ファイルへのアクセスの自動分散を行うグリッド用分散ファイルシステム

佐藤　仁[†]　松岡　聡[†,††]　中田秀基[†††,†]

HPC クラスタやグリッドなどの並列計算環境では，アプリケーションによっては，ファイルを保持するノードへのアクセス集中が発生し，実行性能の低下が問題となる．既存の分散ファイルシステム上でこのようなアクセス集中を避けるためには，ユーザがアプリケーションの作成時や実行時に明示的にファイルアクセスの分散を行うことが必要となる．しかし，環境が不均質であるグリッドではこのような対応は困難であり負担が大きい．我々は，ファイルシステム側でアクセスの集中を検知し，ファイル複製を積極的に利用して，ファイルへのアクセスを分散する手法を提案し，プロトタイプとしてこの提案手法をグリッドファイルシステムである Gfarm 上に実装した．実験として，作為的にファイルへのアクセス集中が発生する状況を作り出し，プロトタイプによって自動的にアクセス集中を検知し，ファイル複製を作成することで，ファイルへのアクセスが分散され，ファイル複製を作成しない場合と比較して，2.33 倍のファイルアクセスの性能が向上することを確認した．

Distributed File System with Automatic File Access Distribution for the Grid

Hitoshi Sato,[†] Satoshi Matsuoka[†,††] and Hidemoto Nakada[†††,†]

In the parallel computing environment like HPC Cluster or the Grid, some application involves large overhead due to the access concentration on the node that maintains the file. To avoid this problem on the traditional distributed file system, users have to distribute the file access manually. However, it is hard and difficult for users to do such file access distribution on the Grid environment because of its resource heterogeneousness. We propose an automatic file distribution scheme using the access concentration detection on the file system and the file replication. We implement this prototype on Gfarm, which is a file system for grid environment, and evaluate its performance. The results showed that our prototype is 2.33 times faster than Gfarm in the file concentration situation.

1. はじめに

近年, HPC クラスタ技術やネットワーク技術等の発達により, グリッドに代表される大規模な並列計算環境が実用的になりつつある．特に, 高エネルギー物理学, 天文学, 生物学, 地震工学などの分野の科学技術計算においては, このような計算環境を利用して, 大規模なデータを複数の異なる組織間で共有し, 解析を行うなどの試みが積極的に行われている．

例えば，高エネルギー物理学の分野では，Large Hadron Collider(LHC) 実験プロジェクト[1] において，測定器から生成される年間ペタバイトオーダーのデータを数十カ国規模，数千人規模の素粒子物理学者が共有，解析などを行うために，適切に蓄積，処理する必要があり，このような環境を実現するための基盤として，グリッド技術に関する研究の必要性が叫ばれている．

このようなグリッド環境で複数のプログラムを連携するには，ファイルシステムをベースにするのが望ましいと考える．これは，既存のプログラムをそのまま利用でき，多くのプログラマがこのモデルで慣れ親しんでいるからである．しかし，このようなファイルシステムをグリッド上で実際に運用しようとするとうまくいかない場合がある．これは，グリッド環境が大規模で不均質であるので，ある特定のノード及びファイルへのアクセスが時間的に近接するため，ファイルアクセスの実行性能が低下し，アプリケーション全体の実行にも影響を及ぼすことが問題となるためである．HPC クラスタ上の既存の分散ファイルシステム上でこ

† 東京工業大学
Tokyo Institute of Technology
†† 国立情報学研究所
National Institute of Informatics
††† 産業技術総合研究所
National Institute of Advanced Industorial Science and Technology

のようなアクセス集中によるアプリケーションの実行性能の低下を避けるためには, ユーザがアプリケーションの作成時や実行時に明示的にファイル複製やノードへのアクセスの分散等を行うことが必要である. しかしながら, 計算環境が動的に変化し, ある特定の計算環境を想定することが難しいグリッド環境では, アプリケーション側でこのような対応を行うことは困難であり, また, ユーザへの負担の増加が問題となる.

我々は, ファイルシステム側でアクセスの集中を検知し, ファイル複製を積極的に利用して, ファイルへのアクセスを分散する手法を提案し, プロトタイプとしてこの提案手法の機能をグリッドファイルシステムである Gfarm 上に実装した. 実験として, 作為的にファイルへのアクセス集中が発生する状況を作り出し, プロトタイプによって自動的にアクセス集中を検知し, ファイル複製を作成することで, ファイルへのアクセスが分散され, ファイル複製を作成しない場合と比較して, 2.33 倍のファイルアクセスの性能が向上することを確認した.

2. 関 連 研 究

既存のグリッド環境は, HPC クラスタを複数統合して構成されることが多い. また, 典型的な HPC クラスタは, NFS や Andrew File System(AFS), Coda[2] などの既存のネットワークファイルシステムで構成されることが多い. グリッド環境でもこのようなファイルシステムが存在すると, ユーザに対してシングルシステムイメージでのファイルアクセスを提供できるため, 高い利便性を実現できると考えられる. グリッド環境のファイルシステムとして実現すべき要件としては, 異なるサイト間でも安全なデータ共有ができること, 大規模計算のためのスケーラビリティを備えていることが挙げられる. しかしながら, 従来のネットワークファイルシステムでは, これらの実現に問題があると考えられる.

安全性を実現するファイルシステムとして, GSI-SFS[3] が挙げられる. これは, 基盤技術として標準的なグリッド認証技術である Grid Security Infrastructure(GSI)[4] や NFS を基盤としたセキュアなファイルシステムである Self-certifying File System(SFS)[5] が用いられており, 高い安全性やユーザ利便性を提供するが, NFS の拡張であるためスケーラビリティの実現に問題があり, 単一ノードでファイルを共有することには限界があると考えられる.

スケーラビリティをファイルシステムで実現する手法として, ファイルをストライピングして異なるディスクに格納する方法 (並列ストライピングファイルシステム) やローカル I/O を積極的に利用する方法が挙げられる.

前者の手法を実現したファイルシステムとして, 主にクラスタ型計算機での利用を想定した, PVFS[6],Lustre[7], Google File System[8] が挙げられる. これらのファイルシステムでは, ファイルをチャンクに分割することで, 単一ファイルの連続読み出し時に複数ディスクの利用を可能にしている. このため, 高バンド幅なディスク I/O が実現し, また, ファイルアクセスの均一化や分散化を実現する. しかしながら, グリッド環境にこのような特定のファイルシステムで静的で恒久的に構成された環境を想定することは, グリッドが「地理的に, 組織的に広範囲に分散した資源を仮想化し, 必要に応じて動的に共有するための基盤技術」である観点から望ましくないと考える. また, I/O バンド幅がネットワークバンド幅に制限される点も問題となる.

後者の手法を実現したファイルシステムとして, Gfarm[9] が挙げられる. Gfarm は, 主に大規模データを用いる科学技術計算 (データ・インテンシブ・コンピューティング) を対象にした分散ファイルシステムである. Gfarm は並列ストライピングファイルシステムを拡張したものであるが, Gfarm 上の 1 つのファイルはローカルディスク上の複数のファイルから構成され, ファイル単位でディスクに分散格納される. データ・インテンシブ・コンピューティングでは, 数多くのファイルに対し, 同じプログラムで処理を行うという「ファイルアクセスの局所性」が存在する. Gfarm では, この性質を利用するために, CPU とディスクを統合し, ファイルのあるノードに対して計算ジョブを投入することで, ディスク I/O を積極的に利用し, ネットワークバンド幅の制限を受けることを避ける.

前者の手法を実現したファイルシステムは, ファイルの読み出し時に, 複数のディスクを利用してチャンクを読み出しファイルを構成しなければならないため, データ・インテンシブ・コンピューティングで求められるファイルアクセスの局所性が利用できないという問題がある. 一方, 後者の手法を実現したファイルシステムでは, データ・インテンシブ・コンピューティングにおいてファイルアクセスの局所性を利用した効率的な処理が可能になるという長所がある反面, 単一ファイルへのアクセスが向上しないという短所がある.

3. 提 案 手 法

本研究では, グリッド環境でのアクセス集中によるファイルアクセスの性能低下を避けるためにファイルの複製を作成し, ファイルシステム側でファイルアクセスを制御し, アクセス集中を避ける手法を提案する. この手法では, まず, ファイルシステム側で提供される名前空間上のファイル対してファイルの複製を用意する. ファイルアクセスの際は, アプリケーションは名前空間上のファイルへのアクセスを行うよう試みるが, 実際のファイルアクセスは, ファイルの複製のう

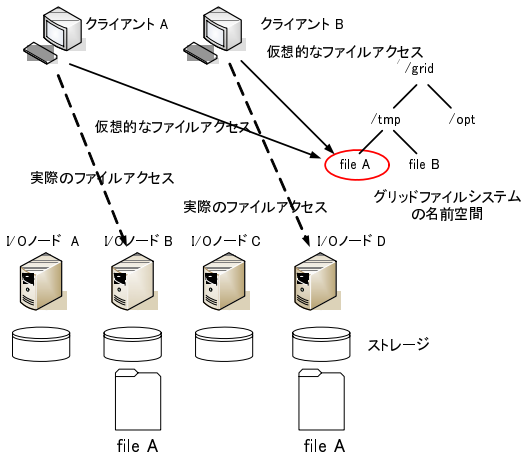

図 1 提案手法によるファイルへのアクセス分散

ちのいづれかにアクセスするようにする．この動作を図 1 を用いて説明する．クライアント A とクライアント B がファイルシステム上の/grid/tmp/fileA へアクセスを試みるが，実際には，クライアント A が I/O ノード B 上の fileA に，クライアント B が I/O ノード D 上の fileA へアクセスする．このような動作を行うことで，ファイルへのアクセスの分散を実現する．

提案システムが想定するファイルシステムとしては，並列ストライピングファイルシステムの構成で多くみられるような，実際にファイルアクセスを行うクライアント，ファイルシステムのメタデータを扱うメタデータサーバ，実際にファイルやファイルのチャンクを格納する I/O ノードから構成されるものとする．このような構成をとるファイルシステムでは，クライアントからメタデータサーバへアクセスするファイルあるいはチャンクの所在を問い合わせ，得られた情報を元に，実際のファイルあるいはチャンクへのアクセスを行う．ファイルの所在に関する情報がメタデータサーバの部分で一元的に管理されているため，メタデータサーバでファイルアクセス時にファイルの実体へのアクセスを制御することで，アクセス集中の起きている I/O ノードへのファイルアクセスを避けるようなスケジューリングを行うことが可能であり，また，このような制御によりファイルアクセスの性能向上が期待できると考える．

4. Gfarm

3 章の提案手法のプロトタイプをグリッド用分散ファイルシステムである Gfarm 上にファイルアクセス分散機能を追加することで実現することを考える．これは，Gfarm が並列ストライピングファイルシステムにみられるように，クライアント，メタデータサーバ，I/O ノードからファイルシステムが構成されている点，グリッド上のファイルシステムとして実現すべき項目である利便性，安全性，スケーラビリティを備えている点，また，ファイル複製作成のための API を提供しているため提案手法のプロトタイプの実装が容易である点などから提案手法の実現に適していると判断した．この章では，プロトタイプの基盤となるグリッド用ファイルシステムである Gfarm について説明する．

4.1 Gfarm ファイルシステム

Gfarm は，産業技術総合研究所グリッド研究センターを中心に開発されている大規模データを用いる科学技術計算のための並列ファイルシステムであり，ネットワーク上の計算機クラスタを統合してグリッドとして提供するためのグリッドミドルウェアである．

Gfarm は，ファイルシステムノードと呼ばれる計算機クラスタのノードのローカルファイルシステムを利用して構成される．ファイルシステムノードは，計算ノードとストレージノードを兼ね，ノード上で I/O を担うデーモン (gfsd) が動作する．また，ファイルの所在や状態などに関するファイルシステムのメタデータは，メタデータサーバ (gfmd) により管理される．ただし，現在の Gfarm の実装 (Gfarm Version 1.0.4) では，メタデータの管理は LDAP により行われている．

Gfarm は，並列ストライピングファイルシステムを拡張したファイルシステムであるため，Gfarm 上のファイルは分割されファイルシステム上へ格納される．ファイル分割は，一定のブロックサイズによる分割だけでなく，それぞれの断片のサイズは自由に動的に決定することができる．また，複数のファイルを統合して一つの仮想ファイルを構成することができ，仮想ファイルに対して，プログラムの処理を行うことができるため，大規模データを用いる科学技術計算に多くみられる「ファイルアクセスの局所性」を利用した効率的な処理を可能にする．

4.2 Gfarm でのファイルアクセスの動作

この節では，Gfarm でのファイルアクセスの動作を簡単に説明する．Gfarm でのファイルアクセスは次のように動作する．

(1) クライアント上で，Gfarm ファイルシステムへアクセスするための API が呼ばれる．
(2) クライアントからメタデータサーバにアクセスするファイルの所在に関するクエリを行い，ファイル複製を保持するホスト名などのノード情報を取得する．ファイルの複製が作成されている場合は，これらのノード情報は複数 (ファイル複製の数だけ) 取得される．
(3) メタデータサーバから取得したノード情報からクライアントのローカルディスクにアクセスするファイルが存在する場合は，そのファイルにアクセスする．
(4) クライアントのローカルディスクにアクセスするファイルが存在しない場合は，メタデータサー

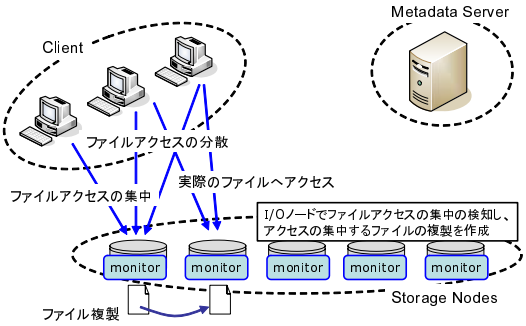

図 2 プロトタイプの概要

バから取得したノード情報を元に，それらのホストの負荷平均を取得する．

(5) 負荷平均の低いホストを 1 つ選択し，実際にファイルへアクセスする．

5. プロトタイプの設計と実装

5.1 プロトタイプの設計

3 章の提案手法のプロトタイプをグリッド用分散ファイルシステムである Gfarm 上にファイルアクセス分散機能を追加することで実現する．このプロトタイプの概要を図 2 に示す． Gfarm ファイルシステムは，クライアント，メタデータサーバ，I/O ノードから構成されるが，このうち，I/O ノードでファイルアクセスのモニタを行うことで，提案手法のプロトタイプを実現する．

I/O ノードのモニタ部分では, I/O ノードのアクセス状況のモニタを行うために, I/O ノードにおいてそのノードへのアクセス状況とファイルへのアクセス状況を記録する. このようにすることで，ノードとノード上のファイルのアクセス状況が把握でき，I/O ノード上で自発的にアクセス集中を検知することが可能になる．I/O ノードでアクセス集中を検知した後，そのノード上でアクセス頻度の高いファイルに対しファイルの複製をアクセスの集中していない I/O ノードへ作成するように指令する. このようにすることで，頻繁にアクセスがあるようなファイルが I/O ノードへ複製アクセス集中を回避できると考える．ファイル複製作成後，参照される回数が少なくなったファイル複製に関しては，ファイルシステム中にファイルの存在を保証した上で，そのファイル複製を消去する．

クライアントが実際にファイルへアクセスする場合は, 現在の Gfarm の実装 (Gfarm Version1.0.4) では，ファイルアクセスのスケジューリングに関してメタデータサーバで行わずに，各クライアントが各々アクセスするファイルを保持する I/O ノードの負荷平均をクエリし，実際のファイルアクセス先を決定する．プロトタイプにおいても同様に，メタデータサーバではなく，クライアントで実際のファイルアクセス先を決定することにした．

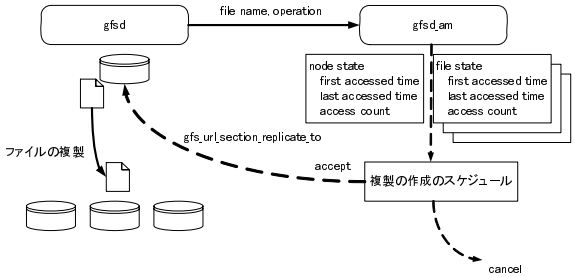

図 3 I/O ノードの構成

現在のプロトタイプでは，ファイルの一貫性に関して考慮されていない．しかし，データ・インテンシブ・コンピューティングにおいては，write once，read mostly なアプリケーションが多いため，現在のプロトタイプでは，このようなファイルアクセスを行うアプリケーションを対象にする．

5.2 プロトタイプの実装

I/O ノードの構成を表す図を図 3 に示す． Gfarm の I/O ノードでは，gfsd というデーモンが動作する．このデーモンは，ファイルシステムノード上の全ホストで動作してファイルアクセス機能を提供し，Gfarm ファイルシステムを構成する．プロトタイプでは，gfsd とは別に，I/O ノードのアクセス状況を監視する gfsd_am というデーモンを動作させる．gfsd は，その I/O ノード上のファイルアクセス要求がきた際に，その要求をキャッチして，そのファイル操作とファイルの名前を gfsd_am に通知する．それに対し，gfsd_am は，I/O ノードのアクセス状況を把握するために，その I/O ノードに最初にアクセスした時刻，最後にアクセスした時刻，およびアクセス回数を記録する．また，ノードの保持する各々のファイルのアクセス状況を把握するために，そのノードに最初にアクセスした時刻，そのファイルに最後にアクセスした時刻，また，アクセス回数を記録する．これらの情報より，以下のような式を用いてアクセス状況を算出する．ここで, アクセス状況を $access\_status$ とし, 最初にアクセスした時刻を T_{first_access}, 最後にアクセスした時刻を T_{last_access}, アクセス回数を $access\_count$ とする．

$$access\_status = \frac{access\_count}{T_{last\_access} - T_{first\_access}}$$

gfsd_am は，gfsd からファイルアクセスの通知を受ける毎に，ノードおよびファイルのアクセスに関して，最後にアクセスした時刻とアクセス回数の更新を行う．また，ある一定時間内にノード，あるいはファイルへのアクセスがなかった場合は，いままで記録していた

表 1　実験環境 (プロトタイプの評価)

	メタデータサーバ	クライアント, I/O ノード
CPU	Opteron 240	Opteron 242
Memory	2GBytes	2GBytes
OS	Linux 2.6.5 (32bit mode)	Linux 2.4.27 (32bit mode)
Network	1000Base-T	1000Base-T

時刻，アクセス回数などに関するデータをクリアする．

また，gfsd_am は，gfsd からファイルアクセスの通知を受ける毎に，ファイルの複製の作成を試み，上で定義した式を元にノードのアクセス状況を計算し，そのアクセス状況の値が閾値を超えるような場合に，そのノードへのアクセスが集中している判断し，アクセスが集中しているファイルを他のアクセスの集中していないノードへファイルの複製を作成する．アクセス状況の値が閾値を超えない場合は，また，gfsd_am は，ファイルアクセスのモニタを継続する．ファイルの複製は，Gfarm の提供するファイル複製作成のための API `gfarm_url_section_replicate_from_to` を呼ぶことで実現する．

上述の動作をまとめると次のようになる．

(1) 一定時間ノード，ファイルへのアクセスがない場合，モニタしたデータをクリアし，最初にアクセスした時刻 (T_{first_access}) を更新する．
(2) ノード，ファイルに関して最後にアクセスした時刻 (T_{last_access}) とアクセス回数 ($access\_count$) を記録する．
(3) アクセス状況 ($access\_status$) を算出する．
(4) ファイルの $access\_status$ が閾値を超えている場合，ノードの $access\_status$ が小さいノードへファイルを複製する．
(5) ファイルの $access\_status$ が閾値を超えていない場合，そのまま動作を継続する．

このような動作を繰り返すことで，各 I/O ノードのアクセス状況の均一化を実現する．

クライアントが実際にファイルへアクセスする場合は，アクセスするファイルを保持する I/O ノードの $access\_state$ をクエリし，その値の最も小さいノードへ優先的にファイルアクセスを行う．これは，アクセスの少ないノードへファイルアクセスを優先的に行うためである．

6. 実　　験

本研究の前提となっているアクセス集中の問題点の検証を行うための実験と，提案手法であるファイルアクセスの自動分散の有効性を確認するための実験を行った．実験環境は，松岡研究室の PRESTO III クラスタを用い，スペックは表 1 に示すとおりである．

また，クライアント，I/O ノードとメタデータサーバ間のスループットは，89.13MBytes/sec で，RTT は 0.1msec，また，クライアント，I/O ノード同士の間のスループットは，117.6MBytes/sec で，RTT は 0.1msec である．なお，スイッチ間のネットワークでの輻輳の発生による測定の混乱を避けるために，同じスイッチに接続されているクラスタノード (32 台) を使用した．

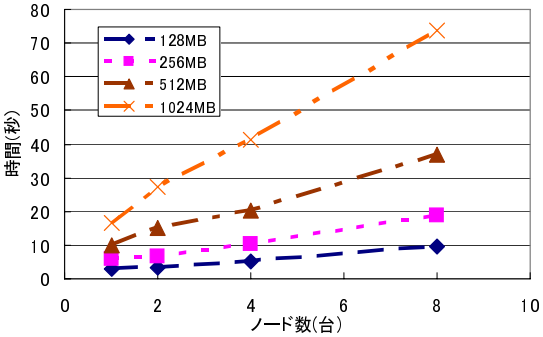

図 4　複数クライアントから単一ノード上の 1 つのファイルへの一斉アクセスの際の実行時間

6.1　アクセス集中の検証

まず，本研究で前提となっているアクセス集中に関する検証を行う．そのために，以下のような 2 つの単純なシナリオを想定した．

- 最悪なアクセスパターン
  ファイルシステム上の 1 つの I/O ノードに 1 つのファイルを置き，そのファイルへ複数のクライアントが一斉にアクセスする．
- 理想的なアクセスパターン
  ファイルシステム上の複数の I/O ノード上に同じファイルを置き，それらのファイルへ複数のクライアントがそれぞれ別の I/O ノード上のファイルへアクセスする

上記のシナリオについて Gfarm を用いて実験を行った．

6.1.1　最悪なアクセスパターンの検証

最悪なアクセスパターンのシナリオにおけるファイルアクセスの性能を検証するために，次のような実験を行った．Gfarm 上の 1 つ I/O ノード上に 1 つのファイルを置き，そのファイルに対して，8 台のクライアントが一斉に open, read, close を行った．このときの buffer サイズは 1MBytes とし，ファイルサイズを 128MBytes，256MBytes，512MBytes，1024MBytes と変更しながら実験を行った．この実験の結果を図 4 に示す．ファイルへアクセスするクライアントノード数が増加するにつれ，線形に実行時間が増加していることが確認できる．

このときの 1 つのクライアントがファイルアクセス処理に費やした時間の内訳を表 2 に示す．ただし，このときのファイルサイズは，1024MBytes とする．結果からファイルアクセス処理に費やされる時間の大部分は，read の処理に費やされており，open や close の

表 2　複数クライアントから単一ノード上の 1 つのファイルへの一斉アクセスの際の内訳 [秒])

	1node	2nodes	4nodes	8nodes
open	0.00738	0.00105	0.0160	0.0270
read	16.7	27.1	41.2	73.4
close	0.0304	0.0618	0.160	0.300

表 3　複数ノードから単一ノードへの一斉アクセスの際のスループット

ノード数 (台)	1	2	4	8
帯域幅 (MBytes/sec)	112.2	58.86	29.44	14.72

際のメタデータサーバへのクエリの集中がボトルネックになっていないことが伺える．

また，複数のクライアントが一斉に 1 つのノードへアクセスした際のネットワークスループットがどの程度かを見積もるために，netperf を用いて実験を行った．複数のクライアントが各々1 つのサーバに対して一斉に netperf を実行することで同等の状況を再現した．この実験の結果を結果を表 3 に示す．　表より，アクセスを行うクライアントの数を n 台とした場合は，クライアントと I/O ノード感のスループットを $MaxThput$MBytes/sec としたとき，$MaxThuput/n$ 程度しか出ていないことが伺える．

以上のことから，アクセス集中が発生した場合は，ネットワークの輻輳の影響のために，ファイルへのアクセス性能が著しく低下していると考えられる．

6.1.2　理想的なアクセスパターンの検証

理想的なアクセスパターンのシナリオにおけるファイルアクセスの性能を検証するために，次のような実験を行った．ある大きさのファイルを Gfarm 上に置き，そのファイルに対して，8 台のクライアントから一斉にアクセスを行った．このとき，クライアントからアクセスされるファイルの複製の数を変更しながら実験を行った．ファイルへは，リモートアクセス (すなわち，クライアントと I/O ノードが一致しないようなアクセス) になるようにした．また，ファイルの複製へのアクセスの方法は，I/O ノード間でアクセスされるクライアントの数に偏りが起こらないように制御した．つまり，クライアントが n 台で，ファイルの複製が m 個である場合，ファイルの複製の 1 つが，n/m 台からのクライアントからのファイルアクセスを受け持つ．この実験結果を図 5 に示す．　ファイルの複製の数が 1 の場合, 8 台のクライアントが一斉に 1 台の I/O ノードへアクセスするため, 6.1.1 節の場合と同じ状況になり, ネットワークの輻輳の要因で実行時間が増加していると考えられる．一方, ファイルの複製の数が 8 の場合, 各クライアントが各々別の I/O ノードへファイルアクセスを行うため，アクセスが分散され，実行時間の増加が抑えられている．

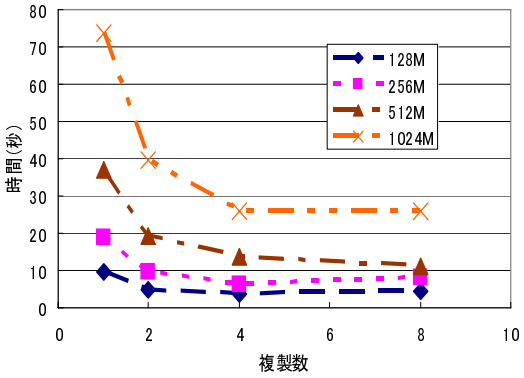

図 5　8 台のクライアントからのファイルアクセスの際の実行時間と複製数の関係

表 4　8 ノードからのファイルアクセスを行った際のネットワークスループット

ノード数 (台)	1	2	4	8
帯域幅 (MBytes/sec)	14.63	30.03	58.84	117.7

また，複数のクライアントが各々別の I/O ノードへアクセスした際のネットワークスループットがどの程度かを見積もるために，netperf を用いて実験を行った．複数のクライアントが各々1 つのサーバに対して一斉に netperf を実行することで同等の状況を再現した．

結果を図 5 に示す．8 台のアクセスのときも, ネットワークのスループットの理論値 (125MBytes/sec) 近くまで出ており, ネットワークの輻輳が発生していないことが確認できる．このことから, 理想的なアクセスパターンが行われた場合は, 1 台のクライアントが I/O ノードへリモートアクセスするのと同等の程度の時間でファイルアクセスが行えると考えられる．

6.2　提案手法の有効性の検証

提案手法の有効性を検証するために，Gfarm とファイルアクセスの自動分散機能を Gfarm 上に実装したプロトタイプとを用いて比較実験を行った．

6.2.1　ファイルアクセスの自動分散の検証

提案手法である他の I/O ノードへのファイルの複製の自動作成の効果を検証するために，3 節の提案手法を実装したプロトタイプと Gfarm を用いて次のような実験を行った．

ファイルシステム上に 1 つのファイルを置き，8 台のクライアントからリモートアクセスするようにファイルへの open, read, close の一連の動作を 5 回繰り返した．アクセスの対象となるファイルサイズは，ファイルの複製が自動的に作成される様子をわかりやすく示すために，128MBytes に固定した．このときの実行時間と性能比の結果を表 5 に示す．ただし，性能比は理想的な場合のファイルアクセスを基準とした．　表

表 5 ファイル複製の自動作成機能の比較

	理想的な場合	Gfarm	プロトタイプ
実行時間 (sec)	21.55	178.9	76.81
性能比	1.0	0.12	0.28

表 6 プロトタイプのファイルアクセスの実行時間の内訳
(縦軸はクライアントの番号, 横軸は実行回数, 単位は [秒])

	1 回目	2 回目	3 回目	4 回目	5回目
[0]	35.6	15.1	3.72	2.48	2.39
[1]	35.7	14.6	9.43	20.5	7.74
[2]	36.4	5.94	9.49	3.79	3.83
[3]	35.6	14.8	9.52	21.0	7.69
[4]	35.5	14.9	9.57	21.0	7.82
[5]	35.7	14.7	9.61	4.83	7.70
[6]	35.8	14.9	9.64	5.75	2.46
[7]	35.7	14.8	9.68	21.0	7.79

5 において，理想的な場合というのは，理想的な場合のファイルアクセスのことを指し，1 台のクライアントがリモートの 1 台の I/O ノード上のファイルに対して，open，read，close の操作を 5 回行った場合の実験を表す．

Gfarm では，8 台のクライアントが常に 1 台の I/O ノード上のファイルへのアクセスを行うため，ファイルを保持する I/O ノードが常にアクセス集中状態である．このため，ファイルアクセスの実行時間が大幅に増加し，理想的な場合のファイルアクセスと比較して性能比が著しく低くなっている．一方，プロトタイプでは，実行開始直後は 8 台のクライアントが 1 台の I/O ノード上のファイルへのアクセスを行い，I/O ノードでアクセス集中を検知した後は，他の I/O ノードへファイルの複製の作成を行うため，理想的な場合のファイルアクセスと比較しては性能比が著しく低くなっているが，Gfarm と比較した場合は，全体のファイルアクセスの実行時間が短縮されており，性能比としてはプロトタイプの方が 2.33 倍性能向上していることを確認した．また，この実験においては，最終的にファイル複製が 5 つ作成されることを確認した．

また，プロトタイプのファイルアクセスの実行時間の内訳を表 6 に表す．縦軸に示された番号が実験で用いたクライアントのノード ID を表し，横軸は，1 回が一連の open，read，close のファイルアクセス処理を表す．1 回目のアクセスでは，1 つの I/O ノードにアクセスが集中するため，実行時間が増加しているが，ファイルアクセスを繰り返すにしたがって，実行時間が 1 回目の実行時間と比較して低く推移しているのが確認できる．

7. 議　　論

7.1 アクセス集中の検証

6.1.1 節の最悪なアクセスパターンのシナリオでの実験より，単一の I/O ノードへファイルアクセスを故意に集中させた場合，ファイルアクセスのオーバーヘッドが非常に大きくなることを確認し，本研究の前提であるアプリケーションのアクセス集中回避の必要性を確認した．また，この際のオーバーヘッドの主たる要因は，ネットワークの輻輳が発生し，スループットが大幅に低下することを確認した．今回の実験では，ディスク I/O のオーバーヘッドは確認できなかったが，これは，この実験では，各クライアントが同じファイルへアクセスしたため，ディスクキャッシュが効いている可能性がある．例えば，複数のクライアントがある単一 I/O ノード上の複数のファイルへのアクセスが集中するシナリオを考えた場合，ディスクキャッシュが効かなくなり，ネットワーク I/O とディスク I/O の性能に依存したファイルへのアクセス性能の低下が発生すると考えられる．ファイルへのアクセスに関して，ネットワーク I/O とディスク I/O の性能はトレードオフの関係にあると考えられるため，この関係性についての検証も必要だと考える．

しかしながら，6.1.2 節の理想的なアクセスパターンのシナリオでの実験により，ファイルの複製が存在する場合，最も理想的な場合は，1 台のクライアントが 1 台の I/O ノード上のファイルへアクセスするのと同等の時間でファイルアクセスを行うことができることを確認した．

今回の実験では，1 つのスイッチに接続された計算機クラスタのノードを用いて実験を行ったため，高バイセクションバンド幅を実現できたため，最も理想的なファイルアクセス時間を実現できたが，実際のグリッド環境では，任意のノードへ高バイセクションバンド幅が実現できるとは限らない．例えば，次のようなシナリオを考える．異なる管理ドメインであるサイト A，サイト B が存在し，サイト A とサイト B 間が高速なネットワーク C で結ばれているとする．このような状況において，サイト B のある I/O ノード上のファイルに対して，サイト A の複数のクライアントが一斉にアクセスを行った場合は，6.1.2 節のように，仮にサイト B 上の別の I/O ノード上にファイルの複製を作成し，クライアントのアクセス先を理想的な状態に設定したとしても，ネットワーク C で輻輳が発生し，6.1.1 節と同じ状況が起きるため，ファイルアクセスの実行時間が増加すると考えられる．実際，表 1 の環境でスイッチを跨ぐように 6.1.2 節と同等の実験を行った場合，8 台のクライアントが 8 台の I/O ノード上のファイルに並列にアクセスしたとしても，ファイルサイズが 1024MBytes の場合，ファイルアクセスの実行時間が 76 秒程度かかることを確認している．この値は，図 4 より，6.1.1 節の最悪のファイルアクセスパターンのシナリオにおいてファイルサイズが 1024MBytes の場合に 8 台のクライアントが 1 台の I/O ノード上のファイルへアクセスするのにかか

る時間と同等となっている．このことからも，ネットワークの輻輳を避けることが重要であると考える．

7.2 提案手法の有効性の検証

6.2.1 節のファイルの複製の自動作成の実験において，提案手法のプロトタイプが，I/O ノードのアクセス集中を検知し，ファイルの複製が自動的に作成できることを示した．また，プロトタイプにより作成されたファイルの複製へのクライアントからのアクセスが分散され，ファイルアクセスの実行性能が向上することを確認し，提案手法の有効性を確認した．

今回の実験で用いたプロトタイプのファイル複製の作成は，アクセスの集中するホストからアクセスの集中していないホストへ行っていた．しかしながら，ファイルの複製が多数存在する場合は，アクセスの集中するホストからのファイルの複製作成を避け，アクセスの集中していないホストとの間でファイルの複製を作成する手法の方が，アクセスの集中するホスト上でのクライアントのファイルアクセス処理を妨げることなく実行ができ，また，ファイル複製も高速にできると考えられるため，更なる性能向上が見込めると予測できる．また，今回の実験で用いたプロトタイプのファイル複製の作成は，送信元と受信先とで一対一転送で行われていた．ファイルシステム全体であるファイルが頻繁にアクセスされる場合は，あらかじめファイルの複製を一対多転送で効率良く行うことにより，更なる性能向上が見込めると予測できる．このために，I/O ノード上での局所的なファイルアクセスのモニタリングだけでなく，メタデータサーバ上でファイルシステム全体のファイルアクセスをモニタする必要があると考えられる．

8. おわりに

8.1 まとめ

本稿では，ファイルシステム側でアクセスの集中を検知し，ファイル複製を積極的に利用して，ファイルへのアクセスを分散する手法を提案し，プロトタイプとしてこの提案手法の機能をグリッドファイルシステムである Gfarm 上に実装した．実験として，作為的にファイルへのアクセス集中が発生する状況を作り出し，プロトタイプによって自動的にアクセス集中を検知し，ファイル複製を作成することで，ファイルへのアクセスが分散され，ファイル複製を作成しない場合と比較して，2.33 倍のファイルアクセスの性能が向上することを確認した．

8.2 今後の課題

今後の課題としては, 次のような点が挙げられる。

- 適切なファイルアクセス先のスケジューリング
  現在は，各クライアントが各々ファイルアクセス先を決定しているため，ファイルアクセス先の割り当てに重複が発生し，アクセス集中が発生していた．このようなアクセス集中の発生を押さえるため，メタデータサーバでファイルアクセス先をスケジュールする必要があると考えている．
- ファイル複製作成のポリシーやファイル複製作成先の決定方法の検証
  本稿の実験では，I/O ノードでアクセス集中が発生する毎に，ファイルの複製を作成していたが，その複製をシステム上でどこに，どれだけ作成するとどの程度ファイルアクセスが効率良く実行できるようになるかは，明らかではないため，その検証が必要である．
- 実アプリケーション及び，実環境での評価
  本稿では, 非常に限定された条件下でのファイルアクセス集中の自動分散を検討した. より一般化された条件下のもとで, どのようなポリシーでファイルへのアクセスの分散を検討するために，実アプリケーションや実環境での評価が必要であると考えている．

参考文献

1) The Large Hadron Collider, `http://www.cern.ch/lhc/`.
2) Coda File System, `http://www.coda.cs.cmu.edu/`.
3) 武田伸悟, 伊達進, 下條真司：グリッドファイルシステム GSI-SFS, 情報処理学会研究報告 2004-OS-93, pp. 97 – 104 (2003).
4) Butler, R., Welch, V., Engert, D., Foster, I., Tuecke, S., Volmer, J. and Kesselman, C.: A National-Scale Authentication Infrastructure, *Computer*, Vol.33, No.12, pp. 60–66 (2000).
5) Maziéres,D.: *Self-certifying File System*, PhD thesis, Massachusetts Institute of Technology (2000).
6) Carns, P. H., Ligon III, W. B., Ross, R. B. and Thakur,R.: PVFS: A Parallel File System for Linux Clusters, in *Proceedings of the 4th Annual Linux Showcase and Conference*, pp. 317–327, Atlanta, GA (2000), USENIX Association.
7) Lustre, `http://www.lustre.org`.
8) Ghemawat, S., Gobioff, H. and Leung, S.-T.: The Google File System, in *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, pp. 96–108, Bolton Landing, NewYork (2003), ACM Press.
9) 建部修見, 森田洋平, 松岡聡, 関口智嗣, 曽田哲之：ペタバイトスケールデータインテンシブコンピューティングのための Grid Datafarm アーキテクチャ, 情報処理学会論文誌：ハイパフォーマンスコンピューティングシステム, Vol.43, No. SIG6 (HPS5), pp. 184–195 (2002).