Vol.2013-ARC-207 No.9
Vol.2013-HPC-142 No.9
2013/12/16

エクサスケールシステムに向けたファイルシステムデザインの検討

上村佳史[†1]　酒井憲一郎[†1]　樋口雄太[†1]　小田和友仁[†1]
住元真司[†1]　宇野俊司[†1]　清水正明[†2]　石川裕[†3]

将来の HPC システムでは計算量の増加に伴い入出力のデータ量も増大するため，ファイルシステムには大容量かつ高い I/O 性能が必要になると予想される．これらは単純にサーバとストレージ装置の台数を増やすことでも実現できるが，ポストペタスケールシステムではコストや消費電力面で難しい．我々はこれらの問題を解決するために，3 階層の階層ストレージを提案してきた．しかし，この階層ストレージを有効活用するためには高速かつ透過的なファイル I/O を実現する必要がある．本論文では，Delegation 機能を持つマイクロカーネルを使用したファイルシステムの実現方式を提案する．本方式では，マイクロカーネル上にローカルストレージへのキャッシュ機能を実装すると共に，キャッシュとユーザメモリ間のファイルデータ転送をユーザレベルで行う I/O 機構を実装することで，高速な大容量ファイルへのアクセスを実現できる．さらに，本方式は共有ファイルシステムを修正しないため，既存のファイルシステムを利用でき，テープアーカイブへのアクセスも容易に実現できる．

Design Consideration of file-system for exascale systems

YOSHIFUMI UEMURA[†1]　KENICHIRO SAKAI[†1]　YUTA HIGUCHI[†1]
TOMOHITO OTAWA[†1]　SHINJI SUMIMOTO[†1]　SHUNJI UNO[†1]
MASAAKI SHIMIZU[†2]　YUTAKA ISHIKAWA[†3]

Abstract: It is anticipated that a large capacity and high I/O performance file system is needed for a future HPC system because the volume of data of I/O increases as the computational complexity increases. Though a high I/O performance mass storage system can be achieved by simply providing the huge amount of the storage devices, it is difficult to deploy such a system due to high cost and power consumption. We have proposed a hierarchical storage system, forming three layers, to solve these problems. To utilize this storage system, a transparent high speed file I/O mechanism must be achieved. In this paper, a new method of file I/O functions, based on the delegation mechanism of a micro kernel, is proposed. In order to carry out a high speed data access to huge files, storage cache is located on the local storage of the micro kernel and file data between cache and user memory is transferred in the user level. Because the proposed method is independent from the file system internals, it is easy to access tape archive by utilizing existing shared file system.

1. はじめに

現在我々は，エクサフロップス級スーパーコンピュータを目指し，2018 年の実現を目標にしているポストペタスケールシステムに必要なハードウェア及びソフトウェアを検討している．ソフトウェアとしては，想定ハードウェア構成上で動作させる OS カーネル，通信ライブラリ，ファイルシステムからなるシステムソフトウェアスタックについて検討・設計を進めている[1]．本稿は，ストレージ構成とファイルシステムについて主眼を置いている．

本稿では，ポストペタスケールシステムにおける階層ストレージを有効活用するためには高速かつ透過的なファイル I/O を実現する方法について検討した結果を報告する．

2. 想定システム構成と要件

本章では，ポストペタスケールシステムで想定されるストレージと計算ノードの構成，およびファイルシステム要件について述べる．

2.1 階層ストレージの構成

スーパーコンピュータ用のファイルシステムには，高い I/O 性能と大容量，省電力，並列プログラムへの外乱が少ないことが求められる．一つのファイルシステムによりこれら全ての要件を満たすことは困難であることから，ストレージ構成を階層化する必要がある．現在国内で最大規模の「京」では，これを 2 階層のストレージ構成により実現してきた．

我々がターゲットとするポストペタスケールシステムでは，計算量の増加に伴い「京」の 20 倍を超える 1EB のストレージが必要と予想されており，これを低コストかつ省電力に実現するため，図 1 に示す 3 階層の階層ストレージを提案してきた[2]．1 階層目はフラッシュメモリなどの半導体記憶装置を用いた高速なローカルストレージ，2 階層

†1 富士通(株)
Fujitsu Ltd.
†2 (株)日立製作所
Hitachi Ltd.
†3 東京大学
University of Tokyo

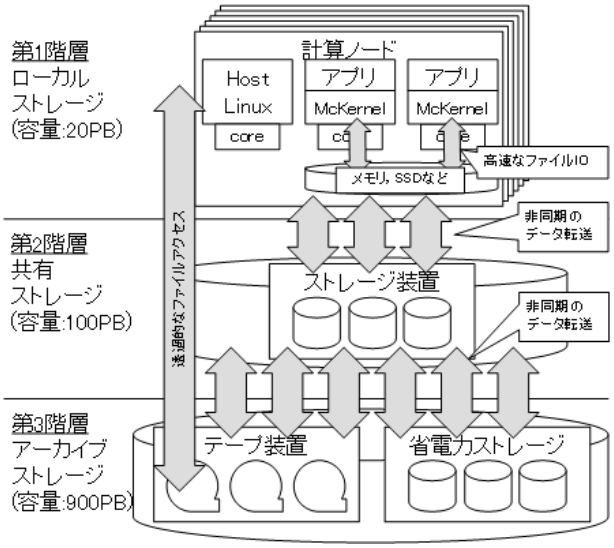

図 1　階層ストレージのシステム構成

目はノード間でファイル共有するディスク装置を使った共有ストレージ，3 階層目はテープ装置を使用する省電力なアーカイバである．

2.2 計算ノードの構成

各計算ノードにはメニーコアプロセッサを搭載し，アプリケーション用の計算コアと Linux カーネルを動作させる OS コアとして使用し，異種カーネルを混在させる方法を想定する．計算コアにはアプリケーションの実行に必要なプロセス生成やメモリ管理など最小限の OS 機能を提供するマイクロカーネル(McKernel)を用い，ファイルシステムのカーネルスレッドなどのシステムデーモンによる OS ノイズを排除する．マイクロカーネルの Linux システムコールは OS コアの Linux カーネルに委譲(Delegation)することで実現する[3].

2.3 階層ストレージの動作とファイルシステム要件

計算ノード上のアプリケーションは，各ノードが占有する 1 階層目のローカルストレージに高速にファイル I/O する．アプリケーションの入力データは 2 階層目の共有ストレージからローカルストレージに転送し，ローカルストレージにアプリケーションが出力したデータは共有ストレージに転送する．共有ストレージ上のファイルはサイズや最終更新日付などの条件に基づき 3 階層目のアーカイバに転送し，次にアクセスする際にアーカイバから共有ストレージに転送する．

この階層ストレージをアプリケーションから最大限有効に活用するためには，アプリケーションのファイル I/O 時間を最小化し，計算ノードを有効利用するための高速な I/O と，階層の違いを意識せずにファイルにアクセスできる透過的な I/O が重要である．この高速かつ透過的なファイル I/O を実現するためには以下の実現が重要であり，現在我々はこれらを実現するための方法を検討している．

① アプリケーションからの高速な I/O
アプリケーションのファイル I/O にかかる時間を最小限に抑え，アプリケーションがシステムの演算性能を最大限に引き出せるようにする．

② 階層間のデータ転送の隠ぺい
計算ノードの有効利用のため，ローカルストレージと共有ストレージ間のデータ転送時間を隠ぺいする．ただし，データ転送の高速化と OS ノイズによるアプリケーション実行の外乱を排除する必要がある．

③ 透過的なファイルアクセス
3 階層間の透過的なファイルアクセスを実現する必要がある．

次章以降では，上記①②③の実現方式について検討する．

3. アプリケーションからの高速 I/O

アプリケーションから 1 階層目のローカルストレージに対する I/O を高速化するためには，低遅延かつ高バンド幅のデータアクセスを実現する必要がある．本章では，この 2 つの実現方法について検討する．

3.1 低遅延なデータアクセス

Linux のファイル I/O（図 2 左）はシステムコールにより Linux カーネルに処理が移行し，VFS 層やファイルシステム層を通してストレージに I/O が行われる. Linux では複数のアプリケーションが資源を共有して動作するため，アプリケーションやノード間の排他が必要である．

しかし，1 つのカーネル上では 1 つのプロセスのみが動作する事を想定している McKernel では，必ずしも Linux で行っている処理すべてを実行する必要はない. たとえば，プロセス間の排他処理は必要ない．図 2 の右側に示すように，VFS，ファイルシステム層，システムコールを省くことで，ユーザレベル I/O としてローカルストレージにアクセスできる．

このように，アプリケーションからローカルストレージへのアクセスを単純化することで，低遅延なデータアクセスが実現できる．

3.2 高バンド幅なデータアクセス

文献[4]ではメニーコア化が進むにつれて，高バンド幅な I/O を実現するためには I/O 処理の並列化が必要であると述べた. 本階層ストレージにおいてもこのような並列 I/O を実現することで，高バンド幅なデータアクセスが可能となる．並列 I/O の詳細については文献[4]を参照されたい．

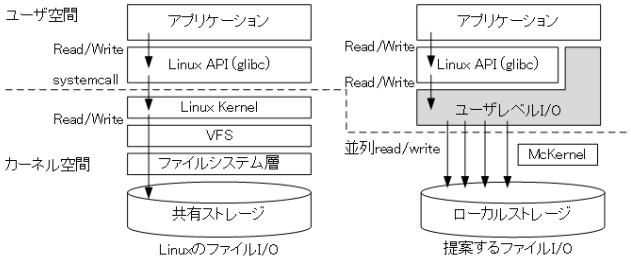

図 2　Linux と提案するファイル I/O

4. 階層間のデータ転送の隠ぺい

4.1 ステージング方式と課題

既存のスーパーコンピュータで採用されているステージング方式について，「京」を例に説明する．

ステージング方式は，アプリケーションの一時領域として使用するローカルストレージとユーザデータを保存するためのグローバルストレージで構成し，アプリケーションの実行前後に両ストレージ間で入出力ファイルを転送する方式である．グローバルストレージより高速かつ少容量のローカルストレージをアプリケーションに占有利用させ，ユーザによるファイルアクセスをローカルストレージから分離することで，アプリケーションとユーザ間のファイル I/O（ファイルデータおよびメタ情報）の競合を防ぎ，高速なファイル I/O を実現すると共にアプリケーションへの外乱を排除できる．

「京」は，図 3 に示すように約 11PB のローカルストレージと約 30PB のグローバルストレージにより構成される．アプリケーションが使用するファイルは実行前にグローバルストレージからローカルストレージにコピー（ステージイン）され，アプリケーションが生成または変更したファイルはアプリケーションの実行後にローカルストレージからグローバルストレージにコピー（ステージアウト）される．「京」のファイルシステムは当時世界最大の規模であったが，2 階層のストレージとステージング方式によりシステムを安定稼働させ運用に供することができている．

しかし，ステージング方式をポストペタスケールシステムで採用する場合には以下の課題がある．

・ローカルストレージ容量

ジョブの全入出力ファイルは，アプリケーション実行中にローカルストレージ上に保持する必要があるため，アプリケーションの入出力データ量に比例して必要なストレージ容量が増大する．

・ステージング時間

アプリケーションの入出力データ量の増大に比例して，ローカルストレージとグローバルストレージにファイル転送時間が増大し，アプリケーションが終了してからユーザが実行結果を確認できるまでの時間がかかる．

引用文献[1]には，ポストペタスケールシステムのターゲットアプリケーション一つである COCO（海洋シミュレーション）を例に，ステージング方式による上記の課題について述べられている．COCO は 10 年分のシミュレーション結果として合計 365PB のデータを出力するが，ステージング方式ではローカルストレージに 365PB もの容量が必要

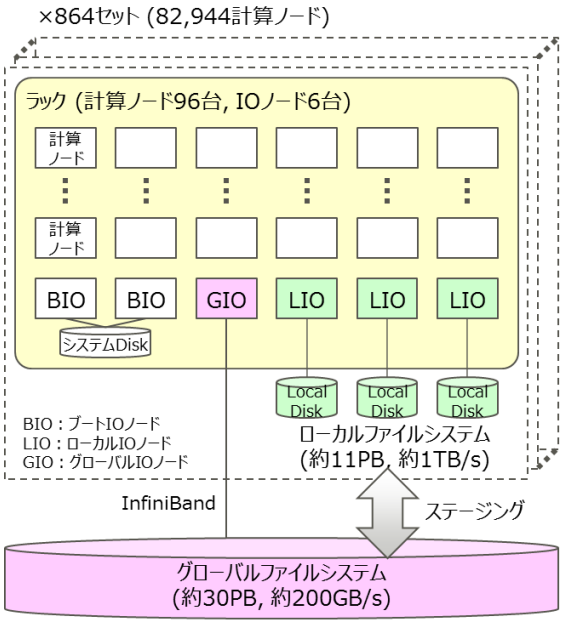

図 3　「京」の構成

となり，我々が想定する階層ストレージの容量を超過してしまう．また，1TB/s でステージアウトしても約 4 日を要し，シミュレーション結果を確認するまでに時間がかかる．

4.2 キャッシュ方式と課題

前節で述べたステージング方式の課題を解決する方法として，ローカルストレージをキャッシュ的に用い，ローカルストレージと共有ストレージ間のデータ転送をアプリケーションのバックグランドで非同期に実行する方式を考える．以降，本方式をキャッシュ方式と呼ぶ．

キャッシュ方式では，アプリケーションが実行中に必要とするファイルの一部をローカルストレージ上に一時的に保持することで，ローカルストレージに必要な容量を削減できる．また，ローカルストレージと共有ストレージ間のデータ転送をアプリケーションのバックグランドで非同期に実行することで，データ転送に掛かる時間をアプリケーションから隠ぺいできる．

図 4 にステージング方式に対するキャッシュ方式の効果を示す．アプリケーション開始時に必要なファイル以外を，アプリケーション内で読み込む前に共有ストレージからローカルストレージに非同期に転送する（プリフェッチ）．アプリケーションがローカルストレージに書き出したデータ

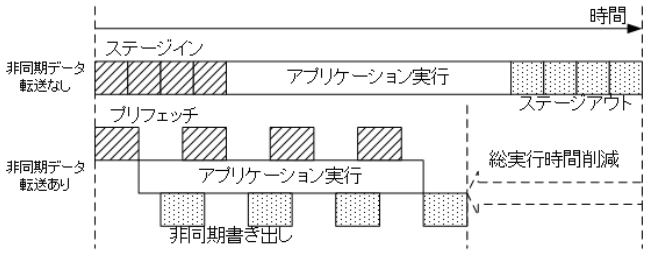

図 4　非同期データ転送

による総実行時間削減は，アプリケーションの完了を待たずに非同期に共有ストレージに転送する（非同期書出し）．アプリケーションで計算フェーズと I/O フェーズを分けることで，ローカルストレージと共有ストレージ間で非同期データ転送が可能になり，ステージング方式に比べてデータ転送を含む総実行時間を短縮することができる．

前述の引用文献では，COCO で 1 日分のシミュレーションを行うごとにローカルストレージに出力データを書き出し，次の 1 日分のシミュレーションを行うバックグランドでローカルストレージからグローバルストレージに出力ファイルを転送することが提案されている．1 日分のシミュレーション出力は 100TB であるため，キャッシュ方式を適用することで，ローカルストレージの使用量をアプリケーションの出力用の 100TB と，グローバルストレージへの転送が終わるまで保持しておくための 100TB の合計 200TB に抑えることができる．

このように，キャッシュ方式はステージング方式の課題であるローカルストレージ容量とステージング時間の問題を解決できるが，ポストペタスケールシステムにキャッシュ方式を適用する場合には，以下の二つの課題を解決する必要がある．

・アプリケーションへの外乱

ステージング方式ではジョブの前後にステージングを行うためアプリケーションへの外乱にならないが，キャッシュ方式ではアプリケーションの動作中にデータを転送する必要があるため，アプリケーションの外乱にならないデータ転送方法が必要である．

・メタアクセス性能

ステージング方式では，ローカルストレージと共有ストレージでファイルシステムを分けるため，両者へのメタアクセスは分離・分散される．しかし，キャッシュ方式ではローカルストレージにはメタ情報を持たず，全てのメタアクセスは共有ストレージに対して行われる．このため，共有ストレージへのメタアクセスの負荷を低減させる施策が必要である．

次節では，これらの課題を解決するための方法について検討した結果を述べる．

4.3 アプリケーションへの外乱に対する施策

アプリケーションの外乱要因には，非同期書き出しのデーモン処理による CPU 資源の競合が考えられる．CPU 競合の抑止は，ユーザアプリケーションが占有する資源以外のコアとメモリを使用して，アプリケーションのデータアクセスとは非同期にデータ転送することで実現する．本方式での write 時のデータの流れを以下に示す（図 5）．

1. アプリケーションからユーザレベル I/O でアプリケーションのデータをローカルストレージに転送する．
2. Host Linux で動作しているデーモンが，ローカルストレージのデータを共有ストレージに書き出す．

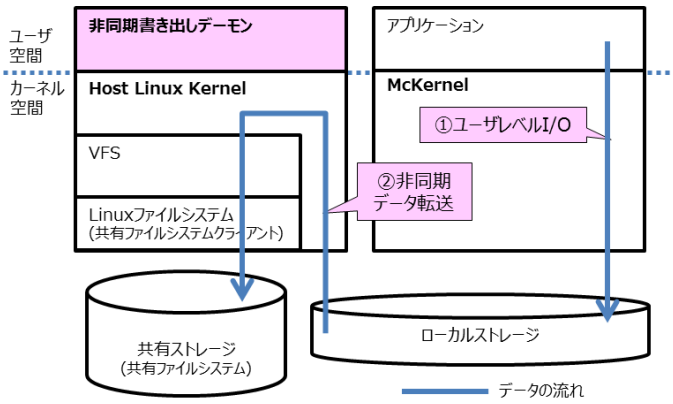

図 5　ユーザレベル I/O と非同期データ転送の流れ

共有ストレージに書き出しのタイミングは以下の 3 つを想定する．

- 明示的に非同期書き出し指示があった場合
- ファイルが close された場合
- 定期的な Dirty データの書き出し

4.4 メタアクセス性能に対する解決のアプローチ

共有ストレージへのメタアクセスの負荷の削減は，ファイルのメタ情報を計算ノードにキャッシュすることで解決する．一般に共有ファイルシステム上のキャッシュを実現するためにはノード間での一貫性の保証が課題となる．

本課題は，McKernel の Delegation 機能と VFS 層のキャッシュの仕組みを活用することにより，共有ファイルシステムの排他制御の仕組みを利用して解決する．

本方式によるメタ情報アクセス時の流れを以下に示す（図 6）．

1. アプリケーションからのメタアクセス要求を McKernel が Host Linux の mcexec に回送し，mcexec が Linux に対してシステムコールを発行する．
2. VFS 層は，メタ情報キャッシュの有無を確認する．無い場合は共有ファイルシステムを通して共有ストレージからメタ情報を取得し，キャッシュに載せる．キャッシュに有る場合は，キャッシュのメタ情報を使用する．
3. システムコールの結果をアプリケーションに返す．

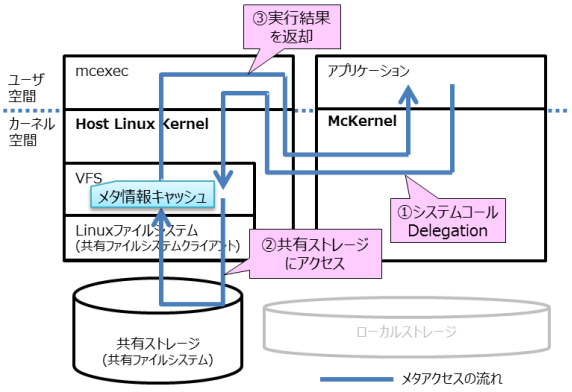

図 6　メタデータの流れ

5. 透過的なファイルアクセス

透過的なファイルアクセスとは，階層ストレージ全体で 1 つのファイルシステムの名前空間を持ち，ユーザやアプリケーションがデータの存在する階層を意識することなくファイルにアクセスできることと定義する．本章では各階層間の透過的ファイルアクセスについて検討する．

5.1 1,2 階層間

4 章で述べたように，1 階層目のローカルストレージは 2 階層目の共有ストレージのキャッシュとして扱い，名前空間を持たない．アプリケーションからのアクセス要求は共有ストレージに発行され，共有ストレージの名前空間が使用される．このように 1,2 階層を通しての透過的アクセスが実現できる．

5.2 2,3 階層間

ファイルシステムとテープアーカイバ間での透過的ファイルアクセスを実現する方法として HSM(Hierarchical Storage Management)が広く使われている．また共有ストレージは安定性が重要であることから，ペタフロップス級の大規模なスーパーコンピュータで実績があり，HSM に対応した Lustre などの既存の分散ファイルシステムを使用することで，共有ストレージとアーカイバ間の透過的なファイルアクセスを容易に実現できる[2].

6. 関連研究

Fu 等は check pointing についてアプリケーションレベルでの two-phase I/O 及び MPI-IO の collective I/O による手法について提案，評価を実施している[5,6]．この文献ではアプリケーションレベルでの two-phase I/O により高い性能及びスケーラビリティが得られたと述べられている．提案している two-phase I/O は計算と I/O 処理を分離し，I/O を非同期に行う手法であり，本研究も計算処理と I/O を非同期に行うが，システムレベルで行うためアプリケーションによらず実現できる点が異なる．

IOFSL(I/O Forwarding Scalability Layer)[7,8]はアプリケーションから共有ファイルシステムへのファイル I/O 要求を I/O ノードにオフロードし，I/O ノードでファイル I/O 要求を収集，バッファし，入出力タイミングを分散することで，ファイルシステムへのアクセスを削減することを目的としている．本研究とは I/O 処理を移譲するという点が似ている．しかし本研究では共有ファイルシステムへのデータ転送はバックグランドで行い，アプリケーションに対してデータ転送コストを隠ぺいするという点で異なる．

また，安井等は同一ファイルに複数ノードからのアクセスが集中することに起因するサーバ負荷を低減する手法として，ユーザレベルでの並列ファイルキャッシュシステムを提案[10]している．この手法では，計算ノードが共有ファイルシステムにアクセスする際に，ファイル名から求めたハッシュ値をもとに決定したシステム上のいずれかの計算ノードを経由するよう設計している．また各計算ノードのメモリとローカルディスクを中継機能のキャッシュとして活用し，同一ファイルに対する中継要求はキャッシュで折り返すことで，サーバにかかる負荷を低減している．本研究とは計算ノード間でキャッシュを共有している点では異なるが，同一ファイルに対する大量アクセスの効率化の観点で，本論文で提案しているファイルシステムアーキテクチャと組み合わせることで本技術を活かすことができる．

7. おわりに

本論文では 3 階層の階層ストレージを有効活用するために，Delegation 機能を持つマイクロカーネル上にアプリケーションからの高速な I/O，階層間のデータ転送の隠ぺい，そして透過的なファイルアクセスについて実現方法および検討結果について述べた．高速なファイル I/O はユーザレベル I/O 及び並列 I/O とすることで実現すると述べた．階層間のデータ転送の隠ぺいはローカルストレージをキャッシュとして扱い，データ転送をアプリケーションのバックグラウンドで非同期に行う方法について述べ，その時の課題の解決方法についても述べた．このキャッシュ方式は海洋や気象のように時系列の変化をシミュレートし，計算フェーズと I/O フェーズを繰り返すようなアプリケーションで有効である．透過的なファイルアクセスについてはキャッシュ方式と HSM 機能を有する既存の共有ファイルシステムを使用することで 3 階層ストレージの全階層にわたって透過的なファイルアクセスが実現できると述べた．

今後は，本稿で提案した方式のプロトタイプ実装及び評価を実施する予定である．

謝辞 本研究の一部は，文部科学省「将来の HPCI システムのあり方の調査研究」の研究課題「レイテンシコアの高度化・高効率化による将来の HPCI システムに関する調査研究」によるものである．研究にあたり検討に参加いただいた参加メンバ各位にこの場を借りて謝辞を述べる．

参考文献

1) 石川裕，堀敦史，Gerofi Balazs，高木将通，島田明男，清水正明，佐伯裕治，白沢智輝，中村豪，住元真司，小田和友仁: 次世代高性能並列計算機のためのシステムソフトウェアスタック，情報処理学会第 124 回システムソフトウェアとオペレーティング・システム研究会(2013)
2) レイテンシコアの高度化・高効率化による将来の HPCI システムに関する調査研究，SDHPC,
http://www.open-supercomputer.org/wp-content/uploads/2013/06/sdhpc10-ishikawa.pdf
3) 佐伯裕治,清水正明,白沢智輝,中村 豪,高木将通,Gerofi, B.,思 敏,石川 裕,堀 敦史:ヘテロジニアス計算機上の OS 機能委譲機構,情報処理学会 第 124 回システムソフトウェアとオペレーティング・システム研究会 (2013).
4) 小田和友仁, 住元真司, 堀敦史, 石川裕: メニーコア向け NUMA 最適並列分散 I/O の予備検証，情報処理学会第 124 回システムソ

フトウェアとオペレーティング・システム研究会(2013)
5) J. Fu, N. Liu, O. Sahni, K. Jansen, M. Shephard, and C. Carothers: Scalable parallel I/O alternatives for massively parallel partitioned solver systems. In 2010 IEEE International Symposium on Parallel and Distributed Processing
6) J. Fu, M. S. Min, R. Latham, C. D. Carothers. Parallel I/O Performance for Application-Level Checkpointing on the Blue Gene/P System. In Workshop on Interfaces and Architectures for Scientific Data Storage (IASDS), in conjunction with IEEE International Conference on Cluster Computing,(Cluster 2011)
7) Nawab Ali, Philip Carns, Kamil Iskra, Dries Kimpe, Samuel Lang, Robert Latham, Robert Ross, Lee Ward, and P. Sadayappan: Scalable I/O Forwarding Framework for High-Performance Computing Systems, IEEE International Conference on Cluster Computing (Cluster 2009)
8) Kazuki Ohta, Dries Kimpe, Jason Cope, Kamil Iskra, Robert Ross, and Yutaka Ishikawa: Optimization Techniques at the I/O Forwarding Layer, IEEE International Conference on Cluster Computing (Cluster 2010)
9) Lustre file system: http://lustre.opensfs.org/
10) 安井隆，清水正明，堀淳史，石川裕:ローカルディスクを活用するユーザレベル並列ファイルキャッシュシステムの設計，第 10 回先進的計算基盤システムシンポジウム SACSIS(2012)