Vol.2009-HPC-121 No.40
2009/8/6

ファイルキャッシュシステムの有効性向上に向けた科学技術計算アプリケーションのI/O特性評価

安井　隆[†]　清水正明[†,†††]　住元真司[††]　太田一樹[†††]
鴨志田良和[†††]　松葉浩也[†††]　堀　敦史[†††]　石川　裕[†††]

スーパーコンピュータが搭載するプロセッサコア数の増加により，ファイルシステムに対するI/O要求の多重度が増大しアプリケーションのI/O性能の確保が困難になっている。この解決のため，計算機ノードでファイルデータをキャッシュしファイルシステムの負荷を低減する手法などを提案している。これらの手法の有効性を確認するには，アプリケーションのI/O特性の把握が必要である。そこでアプリケーションのI/Oを観測する技術を開発した。この技術によりファイルI/Oの発生状況などを確認することが可能となる。実際に科学技術計算アプリケーションに適用した結果，提案している手法が有効であるケースを確認できた。

I/O Characteristics Evaluation of Scientific Applications for Effectiveness Improvement of File Cache System

Takashi Yasui,[†] Masaaki Shimizu,[†,†††] Shinji Sumimoto,[††]
Kazuki Ohta,[†††] Yoshikazu Kamoshida,[†††]
Hiroya Matsuba,[†††] Atsushi Hori[†††] and Yutaka Ishikawa[†††]

In the supercomputer system, the increase of processor cores raises the complexity of I/O requests, and it is difficult to achieve high I/O performance for each application. To solve this problem, we have proposed a method that reduces the load of the file system by caching file data on the computer nodes, and so on. We need to know I/O characteristics of applications in order to observe the effectiveness of the proposed methods. Then we developed the tools that survey I/O of an application. By the tools, we can observe the I/O patterns. We actually used the tools for the scientific applications. As a result, we observed the effective case of the proposed methods.

1. はじめに

近年スーパーコンピュータで取り扱う科学技術計算の分野が拡大している。スーパーコンピュータは，大規模なデータを処理するアプリケーションを実行(capability computing)する目的だけではなく，パラメータスイープのように計算条件を変更してアプリケーションを多数実行(capacity computing)する目的でも利用されている。このようなさまざまな利用目的に対応するために大規模なクラスタシステムが普及している。例えば，東京大学情報基盤センターで運用されているT2K東大システムは，952台の計算ノード，15,000個以上のプロセッサコアから構成される大規模クラスタシステムである。このような大量なプロセッサコアから構成されるクラスタシステムにおいては，演算性能とネットワークの性能向上だけでなく，ファイルI/Oの性能の確保が重要となる。例えば，1万個のプロセッサを使用するアプリケーションが各プロセッサ上でプロセスを実行しファイルI/Oを行なうとすると，1万プロセスからのI/O要求が発生することになり，ファイルシステムに対して過剰な負荷がかかる。

ファイルI/O性能を確保する技術として，ファイルステージングや並列ファイルシステムがある。ファイルステージングは，理化学研究所のRIKEN Super Combined Cluster System[1]や海洋研究開発機構のEarth Simulator[2]が採用している。ファイルステージングは，共有ファイルシステムと計算機ノードが高速にアクセスできるファイルシステム(ここではローカルファイルシステムと呼ぶ)の2つのファイルシステムを想定する。アプリケーションの実行に先だって，共有ファイルシステム上にあるファイルをローカルファイルシステムにコピーし，アプリケーションの実行後は，アプリケーションがローカルファイルシステム上の修正あるいは生成したファイルを共有ファイルシステム上にコピーする。クラスタシステムにおいて計算機ノードにローカルディスクが搭載されている場合，ローカルファイルシステムはローカルディスクを使用して構築することが多い。この場合，ローカルディスクの容量や，各計算機ノードはローカルファイルのみしか利用できないという制限がある。

並列ファイルシステムとしては，PVFS[3]やLustre File System[4]，GPFS[5]，Shared Rapid File System[6]，Hitachi Striping File System[7]などがある。並列ファイルシステムは，複数のファイルサーバノードから構成され，各計算機ノードから発生するファイルI/Oを負荷分散させる。従来の並列ファイルシステムは，大容量のファイルI/Oに対する性能を考慮した設計となっていることが多い。また，ファイルへアクセスする

[†] 株式会社 日立製作所
Hitachi, Ltd.
[††] 株式会社 富士通研究所
Fujitsu Laboratories, Ltd.
[†††] 東京大学
The University of Tokyo

プロセス数に対するスケーラビリティは考慮されていないことが多い。しかし，先に述べたようにプロセッサコア数の増大と共に,従来よりも小さいサイズのファイル I/O の多重度が増大している。このため，並列ファイルシステムによるファイル I/O の性能確保が困難なケースも起こっている。

我々は並列ファイルシステムに対する負荷を軽減する手法として，計算機ノードからの I/O 要求をバッファリングし一括処理する Gather-Arrange-Scatter[8]や，計算機ノードと並列ファイルシステムとの間にキャッシュ機構を設ける pdCache[9]を提案している。これらの手法の有効性を確認するとともに，さらに有効性を向上していくためには，アプリケーションのファイル I/O の特性を分析していく必要があり，さまざまなアプリケーションにおけるファイル I/O の発生状況や I/O 量の統計情報の採取が重要となる。

本稿では，ファイル I/O のアクセス情報を採取し，アプリケーション実行時に発生するファイル I/O の分析を支援する I/O 特性評価支援技術 I/O Counter(IOC)を提案する。IOC は，動的にライブラリをリンクするバイナリコードに対して，ユーザレベルでファイル I/O のアクセス情報の採取を可能にする。このため IOC は，アプリケーションのコードを修正あるいは再コンパイルすることなく利用することができる。

以下 2 章で，IOC の実現に向けた課題について述べた後，開発した IOC について述べる。3 章では，一般に公開されている科学技術計算アプリケーションを使用し，IOC によるアプリケーションのファイル I/O の観測結果を示す。4 章ではファイル I/O の観測技術についての関連研究を紹介し，最後に 5 章で本稿をまとめる。

2. I/O 特性評価支援技術

2.1 課題

アプリケーション実行時に発生するファイル I/O を分析するためには, I/O 要求の発生タイミングや対象ファイル，I/O 量を把握する必要がある。また I/O 要求としては，データの読み込みや書き出しの他，ファイルのアクセス状況を分析できるようにファイルのアクセス位置を変更する要求を観測する必要がある。ファイル I/O の分析に向けて観測するデータと観測の対象とする I/O 要求の種類を表 1 に示す。

表 1　観測データ

観測データ	内容
type	I/O 要求の種類 (read, write, open, close, creat, lseek, fsync, ftruncate)
time	I/O 要求の発生時刻
fd	I/O 要求の対象ファイル識別子
data	I/O 要求の情報 (I/O 量，オフセット，etc.)

本稿では，上記のような観測データを採取し，アプリケーション実行時に発生するファイル I/O の分析を支援する I/O 特性評価支援技術 I/O Counter(IOC)を開発する。この IOC がアプリケーションの I/O 要求を観測する位置を図 1 に示す。

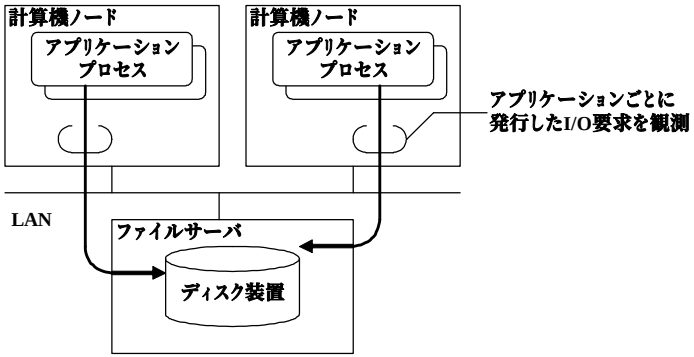


図 1　I/O 要求の観測位置

アプリケーション実行時に発生するファイル I/O を観測する位置として，アプリケーションの I/O 要求が集まってくるファイルサーバ上で観測することが考えられる。しかし，計算機センターなどで運用しているスーパーコンピュータにおいて一般ユーザがファイルサーバにアクセスし，ファイルサーバに到達する I/O 要求を観測することは困難である。そこで IOC では，計算機ノード上でアプリケーションごとに発行した I/O 要求をユーザレベルで観測できるようにする。

このとき，IOC の実現に向けては次の 4 点が課題となる。

(1) アプリケーションへの観測コードの追加:
さまざまなアプリケーションのコード中の I/O 要求を発行する部分へ観測データを採取するコードを追加しなければならない。

(2) 観測データの保存:
アプリケーションのファイル I/O の分析に向けて，各 I/O 要求について採取した観測データを保存しておかなければならない。

(3) 観測データの収集:
アプリケーションによっては複数のプロセスで並列に処理が行なわれる。このようなアプリケーションのファイル I/O の分析に向けては，各プロセスについて観測データを採取し，収集する必要がある。

(4) 観測データの整形:

アプリケーションのファイル I/O を分析しやすくするため，I/O 要求の発生タイミングや I/O 量を把握できるよう，観測データを整形する必要がある。

2.2 課題の解決に向けたアプローチ

2.1 節で述べた IOC の実現に向けた課題に対し，次の 4 点のアプローチを採ることで解決を図る。

(a) I/O 要求コードへの観測コードの動的挿入:
観測データを採取するコードの追加をアプリケーションごとに人手で行なうには多大な工数が必要となる。また，アプリケーションによってはバイナリコードのみが提供され再コンパイルが不可能なものもある。そこで IOC では，アプリケーション実行時に I/O 要求コードへ動的に観測コードを追加できるようにする。

(b) 観測データのメモリ上での保持:
アプリケーション実行時に発生する I/O 要求ごとに観測データを出力することは，アプリケーションに対する負荷を大きくすることになる。そこで IOC では，アプリケーション実行中に採取する観測データはメモリ上に保持しておき，アプリケーション終了時に観測データをファイルへ書き出すようにする。

(c) 同一ファイルへの観測データの書き出し:
アプリケーションの複数のプロセスが観測データを書き出したファイルを人手で収集するには利便性に問題がある。そこで IOC では，アプリケーションの各プロセスが同じファイルに対して観測データを書き出すことも可能にする。

(d) アプリケーション終了後の観測データの整形:
アプリケーション実行時に観測データを整形することは，アプリケーションに対する負荷を大きくすることになる。そこで IOC では，アプリケーション実行時には観測データの採取のみを行ない，アプリケーション終了後に別途観測データを整形するようにする。

2.3 IOC の実現

2.2 節で述べたアプローチにしたがって，アプリケーションの観測データを採取するライブラリと観測データを整形するツールとして IOC を実現した。

IOC ライブラリは次の 3 点の機能を備えている。

(1) プリロードを利用した I/O 要求コードへの観測コード挿入機能:
アプリケーション実行時に動的リンクライブラリをプリロードする機能を利用して観測の対象とする各 I/O 要求をフックし，観測データを採取する。IOC ライブラリによる I/O 要求のフックの概要を図 2 に示す。

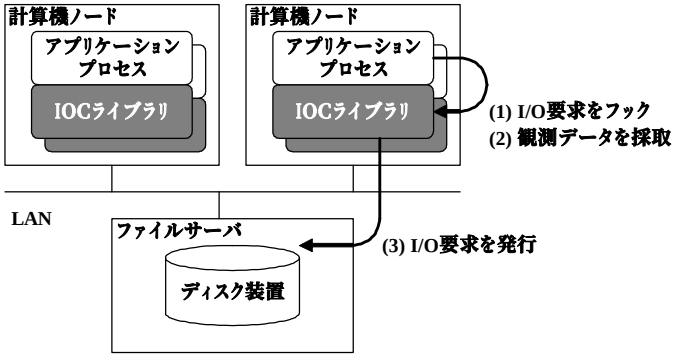


図 2 I/O 要求のフック

プリロードを利用してアプリケーション実行時に IOC ライブラリをリンクする。その後，アプリケーションが I/O 要求を発行すると IOC ライブラリでフックされる。IOC ライブラリは，フックした I/O 要求の観測データをメモリ上に採取し，実際にファイルに対して I/O 要求を発行する。

(2) コンストラクトルーチンにおける観測データ保存領域の確保機能:
アプリケーション起動時に呼び出されるライブラリのコンストラクトルーチンを利用して観測データの保存領域をメモリ上に確保する。IOC ライブラリが確保する観測データ保存領域の 1 エントリのサイズを表 2 に示す。

表 2 観測データのサイズ

観測データ	データ長
type	4 bit
time	28 bit
fd	32 bit
data	64 bit

メモリ上により多くの観測データを採取することができるようにするため，I/O 要求の種類を表すビット数を削減し，I/O 要求の発生時刻を差分で保持するようにすることで，観測データのサイズを 128bit に納めた。例えば，観測データ保存領域をメモリ上に 1MByte 確保した場合，最大で 65,536 個の I/O 要求についての観測データを採取することが可能である。なお，観測データ保存領域

のエントリが一杯になった場合，現在は最も古いエントリを上書きするように設定している。

(3) ファイルロックを利用した同一ファイルへの観測データの書き出し機能:
アプリケーションの終了時に呼び出されるライブラリのデストラクトルーチンを利用してメモリ上に採取した観測データをファイルに書き出す。このとき，ファイルを共有しているシステムについては，ファイルロックを利用し，アプリケーションの複数のプロセスが採取した観測データを同一のファイルに対して書き出す。IOC ライブラリによる観測データ書き出しの概要を図 3 に示す。

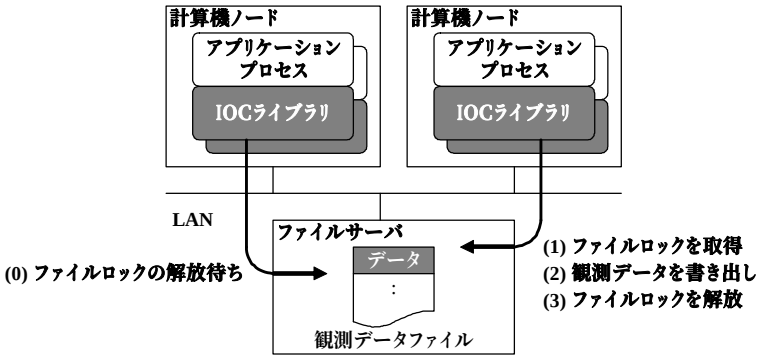


図 3　観測データの書き出し

アプリケーション終了時に IOC ライブラリの終了処理が行なわれる。IOC ライブラリでは，観測データを書き出すファイルを開き，ファイルロックの取得を試みる。ファイルロックを取得できた場合には保持している観測データをファイルの末尾に書き出す。その後，ファイルロックを解放してファイルを閉じ，終了する。

一方，IOC ツールは次の 4 点の機能を備えている。

(1) ある時間に発生した I/O 量の整形:
読み出し要求と書き出し要求の観測データに含まれる I/O 量を積上げ，ある時間に発生した読み出し量と書き出し量を算出してまとめる。これにより，アプリケーション実行時の I/O の発生状況を把握することができる。

(2) ある時間に発生した I/O 回数の整形:
読み込み要求と書き出し要求の観測データを数え上げ，ある時間に発生した読み込み回数と書き出し回数を算出してまとめる。これにより，アプリケーション実行時の I/O の発生状況を把握することができる。

(3) ある I/O 量の発生回数の整形:
観測データに含まれる I/O 量が同じ読み込み要求と書き出し要求を数え上げ，ある I/O 量について発生した読み込み回数と書き出し回数を算出してまとめる。これにより，アプリケーション実行時に発生する支配的な I/O 量を把握することができる。

(4) あるファイルのアクセス状況の整形:
あるファイルについて観測データに含まれる I/O 量やオフセットを積上げ，アプリケーションのファイルに対するアクセス状況をまとめる。これにより，アプリケーション実行時のファイルの再利用状況やプロセス間の共有状況を把握することができる。

2.4 基本評価

ここで，IOC ライブラリの観測コードのアプリケーションに対する負荷を確認するため，ファイルの open，write，read，close を 100 回繰り返すテストプログラムを用意し，繰り返し前後の時間の差分を測定した結果を表 3 に示す。このとき，ファイルに対して書き出すデータと読み込むデータのサイズは 8Byte とした。なお I/O テストの実行は，続く 3 章の科学技術計算アプリケーションの実行と同様に，東京大学情報基盤センターの T2K 東大システム上で実施した。

表 3　I/O テストの測定結果

アプリケーション	測定時間
I/O テスト	2,799 msec
I/O テスト + IOC ライブラリ	2,831 msec

上記の通り IOC ライブラリの I/O 要求の観測による測定時間の増加は 32msec 程度であり，I/O 要求 1 回当たりの観測データ採取オーバヘッドは 0.079msec 程度であった。

3 章では，上記の IOC による科学技術計算アプリケーションのファイル I/O 特性の評価例について述べる。

3. アプリケーションの I/O 特性評価例

3.1 対象アプリケーション

2 章で述べた IOC を使用してファイル I/O の特性を観測する科学技術計算アプリケーションとしては，「戦略的革新シュミレーションソフトウェアの研究開発」プロジェクト[10]において開発された，乱流音場解析ソフトウェア FrontFlow/Blue と大規模タンパク質の密度汎関数法プログラム ProteinDF を使用した。「戦略的革新シュミレーシ

ョンソフトウェアの研究開発」プロジェクトは，流体計算分野や物性計算分野といった幅広い分野の実用的シミュレーションソフトウェアを開発しており，その成果は一般に公開されている。なお，FrontFlow/Blue と ProteinDF は，理化学研究所が次世代スーパーコンピュータのアーキテクチャの検討に向けて選定した 21 種のアプリケーション[11]にも含まれる著名なものである。

3.2 アプリケーションの観測結果

3.1 節で述べた FrontFlow/Blue と ProteinDF について，T2K 東大システム上での観測時に実行した計算問題を表 4 に示す。

表 4　科学技術計算アプリケーション観測時の計算問題

アプリケーション名	計算問題	規模	並列数
FrontFlow/Blue	曲がりパイプ内の流れ場計算	要素数 3 万	4 プロセス
ProteinDF	インスリンの全電子計算	原子数 790	16 プロセス

上記の計算問題実行時の IOC による観測結果を次の通り示す。

(1) 乱流音場解析ソフトウェア FrontFlow/Blue:

FrontFlow/Blue がアクセスしたファイルは 21 個であり，そのサイズは 2.5MByte であった。このとき，IOC により採取した FrontFlow/Blue 実行時のタイムステップごとに発生した I/O 要求の数を図 4 に示す。ここで，横軸はタイムステップ，縦軸は I/O 要求の数を表しており，奥行きは計算プロセスを表している。

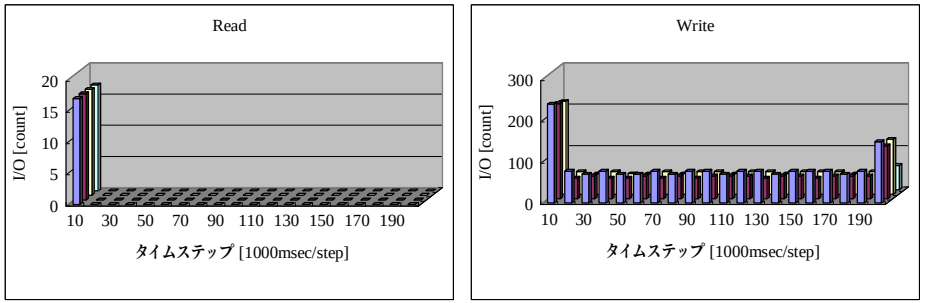


図 4　FrontFlow/Blue 実行時の I/O 要求数

タイムステップごとに発生した I/O 要求の数より，発生した I/O 要求の 70%が実行途中のデータ書き出しであることが分かる。また，IOC により採取した FrontFlow/Blue 実行時の I/O 量ごとの発生回数を図 5 に示す。ここで，横軸は 1 回当たりの I/O 量，縦軸は発生回数を表している。

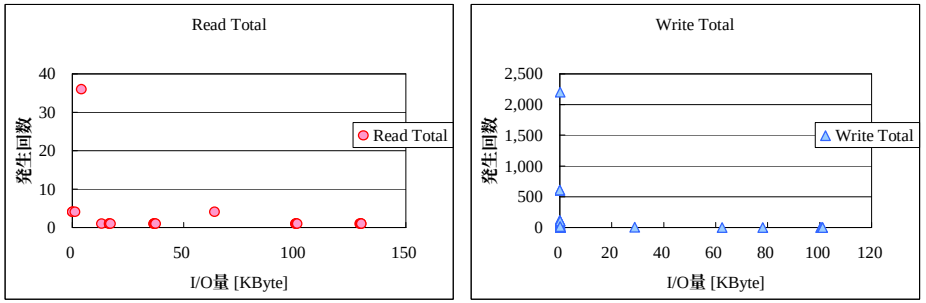


図 5　FrontFlow/Blue 実行時の I/O 量ごとの発生回数

I/O 量ごとの発生回数より，データの読み込みについては 4KByte のアクセスが支配的であり，データの書き出しは 128Byte が支配的であることが分かる。

(2) 大規模タンパク質の密度汎関数法プログラム ProteinDF:

ProteinDF がアクセスしたファイルは 1,226 個であり，そのサイズは 6.3GByte であった。このとき，IOC により採取した ProteinDF 実行時のタイムステップごとに発生した I/O 量を図 6 に示す。ここで，縦軸は I/O 量を表している。タイムステップごとに発生した I/O 量より，ファイルの読み込みを繰り返しながら定期的に多量の I/O が発生していることが分かる。

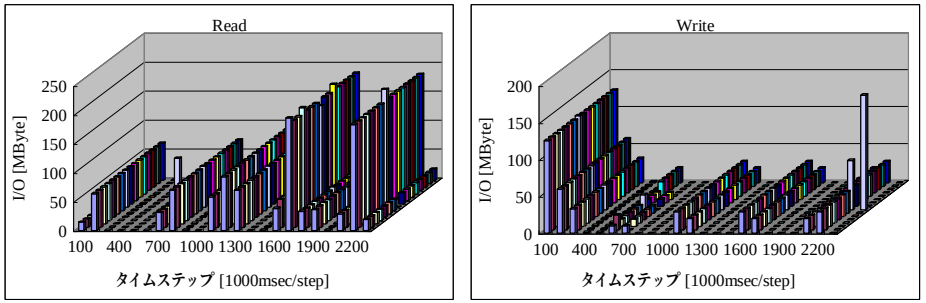
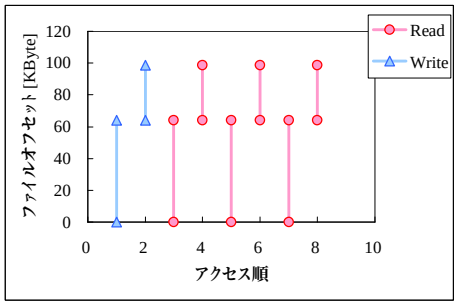


図 6　ProteinDF 実行時の I/O 量

また，IOC により採取した ProteinDF 実行時の一時ファイルのアクセス状況を図 7 に示す。ここで，横軸はファイルに対するアクセス順，縦軸はファイルのアクセス位置を表している。また項目のプロセスは，ファイルにアクセスした計算プロセスを実行していた計算機ノード名とそのプロセス番号を表している。



Read				Write			
アクセス順	オフセット [KByte]		プロセス	アクセス順	オフセット [KByte]		プロセス
	開始	終了			開始	終了	
3	0	64	b198:20604	1	0	64	b199:18894
4	64	99	b198:20604	2	64	99	b199:18894
5	0	64	b198:20604				
6	46	99	b198:20604				
7	0	64	b198:20604				
8	64	99	b198:20604				

図 7　ProteinDF 実行時の一時ファイルアクセス状況

このとき，一時ファイルは，プロセス間のデータの受け渡しに使用されており，あるプロセスが書き出したデータを別のプロセスが繰り返し読み込んでいる。ProteinDF はこのような複数のプロセスでアクセスするファイルを 790 個生成しており，計算機ノードが異なるプロセス間でも同じファイルにアクセスすることがある。

今回の IOC によるファイル I/O の観測は先に示した計算問題について実施したのみであり，上記の結果は FrontFlow/Blue や ProteinDF の一般的なファイル I/O の特徴を示すものではない。ただし今回の IOC による観測から，計算問題によっては少量のファイル I/O が支配的であるケースや，プロセス間で同じファイルにアクセスして繰り返しデータを読み込むケースがあることは分かった。このようなケースについて，提案している Gather-Arrange-Scatter や pdCache は有効に機能すると考える。また，ProteinDF のように大量に一時ファイルが生成されるケースもある。このため，pdCache にファイルの生成を遅延する機能を設けることで，並列ファイルシステムに対するファイルの生成要求の集中を回避し，有効性を向上することができると考える。

4. 関連研究

本稿の対象であるファイル I/O の観測技術に関連した研究を紹介する。

ファイルシステムのプロファイル情報を採取するために Virtual File System(VFS)オペレーションの観測を可能にする技術の研究・開発として FSprof[12]がある。FSprof は，ファイルシステムのソースコードに対して観測コードを追加する方法と，VFS とファイルシステムの間に観測用のスタックファイルシステムを挿入する方法を提供し，より精度の高いファイル I/O の観測を可能としている。

VFS オペレーションの観測を可能にする別の技術の研究・開発として VFS Interceptor[13]がある。VFS Interceptor は，すべてのファイルシステム関数のポインタを検索し，観測コードを呼び出すことができるようにポインタを入れ替えることで，ファイル I/O の観測を可能としている。

また，カーネルレベルのプロファイル情報を採取する技術の研究・開発として Kernel Tuning and Analysis Utilities(KTAU)[14]がある。KTAU は，マクロや関数として観測コードをカーネルに組み込み，カーネルの実行パスに割り込んで観測データを採取している。ファイル I/O だけではなく，ネットワーク等を含めたより詳細な観測を可能としている。

並列アプリケーションのプロファイル情報を採取する技術の研究・開発として Integrated Performance Monitoring(IPM)[15]がある。IPM は，MPI 通信を観測するとともに，Performance API(PAPI)を使用して入手できるハードウェア性能カウンタ情報を併せることで，並列アプリケーションの詳細な観測を可能としている。

以上に対し IOC は，計算機センターで運用されているスーパーコンピュータなどさまざまな環境において一般ユーザによるファイル I/O の観測を可能とする。

5. おわりに

本稿では，並列ファイルシステムに対する負荷を軽減するために提案している手法の有効性を確認するとともに，さらに有効性を向上していくため，アプリケーションのファイル I/O の特性評価を支援する技術 IOC を開発した。IOC を使用することで，ユーザレベルでさまざまなアプリケーションのファイル I/O のアクセス情報を採取することが可能である。また，IOC が採取したファイル I/O の観測データを整形することで，アプリケーションのファイル I/O の分析に向けて次の 4 点を把握することが可能になる。

(1) ある時間に発生した I/O 量
(2) ある時間に発生した I/O 回数
(3) ある I/O 量の発生回数
(4) あるファイルのアクセス状況

実際に T2K 東大システムにおいて科学技術計算アプリケーションである FrontFlow/Blue と ProteinDF に IOC を適用し，各アプリケーションのファイル I/O を観

測した。その結果，今回 IOC を適用した計算問題については，少量のファイル I/O が支配的であることや，プロセス間で同じファイルにアクセスして繰り返しデータを読み込むことを把握でき，提案している手法が有効に機能するケースがあることを確認できた。

今後は，計算問題をより大きくするとともに，さまざまなアプリケーションに対して IOC を適用してファイル I/O の特性を分析し，提案している手法の有効性を向上していく。

謝辞 本研究の一部は，文部科学省「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」からの支援を受けている。

参考文献

1) 姫野龍太郎: 異機種複合システム RSCC(Riken Super Combined Cluster)とその将来計画, 電子情報通信学会技術研究報告, SWoPP 2005, 2005.
2) Habata, S., Yokokawa, M. and Kitakawa, S.: The Development of the Earth Simulator, IEICE transactions on information and system, Vol.E86-D, No.11, pp.1947－1954, 2003.
3) Carns, P. H., Ligon, W. B., Ross, R. B. and Thakur, R.: PVFS: A Parallel File System for Linux Clusters, Proceedings of the 4th annual Linux Showcase & Conference, 2000.
4) Sun Microsystems, Inc.: Lustre File System: High-Performance Storage Architecture and Scalable Cluster File System, White Paper, 2008.
5) Schmuck, F. B. and Haskin, R. L.: GPFS: A Shared-Disk File System for Large Computing Clusters, Proceedings of the Conference on File and Storage Technologies, 2002.
6) Fujita, N. and Ohkawa, H.: Shared Rapid File System on Ethernet for HPC: Central File Server Demonstration, Proceeding of the 2005 IEEE International Symposium on Mass Storage Systems and Technology, 2005.
7) 清水正明, 鵜飼敏之, 三瓶英智, 飯田恒雄, 藤田不二男: スーパーテクニカルサーバ SR11000 の高速分散ファイルシステム HSFS, 情報科学技術フォーラム講演論文, FIT 2005, 2005.
8) Ohta, K., Matsuba, H. and Ishikawa, Y.: Improving Parallel Write by Node-Level Request Scheduling, IEEE International Symposium on Cluster Computing and the Grid, 2009.
9) 太田一樹, 石川裕: ファイルサーバ独立な並列ファイルキャッシュ機構, 情報処理学会研究報告, HOKKE 2009, 2009.
10) 東京大学計算科学技術連携研究センター: RSS21 革新的シミュレーションソフトウェアの研究開発, http://www.ciss.iis.u-tokyo.ac.jp/rss21/index.html.
11) 理化学研究所次世代スーパーコンピュータ開発実施本部: ターゲット・アプリケーション, http://www.nsc.riken.jp/target-application/target-application.html.
12) Joukov, N., Wright, C. P. and Zadok, E.: FSprof: An In-Kernel File System Operation Profiler, Technical Report FSL-04-06, Computer Science Department, Stony Brook University, 2004.
13) Wang, Y., Shu, J., Xue, W. and Xue, M.: VFS Interceptor: Dynamically Tracing File System Operations in real environments, First International Workshop on Storage and I/O Virtualization, Performance, Energy, Evaluation and Dependability, 2008.
14) Nataraj, A., Malony, A. D., Morris, A. and Shende, S.: Early Experiences with KTAU on the IBM BG/L, Proceedings of the European Conference on Parallel Processing, 2006.
15) Borrill, J., Carter, J., Oliker, L., Skinner, D. and Biswas, R.: Integrated Performance Monitoring of a Cosmology Application on Leading HEC Platforms, International Conference on Parallel Processing, 2005.