th USENIX Conference on File and Storage Technologies*, pages 1–14, Berkeley, CA, USA, 2008. USENIX Association.

14) Frank Schmuck and Roger Haskin. GPFS: A shared-disk file system for large computing clusters. In *Proc. of the First Conference on File and Storage Technologies (FAST)*, pages 231–244, January 2002.

15) Osamu Tatebe, Noriyuki Soda, Youhei Morita, Satoshi Matsuoka, and Satoshi Sekiguchi. Gfarm v2: A grid file system that supports high-performance distributed and parallel data computing. In *Proceedings of the 2004 Computing in High Energy and Nuclear Physics (CHEP04)*, 2004.

16) R. Thakur, W. Gropp, and E. Lusk. An abstract-device interface for implementing portable parallel-i/o interfaces. In *FRONTIERS '96: Proceedings of the 6th Symposium on the Frontiers of Massively Parallel Computation*, page 180, Washington, DC, USA, 1996. IEEE Computer Society.

17) Rajeev Thakur, William Gropp, and Ewing Lusk. On implementing MPI-IO portably and with high performance. In *Proceedings of the Sixth Workshop on Input/Output in Parallel and Distributed Systems*, pages 23–32, 1999.

18) Rajeev Thakur, William Gropp, and Ewing Lusk. Optimizing noncontiguous accesses in MPI-IO. *Parallel Computing*, 28(1):83–105, 2002.

19) Qingsong Wei, Bharadwaj Veeravalli, and Lingfang Zeng. Dwc¡sup¿2¡/sup¿: A dynamic weight-based cooperative caching scheme for object-based storage cluster. In *Cluster Computing, 2008 IEEE International Conference on*, pages 167–174, 2008.