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

安井 隆† 清水 正明† 堀 敦史‡ 石川 裕§

大規模なクラスタシステムにおけるファイルサーバへの I/O 負荷を低減するため，計算ノードが搭載するメモリと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を設計する．ファイルキャッシュシステムでは，計算ノード間でのファイル単位の分散管理によるアクセスファイル数に見合う I/O 性能の確保と，計算ノードのローカルディスクの活用による大規模ファイルのキャッシュ対応を図る．基本評価の結果，使用した 32 台の計算ノードに見合う I/O 性能が得られたほか，大規模データソート時の実行時間について計算ノード数に対するスケーラビリティが見られ，ファイルキャッシュシステムの現状での有効性を確認している．

Design of User-Level Parallel File Cache System Utilizing Local Disk

TAKASHI YASUI†, MASAAKI SHIMIZU†, ATSUSHI HORI‡ and YUTAKA ISHIKAWA§

A new user-level parallel file cache system using the memory and the local disk of the computing nodes is designed in order to reduce the I/O load of the file server in the large-scale cluster systems. It is possible to scale the I/O performance by increasing the number of the accessed files by the distributed file management among the computing nodes, and to cache the large file by using the local disk of the computing nodes. We have implemented the file cache system. The evaluation results show that the I/O performance is proportional to 32 computing nodes used for the evaluation, and the execution time of large data sorting is scaled by the number of the computing nodes. The effectiveness of the file cache system proposed in this paper is confirmed by those results.

1. はじめに

近年スーパーコンピュータとして大規模なクラスタシステムの導入が進んでおり，従来の計算規模が大きなアプリケーションの実行(Capability Computing)からパラメータスイープのように計算条件を変更したアプリケーションの複数実行(Capacity Computing)まで，さまざまな用途で利用されるようになってきている．例えば，スーパーコンピュータの上位 500 のランキングである TOP500[1]に登録されているシステムの 80%以上がクラスタシステムであり，十万コアを超えるプロセッサを搭載した大規模なクラスタシステムも登録されている．このような大規模なシステムでは，大量のプロセッサコアからの I/O 要求が発生することになり，ファイルサーバに対する負荷が大きくなる．このため，大規模なシステムにおけるファイルサーバの負荷を低減し，アプリケーションのファイル I/O の性能を確保することは重要である．

ファイル I/O の性能を確保する技術として，Lustre File System[2]や IBM 社の General Parallel File System (GPFS)[3]，日立の Hitachi Striping File System (HSFS)[4]といった並列ファイルシステムがある．並列ファイルシステムは，複

† 株式会社 日立製作所 中央研究所
Hitachi, Ltd., Central Research Laboratory
‡ 理化学研究所
RIKEN
§ 東京大学
The University of Tokyo

数のファイルサーバを使用して構成し，各計算ノードから発生するファイル I/O をファイルサーバ間で分散して処理する．このとき，従来の並列ファイルシステムは，大単位のファイル I/O の性能を重視した設計となっていることが多く，多数の計算ノードからの小単位・大量のファイル I/O には十分対応できていない．

このような中，ファイルサーバに対する負荷の低減に向けた取り組みとして，アプリケーションとファイルサーバの間にファイルデータのキャッシュ階層を設けるユーザレベル並列ファイルキャッシュシステム[5]の研究を行なっており，本研究では，計算ノードが搭載するメモリとローカルディスクを活用するファイルキャッシュシステムを設計した．このユーザレベル並列ファイルキャッシュシステムでは，計算ノード間でのファイル単位の分散管理によるアクセスファイル数に見合った I/O 性能の確保と，計算ノードのローカルディスクの活用による大規模ファイルのキャッシュ対応を図った．また，現状での有効性を確認するため，基本評価を行なった．

以下，2章で科学技術計算アプリケーションにおけるファイル I/O について述べる．3章では計算ノードが搭載するメモリと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について述べ，続く4章でその基本評価について述べる．5章では関連研究について紹介し，最後に6章で本稿をまとめる．

2. 科学技術計算アプリケーションにおけるファイル I/O

本章では，実際の科学技術計算アプリケーションにおけるファイル I/O パターンの例について紹介する．

多くの科学技術計算アプリケーションの実行では，各計算ノードからファイル I/O が発生する．このため，クラスタシステムの大規模化にともない，ファイルサーバに対して大量の I/O 要求が行なわれることになる．このとき，どの程度のサイズのファイルをどの程度の頻度でアクセスするかはアプリケーションに依存する．例えば，時間発展問題系のアプリケーションでは，タイムステップ毎に各計算ノードが計算の途中結果をファイルに書き出す．また，メモリ上に確保しきれないデータの一時保存や，プロセス間のデータ共有を目的としてファイルを利用する場合もある．

実際の科学技術計算アプリケーションにおける I/O 要求パターンを把握するため，これまで，アプリケーション実行時のファイル I/O 情報を採取する技術[6]を開発し，乱流音場解析ソフトウェア FrontFlow/Blue[7]や大規模たんぱく質の密度汎関数法プログラム ProteinDF[8]について分析を行なってきた．その結果，FrontFlow/Blue については，I/O 要求の 70%が実行途中のデータの書き出しであり，4KByte 以下の小単位のファイル I/O が支配的であるケースを確認しており，ProteinDF については，定期的に大量のファイル I/O が発生し，大量のファイルを生成して書き出したデータを繰り返し読み込むケースを確認している．また，ゲノム分野のアプリケーションなどで利用されている External Sorting では，400Byte 程度の小単位のレコードからなる，メモリ上に確保しきれない大規模なデータのソートが，一時ファイルに大量のレコードを書き出しながら行なわれている．

これに対し，アプリケーションのファイル I/O 性能を確保する技術として，複数のファイルサーバを使用して構成する並列ファイルシステムがある．ただし，従来の並列ファイルシステムは大単位のファイル I/O の性能を重視した設計となっており，上記のような小単位のファイル I/O が支配的なケースでのファイル I/O の性能確保が困難になっている．また，システムの大規模化においてはファイルサーバよりも計算ノード数の増加が大きく，上記のように大量にファイルを生成するケースや大量にファイル I/O を行なうケースではファイルサーバへのアクセスの集中が発生し，性能の確保が難しくなっている．

この並列ファイルシステムが苦手とするファイル I/O によるファイルサーバの負荷を低減するため，計算ノードが搭載するメモリを利用し，アプリケーションとファイルサーバの間でファイルのデータをキャッシュするファイルキャッシュシステムを設計してきた．これまでに設計してきたファイルキャッシュシステムの概要を図 1 に示す．

これまでに設計してきたユーザレベル並列ファイルキャッシュシステム[5]では，ユーザレベルでファイルキャッシュ階層を実現し，ユーザが研究室の PC クラスタシステムからセンタ運用されているスーパーコンピュータまで，構成や運用ポリシが異なるさまざまなシステム上でファイルキャッシュ

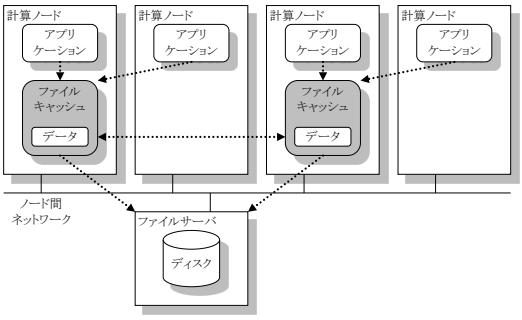

図 1　ファイルキャッシュシステムの概要

システムを導入することができるよう，システムソフトウェアやアプリケーションからの独立性を確保していた．また，計算ノード間で連携したファイルキャッシュ管理を実現し，ファイルキャッシュ階層でファイルの存在情報やデータを保持することでファイルサーバに対する I/O 要求の多重度の削減や I/O 要求の不連続性の抑制を図っていた．実際に，IOFSL[9]をベースとして実装したユーザレベル並列ファイルキャッシュシステムを T2K 東大システム[10]で評価した結果，大量に生成される一時ファイルへのアクセスがキャッシュ上で処理されることで，ProteinDF について最大 1.5 倍程度の実行時間の速度向上が見られていた．

このとき，先で述べたように，科学技術計算アプリケーションのファイル I/O のパターンではプロセス間で同じファイルにアクセスすることがあるが，ファイルキャッシュシステムのように，計算ノードのメモリ上にファイルデータのキャッシュを保持する場合，計算ノード間でキャッシュのコンシステンシを維持する必要がある．このコンシステンシを維持する処理のため，使用する計算ノードの数に見合った速度向上が得られないことがある．また，科学技術計算アプリケーションではメモリ上に配置しきれない大規模なファイルにアクセスするケースがあるが，計算ノードのメモリ上でファイルのデータをキャッシュする場合，キャッシュが溢れることになる．このため，キャッシュするデータの入れ替えが発生し，ファイルサーバに対する負荷の低減が難しくなる．

3. 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

3.1 スケーラビリティの確保と大規模ファイルへの対応に向けた課題とアプローチ

本研究では，使用する計算ノードがアクセスするファイルの数に見合った I/O 性能を確保し，大規模なファイルに対応するユーザレベル並列ファイルキャッシュシステムを設計する．このとき，ユーザレベル並列ファイルキャッシュシステムの実現に向けては次の 2 点が課題となる．

(1) ファイル共有とスケーラビリティの両立:
アプリケーションによっては実行しているプロセス間で同じファイルにアクセスすることがある．このため，ユーザレベル並列ファイルキャッシュシステムは，複数の計算ノード間で同じファイルのキャッシュを共有できるようにする必要がある．このとき，複数の計算ノード間でキャッシュのコンシステンシを維持することが必要となるが，使用する計算ノードがアクセスするファイルの数に見合った性能を得られるように個々のファイルの共有を実現する．

(2) 大規模なファイルに対する I/O 性能の確保:
アプリケーションによってはメモリ上に配置しきれない大規模なファイルにアクセスすることがある．このため，ユーザレベル並列ファイルキャッシュシステムは，計算ノードとファイルサーバとの間でのキャッシュするデータの入れ替えを極力回避する必要がある．

これらの課題に対して，本研究では次の 2 点のアプローチを採る．

(1) ファイルに対するキャッシュ保持ノードの括り付け:
ユーザレベル並列ファイルキャッシュシステムにおいて，あるファイルのデータをメモリ上にキャッシュする計算ノードを一意に固定する．データをキャッシュするファイルと計算ノードを括り付けることで，計算ノード間でのキャッシュのコンシステンシを維持するための処理を不要とする．計算ノード間でファイルを単位とした分散管理を行なうことで，使用する計算ノードがアクセスするファイルの数に見合った性能を得られるようにする．

(2) キャッシュ対象ファイルのローカルディスクへのコピー:
ユーザレベル並列ファイルキャッシュシステムにおいて，計算ノードのメモリ上にファイルをキャッシュする際に，対象ファイルを計算ノードが搭載するローカルディスク上へコピー

する．ローカルディスク上へファイルをコピーし，コピーしたファイルを利用してファイルのデータをメモリ上にキャッシュすることで，計算ノードでキャッシュが溢れた際のキャッシュの入れ替えにともなうファイルサーバへのアクセスを削減する．

本研究では，上記のアプローチにもとづいて，計算ノードが搭載するメモリと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を設計した．

3.2 システムの設計

3.1節で述べたアプローチにもとづいたユーザレベル並列ファイルキャッシュシステムの構成を図2に示す．

ユーザレベル並列ファイルキャッシュシステムでは，アプリケーションを実行するプロセス中にファイルキャッシュを実現するためのスレッドを生成し，このスレッドが計算ノード間で連携してファイルを計算ノード間で分散してキャッシュする．このファイルを分散してキャッシュする機能は，ファイルの分散管理機能，ファイルのコピー機能，ファイルのキャッシュ機能の大きく 3 つの処理からなっている．次にそれぞれの機能について説明する．

3.2.1 ファイルの分散管理機能

ファイルの分散管理機能は，アプリケーションのI/O要求が対象とするファイルの名前をキーとしてファイルのデータをキャッシュする計算ノードをファイル管理ノードとして選択する．具体的には，ハッシュ関数を使用してファイルの名前と計算ノードを括り付ける．ファイルの分散管理機能の処理を図3に示す．

ファイル管理ノードのスレッドは，ハッシュ関数

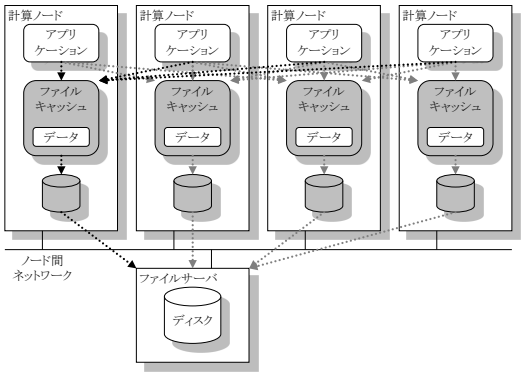

図 2　ファイルキャッシュシステムの構成

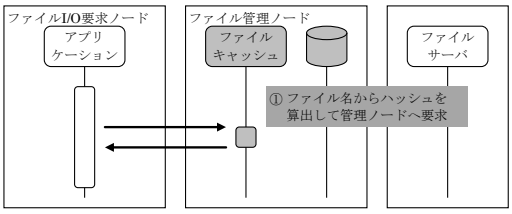

図 3　分散管理機能の処理フロー

によって括り付けられたファイルに対するアプリケーションからのI/O要求を受け取り，その後，コピー機能やキャッシュ機能を使用して，ローカルディスク上へのファイルのコピーやメモリ上へのファイルのデータのキャッシュを行なう．このとき，ファイル管理ノードではアプリケーションの I/O 要求を到着した順に処理し，あるファイルのデータについて，あるアプリケーションのI/O要求で更新された結果が，以降のアプリケーションのI/O要求で参照されるようにする．これにより，ファイルキャッシュシステム上でのコンシステンシを維持する．

3.2.2 ファイルのコピー機能

ファイルのコピー機能は，アプリケーションがファイルをオープンする際に，ファイルサーバからファイル管理ノードが搭載するローカルディスク上へファイルをコピーし，コピーしたファイルを利用してファイルのデータをファイル管理ノードのメモリ上でキャッシュすることを可能にする．ファイルのコピー機能の処理を図4に示す．

ファイルのデータをキャッシュするファイル管理ノードのスレッドは，アプリケーションのオープン要求を受け取り，その後の対象ファイルへのI/O要求に先立ってファイルサーバからローカルディ

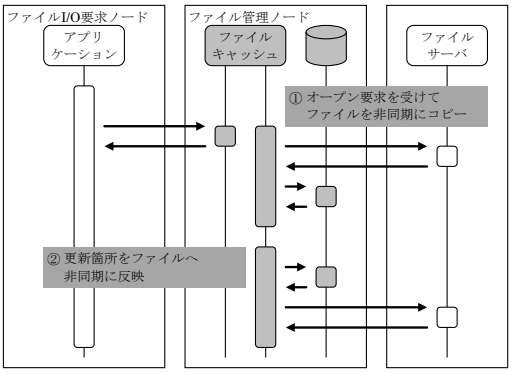

図 4　コピー機能の処理フロー

スク上へのコピーを非同期に開始する．また，メモリ上のキャッシュで更新されているデータについては，ファイルサーバのファイルへ非同期に反映する．なお，アプリケーションの終了時には，メモリ上のキャッシュで更新されているデータは，ファイルサーバのファイルへすべて反映する．

3.2.3 ファイルのキャッシュ機能

ファイルのキャッシュ機能は，アプリケーションがファイル I/O する際に，ファイル管理ノードが搭載するローカルディスク上へコピーされているファイルからデータをメモリ上にキャッシュする．ファイルのキャッシュ機能の処理を図 5 に示す．

ファイルのデータをキャッシュするファイル管理ノードのスレッドは，アプリケーションの I/O 要求を受け取り，ローカルディスク上へコピーされているファイルからデータをメモリ上にキャッシュした上で I/O 処理を行ない，処理結果を返す．このとき，ローカルディスク上へのファイルのコピーは，コピー機能により非同期に行なわれるが，ファイルのすべてのデータのコピー終了を待つことなく，I/O 要求が対象とする部分がコピーされた時点でメモリ上にデータをキャッシュする．また，キャッシュ上で更新されているデータについては，キャッシュの使用状況にしたがって，ローカルディスクのコピーファイルへ非同期に反映する．

3.3 システムの実装

本研究では，3.2節で述べた構成を採るユーザレベル並列ファイルキャッシュシステムを Linux 上で動的リンクライブラリとして実装した．ユーザレベル並列ファイルキャッシュシステムは，Linux の機能を利用してアプリケーションの実行時にプリロードされ，アプリケーションのプロセス中にファイルキャッシュを実現するためのスレッドを生成する．また，アプリケーションの I/O 要求をフックして対象ファイルのデータのキャッシュを行なう．

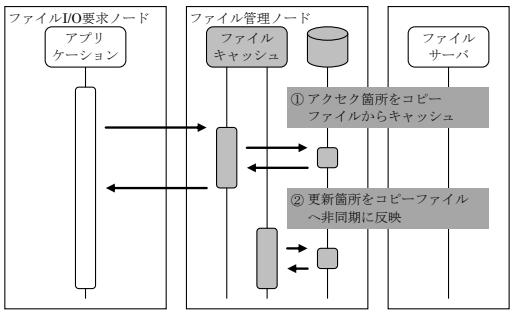

図 5 キャッシュ機能の処理フロー

4. 基本評価

4.1 評価環境

3章で述べたユーザレベル並列ファイルキャッシュシステムの現状での有効性を確認するため，T2K 東大システムにおいて基本評価を実施した．T2K 東大システムにおいて使用した計算ノード環境を表 1 に示す．

T2K 東大システムは，計算ノード間を Myri-10G で接続し，ファイルシステムとしては NFS および Lustre，HSFS が提供されている．評価では，この T2K 東大システムの計算ノードを最大 32 台使用した．

4.2 スケーラビリティ評価

本研究で設計したユーザレベル並列ファイルキャッシュシステムの計算ノードがアクセスするファイルの数に対するスケーラビリティを確認するため，T2K 東大システムの複数の計算ノードを使用し，ファイルがキャッシュされる場合のファイル I/O 性能の測定を実施した．測定では，各計算ノードでファイル I/O を行なうプロセスを起動し，各プロセスが 1GByte のファイルに対して 4KByte，64KByte，1MByte 単位で Write，Read する場合の性能を測定した．測定結果を図 6，図 7 に示す．

図 6, 図 7 は，複数の計算ノードを使用して Write, Read した場合の計算ノード当たりの性能をそれぞれまとめたものであり，横軸は使用した計算ノードの数を示し，縦軸は計算ノード当たりの性能を MB/s 単位で示している．

図 6，図 7 より，本研究でファイル共有とスケーラビリティの両立を図ったユーザレベル並列ファイルキャッシュシステムは，使用する計算ノードが

表 1 T2K 東大システムの計算ノード環境

計算ノード (日立HA8000-tc/RS425)	
CPU	Quad-Core Opteron 2.3GHz × 4
メモリ	32GB
ローカルディスク	250GB RAID1
ネットワーク	Gigabit Ethernet / Myri-10G
OS	Red Hat Enterprise Linux 5.1
MPIライブラリ	MPICH-MX
ファイルシステム	NFS / Lustre / HSFS

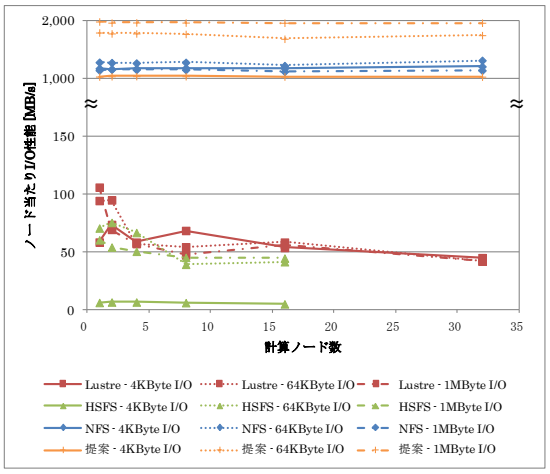

図 6　計算ノード当たりの Write 性能

増加した際にも計算ノード当たりのファイル I/O 性能は，4KByte 単位での I/O 時で 1GB/s 程度とほぼ一定となっており，使用する計算ノード数に見合った性能が得られていることが分かる．また，Write 性能については，ファイルシステムによるキャッシュの効果が現れている NFS と同等の性能が得られ，Read 性能についても，ファイルシステムによるキャッシュの効果が現れている Lustre と同様の性能が得られている．これは，計算ノード間でのキャッシュのコンシステンシを維持するための制御処理がなく，各計算ノードで対象ファイルをキャッシュすることができたためである．

4.3　大規模ファイル評価

本研究で設計したユーザレベル並列ファイルキャッシュシステムの大規模ファイルに対するファイル I/O 性能を確認するため，ゲノム分野のアプリケーションなどで利用されている External Sort-ing の GNU sort による模擬評価を実施した．GNU sort は，ソートの途中結果を一時ファイルに書き出しながらメモリ上に確保しきれない大規模なデータを処理する．測定では，T2K 東大システムの複数の計算ノードを使用して，1 レコードを 400Byte とし，各計算ノードのプロセスが 1 億レコード 40GByte のデータが保存されているファイルを入力として 10GByte のメモリでソートする場合の時間を測定した．なお，測定時の GNU sort の動作を図 8 に示す．

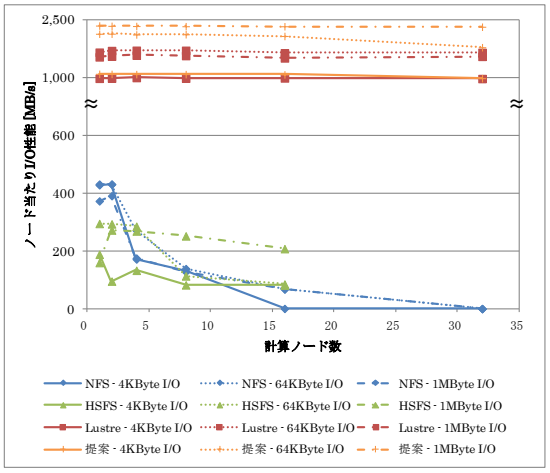

図 7　計算ノード当たりの Read 性能

GNU sort は，40GByte の入力ファイルからデータを読み込みながら 10GByte 分のソートを行ない，その結果を書き出した 4 つの 10GByte の一時ファイルを作成していた．また，GNU sort は，4 つの 10GByte の一時ファイルからデータを読み込みながらソートを行ない，最終的な結果を書き出した 40GByte の出力ファイルを作成していた．

T2K 東大システムの計算ノードを使用した GNU sort 測定結果を図 9 に示す．

図 9 より，本研究で設計したユーザレベル並列ファイルキャッシュシステムは，Lustre と比較して速度が低くなっているものの，使用する計算ノードが増加した際の速度は Lustre のような低下は見られず，使用する計算ノード数に対するスケーラビリティが得られていることが分かる．これは，計算ノードが搭載するローカルディスクを活用することで，計算ノードからファイルサーバへのソートの途中結果の書き出しが削減されたためと考えられる．ただし，1 台の計算ノードが搭載するローカルディスクのファイル I/O 性能は 40MB/s 程度であるの対

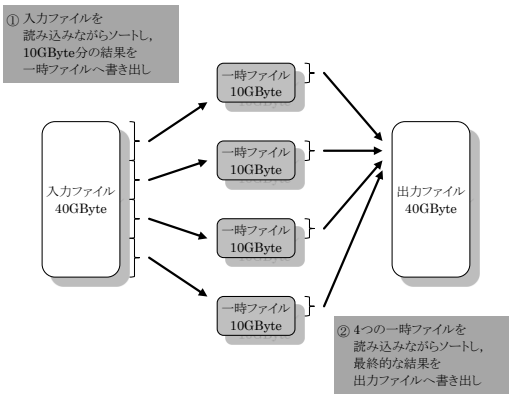

図 8　GNU sort の動作概要

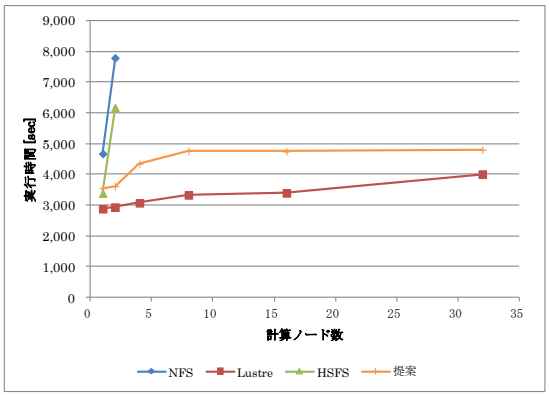

図 9　GNU sort 実行時間

し，32 台の計算ノードによる Lustre に対するファイル I/O 性能は 1.4GB/s 程度であり，このため，ユーザレベル並列ファイルキャッシュシステムの速度は，Lustre と比較して 32 台の計算ノードを使用した場合でも低下が見られていると考えられる．ユーザレベル並列ファイルキャッシュシステムは，大規模なファイルにアクセスするアプリケーションを実行する計算ノードが増え，ファイルサーバに対する負荷が増大するケースで効果を発揮するものと考える．

5.　関連研究

ファイル I/O の性能を確保する技術として，ファイルステージングがある．ファイルステージングは，理化学研究所の RIKEN Super Combined Cluster System[11]や海洋研究開発機構の Earth Simulator[12]が採用している技術であり，研究・開発としては STG[13]や Catwalk[14]がある．ファイルステージングでは，計算ノード間で共有してアクセスすることができるファイルシステムと計算ノードが個別に高速にアクセスすることができるローカルファイルシステムの 2 つのファイルシステムを想定する．ファイルステージングは，アプリケーションの実行に先立って共有ファイルシステム上のファイルをローカルファイルシステムへコピーインし，アプリケーションの実行終了後にローカルファイルシステム上の作成ファイルや変更ファイルを共有ファイルシステムへコピーアウトする．このとき，高速なローカルファイルにコピーインされたファイルは計算ノード間のプロセスで共有することができないため，ファイルを介してデータの受け渡しを行なうようなアプリケーションでは利用できない．本研究で設計したユーザレベル並列ファイルキャッシュシステムでは，ローカルディスクへコピーするファイルと計算ノードを括り付け，プロセスのファイル I/O 要求は対象ファイルを管理する計算ノードで処理するため，計算ノード間のプロセスで同じファイルにアクセスすることが可能である．

アプリケーションが処理するファイルのデータのキャッシュに関する研究・開発として I/O Delegate Cache System (IODC)[15]がある．IODC は，ファイルのデータをキャッシュする専用ノードを用意し，アプリケーションからの MPI-IO を使用したファイルに対する I/O 要求をその専用ノードが代行する．本研究で設計したユーザレベル並列ファイルキャッシュシステムは MPI-IO に非依存であり，さまざまなアプリケーションに適用可能である．

また，オブジェクトベースのストレージを対象とする協調型キャッシュシステムの研究・開発として Dynamic Weight-based Cooperative Caching Scheme (DWC²)[16]がある．DWC² は，オブジェクトデータに対して重み付けを行ない，その重みにしたがってキャッシュしているオブジェクトデータの移譲や破棄を決定することでキャッシュ効率を高める．本研究で設計したファイルキャッシュシステムでは今後，アプリケーションのファイル I/O の特性を反映できるようなキャッシュ機能の拡張を検討し，キャッシュの有効性の向上を図っていく．

6.　おわりに

本研究では，使用する計算ノードがアクセスするファイルの数に見合った性能を確保し，大規模なファイルに対応するユーザレベル並列ファイルキャッシュシステムを設計した．ユーザレベル並列ファイルキャッシュシステムの実現に向けては次の 2 点が課題であった．

(1)　ファイル共有とスケーラビリティの両立
(2)　大規模なファイルに対する I/O 性能の確保

これらの課題に対して，本研究では次の 2 点のアプローチを採り，計算ノードが搭載するメモリと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を設計し，Linux 上で実装した．

(1)　ファイルに対するキャッシュ保持ノードの括り付け
(2)　キャッシュ対象ファイルのローカルディスクへのコピー

実際に，実装したユーザレベル並列ファイルキャッシュシステムを T2K 東大システムの計算ノードで動作させ，基本評価を実施した．複数の計算ノードを使用したファイル I/O 性能を測定した結果，使用した 32 台の計算ノードに見合うファイル I/O 性能が得られることを確認した．また，GNU sort を使用して大規模なデータのソートの実行時間を測定した結果，計算ノードが搭載するローカルディスクの活用により，使用する計算ノード数に対するスケーラビリティが得られることを確認した．

今後，さまざまな実行環境，アプリケーションでの評価を行なっていき，ユーザレベル並列ファイルキャッシュシステムの有効性について確認していく．

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] TOP500.Org，“TOP500 Supercomputing Sites，” http://www.top500.org/.

[2] Sun Microsystems, Inc.，“Lustre File System，” High-Performance Storage Architecture and Scalable Clustre File System，White Paper，2008.

[3] F. B. Schmuck and R. L. Haskin，“GPFS: A Shared-Disk File System for Large Computing Clusters,”Proceedings of the Conference on File and Storage Technologies，2002.

[4] 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.

[5] 安井 隆，松葉浩也，太田一樹，住元真司，鴨志田良和，堀 敦史，石川 裕，“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 22 回コンピュータシステム・シンポジウム (ComSys 2010), 2010.

[6] 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，“ファイル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,” 2009 年並列/分散/協調処理に関する『仙台』サマー・ワークショップ (SWoPP 仙台 2009)，2009.

[7] Y. Guo，C. Kato and Y. Yamade，“Basic Features of the Fluid Dynamics Simulation Software “FrontFlow/Blue”,”SEISAN KENKYU，Vol. 58，No. 1，pp. 11－15，2006.

[8] T. Inaba and F. Sato，“Development of Parallel Density Functional Program Using Distributed Matrix to Calculate All-Electron Canonical Wavefunction of Large Molecules,”Journal of Computer Chemistry，Vol. 28，No. 5，pp. 984－995，2007.

[9] N. Ali, P. Carns, K. Iskra, D. Kimpe, S. Lang, R. Latham, R. Ross，L. Ward and P. Sadayappan，“Scalable I/O Forwarding Framework for High-Performance Computing System,”Proceedings of the 2009 IEEE International Conference on Cluster Computing，New Orleans，LA，USA，2009.

[10] 東京大学情報基盤センター，“HA8000 クラスタシステム,”http://www.cc.u-tokyo.ac.jp/service/ha8000/.

[11] 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.

[12] S. Habata，M. Yokokawa and S. Kitakawa，“The Development of the Earth Simulator,”IEICE transactions on information and system，Vol. E86-D，No. 11，pp. 1947－1954，2003.

[13] 松葉浩也，堀 敦史，石川 裕，“高性能クラスタのための高速汎用ステージングソフトウェア,”先進的計算基盤システムシンポジウム (SACSIS)，pp. 239－246，2009.

[14] A. Hori, Y. Kamoshida, H. Matsuba, K. Ohta, T. Yasui, S. Sumimoto and Y. Ishikawa，“On-Demand File Staging System for Clusters,”IEEE International Conference on Cluster Computing 2009，2009.

[15] A. Nisar，W. Liao and A. Choudhary，“Scaling Parallel I/O Performance through I/O Delegate and Caching System,”Proceedings of the 2008 ACM/IEEE Conference on Supercomputing, Piscataway, NJ, USA, 2008.

[16] Q. Wei, B. Veeravalli and L. Zeng，“DWC^2: A Dynamic Wight-based Cooperative Caching Scheme for Object-based Storage Cluster,”Proceedings of the 2008 IEEE International Conference on Cluster Computing, Tsukua，Japan，2008.