データインテンシブコンピューティングのための グリッドファイルシステム上でのデータ管理

佐藤 仁 † 松岡 聡 †,††

グリッドに代表される大規模並列計算環境ではその不均質性のため，1) クライアントからある特定のノード及びある特定のファイルへのアクセスの際に時間的に近接した状態が発生し，ファイルアクセスが集中する，2) ファイルシステム上のファイルへのアクセスが空間的に遠方に存在するファイルへのアクセスとなる，などの要因によりファイルアクセス性能が低下することが問題となる．既存の分散ファイルシステムを用いて，このような性能低下を避けるためには，アプリケーション利用者や開発者が明示的にファイル複製を作成したり，ファイルアクセスを制御したりすることが必要となる．しかしながら，グリッド環境ではこのような対応は負担が大きく困難である．我々は，データインテンシブコンピューティングのためのグリッドファイルシステム上で動的なデータ管理機構を提案する．本手法は，グリッド上でファイルシステムを構成するノード群をネットワークトポロジに応じたグループに動的に分割し，それらのグループ分割を積極的に利用したファイルアクセスのスケジューリングやファイルシステム上のデータ管理を行うことで，ファイルアクセスの性能向上を試みる．提案手法をグリッド上のファイルシステムの 1 つである Gfarm を基盤として適用し，グリッド上でジョブスケジューラによりファイルシステム上のファイルへアクセスを行うジョブを複数投入して動作させた結果，最大で 3.7 倍の性能向上を確認した．

Data Management on the Grid File System for Data Intensive Computing

Hitoshi Sato † and Satoshi Matsuoka †,††

In parallel computing environments such as HPC clusters and the Grid, data-intensive applications involve large overhead costs due to a concentration of access to the files on common nodes. To avoid this problem in traditional distributed filesystems, users have to distribute the file access manually. However, such solution has some difficulties for users in the Grid environment. We propose a data management mechanism for data-intensive computing on the Grid filesystem. Our technique improves the file access performance by automatically scheduling the file access and the data management on the filesystem. The filesystem is based on dynamically configured node groups corresponding to the network topology. Utilizing the configuration, it monitors file access to detect concentrated situations, creates the file replica, and schedules its placement and access. We applied the proposal technique to the Gfarm, a filesystem that scales to the Grid. We emulate real application workloads using a job scheduler and confirmed a speedup of factor 3.7 compared with a filesystem without automatic file access distribution techniques.

1. はじめに

近年，大規模なデータ処理を必要とするデータインテンシブアプリケーションの実行環境としてグリッドの利用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において，実験及び観測から得られた大規模なデータを複数の異なる資源管理組織間で共有し，このような環境を利用したデータの分析や解析を行うなどの試みが積極的に行われている．

グリッド上でデータ共有を行うには，シングルシステムイメージを提供するためのファイルシステムをベースにするのが望ましい．これは，既存のアプリケーションを修正することなしに利用でき複数のアプリケーション間の連携がファイルベースで行えること，多くのアプリケーション利用者や開発者が慣れ親しんでいること，などの利点があるためである．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，

† 東京工業大学
Tokyo Institute of Technology
†† 国立情報学研究所
National Institute of Informatics

グリッドが大規模で不均質であるため，1) クライアントからある特定のノード及びある特定のファイルへのアクセスの際に時間的に近接した状態が発生しファイルアクセスが集中する，2) ファイルシステム上のファイルへのアクセスが空間的に遠方に存在するファイルへのアクセスとなる，などの要因によりファイルアクセス性能が低下するためである．既存の分散ファイルシステムを用いて，このようなファイルアクセスの性能低下を避けるためには，アプリケーション利用者や開発者が明示的にファイル複製を作成したり，ファイルアクセスを制御したりすることが必要となる．しかしながら，グリッドではこのような対応は負担が大きく困難である．

我々は，データインテンシブコンピューティングのためのグリッドファイルシステム上で動的なデータ管理機構を提案する．本手法は，グリッドを構成するノード群をネットワークトポロジに応じたグループに動的に分割し，それらのグループ分割を積極的に利用したファイルアクセスのスケジューリングやファイルシステム上のデータ管理を行うことで，ファイルアクセスの性能向上を試みる．提案手法をグリッド上のファイルシステムの１つである Gfarm を基盤として適用し，グリッド上でジョブスケジューラによりファイルシステム上のファイルへアクセスを行うジョブを複数投入して動作させた結果，最大で 3.7 倍の性能向上を確認した．

本稿は以下のように構成される．2 章でグリッド上のファイルシステムとその問題点について述べる．3 章で提案手法であるグリッドファイルシステム上でのデータ管理について述べ，4 章でそのプロトタイプの実装について述べる．5 章で提案手法の評価実験について述べる．6 章で関連研究との比較を行い，最後に 7 章でまとめと今後の課題について述べる．

2. グリッド上のファイルシステム

既存のグリッドの多くは HPC クラスタを複数統合して構成される．また，グリッドを構成する個々の HPC クラスタの多くは，NFS，AFS や Coda などのネットワークファイルシステムを用いて構成され，シングルシステムイメージを提供することでユーザに対して高い利便性を実現する．グリッドでもこのようなシングルシステムイメージを提供するためのファイルシステムが存在すると，ユーザに対して高い利便性を実現でき，更に，グリッドの持つ複雑さや不均質さを隠蔽できる．このような，グリッド上のファイルシステムとして実現すべき要件として主に，1) 異なる資源管理組織間での安全なファイル共有を実現する安全性，2) 科学技術計算で求められる大規模高性能計算のサポートを実現するスケーラビリティ，が挙げられる．本章では，これらの要件が既存のネットワークファイルシステムでどのように実現されるかを述べ，問題点について触れる．

2.1 安全性の実現を目的としたファイルシステム

安全性を実現するファイルシステムとして，NFS を基盤としてセキュリティ機構を組み込んだファイルシステムが挙げられる[1)2)]．これらのファイルシステムでは，広域環境上でも高い安全性や NFS と同等のユーザー利便性を提供することが可能である．しかしながら，基本的には 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る．

2.2 スケーラビリティの実現を目的としたファイルシステム

2.2.1 並列ストライピングファイルシステム

スケーラビリティを実現するファイルシステムとして，クラスタ型計算機での利用を目的とした並列ストライピングファイルシステム[3)4)5)] が挙げられる．これらのファイルシステムでは，ファイルをいくつか断片に分割して異なるディスクに分散格納し，ファイルアクセス時に複数のディスクの利用を可能にする．このため，高バンド幅なディスク I/O やファイルアクセスの分散化，均一化が実現できる．しかしながら，このようなファイルシステムは，セキュリティ機構が未サポートであること，ファイルアクセスの I/O バンド幅がネットワークバンド幅に制限されること，などの要因から望ましくない．

2.2.2 Grid Datafarm アーキテクチャ

グリッド上での利用も目的としたファイルシステムとして，Grid Datafarm アーキテクチャ[6)] が挙げられる．これは，ファイルをアプリケーションが操作可能なファイル断片に分割して，異なるノード上のディスクに分散配置し，並列 I/O API によりシングルシステムイメージでそれらのファイル断片にアクセスする手段を提供する．また，スケジューラにより，ファイル断片が格納されているノードに対しプロセスを割り当て，ファイルアクセスをローカルなディスク I/O とすることで (ファイルアフィニティスケジューリング)，データインテンシブコンピューティングで求められるスケーラビリティを実現する．また，Grid Datafarm アーキテクチャの参照実装である Gfarm[7)] ではセキュリティ機構がサポートされており，異なる資源管理組織間においても安全なデータ共有が可能である．しかしながら，ファイルシステム上のデータ管理はユーザが手動で行わなければならず，大規模で不均質なグリッドを考慮した場合に問題となる．

2.3 グリッド上のファイルシステムの問題点

一般的にグリッド上で行わ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て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．2.2.1 節の手法では，ファイル読み出し時に複数のディスクを利用してファイル断片を読み出して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は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，2.2.2 節の手法では，ファイルの存在するノードに対してプロセスを割り当て，ディスク I/O を積極的に利用することで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．更に，2.2.2 節の手法を想定した場合においても，グリッド上のファイルシステムでは，1) クライアントからある特定のノード及びある特定のファイルへのアクセスの際に時間的に近接した状態が発生しファイルアクセスが集中する，2) ファイルシステム上のファイルへのアクセスが空間的に遠方に存在するファイルへのアクセスとなる，などの要因によりファイルアクセス性能が低下することが問題となる．このため，これらの問題を回避するために，グリッド上のファイルシステムでの動的なデータ管理が必要である．

3. グリッドファイルシステム上でのデータ管理

2.3 節で述べた問題点を解決するために，write once, read mostly なワークロードを持つデータインテンシブコンピューティングを対象にしたグリッドファイルシステム上の動的データ管理機構を提案する．本手法は，2.2.2 節の Grid Datafarm アーキテクチャを基盤にし，以下の 2 点を実現する．

(1) ファイルアクセスの集中を回避するために，ファイルシステム上でファイルアクセスのモニタリングを行うことで参照数の多いファイルを特定し，それらのファイルに対して複製を作成する．

(2) 空間的に遠方に存在するファイルへのアクセスを回避するために，クライアントからのファイルアクセスの際にファイルへのアフィニティが高くなるような複製の配置を動的に行い， それらへのアクセスを制御する．

以降，3.1 節で提案手法が対象とするファイルシステムの構成について触れ，3.2 節でファイルシステム上のデータ管理を行う上で基盤となるファイルシステムを構成するノードの管理手法について触れる．その後，3.3 節でクライアントからのファイルアクセスの動作について触れ，3.4 節でファイルシステム上のデータ管理手法について述べる．

3.1 ファイルシステムの構成

本手法が対象とするファイルシステムの構成を図 1 に示す．ファイルシステムへのアクセスを提供するクライアント，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，また，実際にファイルの断片 (以降セクションと表す) を保持する I/O ノードから構成される．ファイルシステム上のファイルへアクセスするために，まず，クライアントがメタデータサーバへファイルの所在に関するクエリを行う．メタデータサーバはクライアントが実際にファイルアクセスを行う I/O ノードを決定しクライアントへ通知する．その後，クライアントはそれらの情報を基に実際にファイルが保存されている I/O ノードへアクセスを行う．また，メタデータサーバにおいてクライアントからのファイルシステム上のファイルへのアクセスに関してモニタリングを行い (ファイルシステムモニタ)，I/O ノード上で CPU の負荷やストレージの利用状況などのリソースに関してモニタリングを行う (I/O ノードモニタ)．ファイルシステムモニタを用いることで，ファイルシステム上のファイルの参照状況を検知し，参照数の多い場合は複製を作成し，参照数の少ない場合は複製を削除する．一方，I/O ノードモニタを用いることで，メタデータサーバがクライアントのファイルアクセス先のスケジューリングの際の情報として用いる．

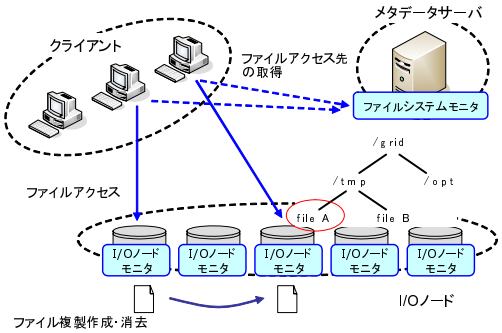

図 1 ファイルシステムの構成

3.2 ファイルシステムを構成するノードの管理

グリッド上でファイルシステムを構成するノードをネットワークの構成に応じていくつかのグループに分割する．ノードをグループへ分割することにより，グリッドの持つ不均質性を隠蔽し，これらのノード分割に基づいたファイルアクセスのスケジューリングやファイルシステム上のデータ管理を行うことでファイルアクセスの性能向上を試みる．

ノードのグループへの分割は次のように行う．まず，ノードのグループに対してサイズ s を以下のように定義する．

$$s(V) = \sum_{e \in E(V)} \left(\frac{1}{C_{bw}(e)} + C_{rtt}(e) \right) \qquad (1)$$

ここで，V はグループを構成するノードの集合，E はグループを構成するノード間のネットワークのリンク，C_{bw} はネットワークのバンド幅，C_{rtt} はネットワークの遅延とする．このグループのサイズ s を用いて，2 つのノードのグループ V_1，V_2 の距離 d を以下のように定義する．

$$d(V_1, V_2) = s(V_1 \cup V_2) \quad (2)$$

(2) 式で定義した距離を用い，以下のステップを繰り返すことでノードをグループへ分割する．

Step 1. ファイルシステムを構成する各ノードを各々 1 つのノードから構成されるグループとする．

Step 2. (2) 式よりグループ間の距離を計算する．

Step 3. グループ間の距離が最小になるような 2 つのグループを選択し，それらのグループを 1 つのグループへと構成する．

Step 4. グループ間の距離の最小値が閾値を超えるまで，Step 2. と Step 3. を繰り返す．

3.3 クライアントからのファイルアクセスの制御

3.2 節で構築したノードのグループに基づき，ファイルアクセス先のスケジューリングを行う．これにより各クライアントからのファイルアクセスを局所化し，2.3 節の問題点の解決を試みる．

- ファイルがファイルアクセスを要求するノードのローカルストレージに存在する場合，その要求元のノードを選択する．
- ファイルがファイルアクセスを要求するノードのローカルストレージに存在しない場合，
  - ファイルアクセスを要求するノードが属するノードと同じグループに属するノード上にアクセス対象とするファイルが存在するとき，そのノードを選択する．
  - ファイルアクセスを要求するノードが属するノードと同じグループに属するノード上にアクセス対象とするファイルが存在しないとき，アクセス対象とするファイルを保持する任意のノードを選択する．

3.4 ファイルシステム上のデータ管理

3.4.1 ファイルアクセス集中の検知

ファイルアクセス集中の検知を行うために，アクセス対象となるファイルのモニタリングを行う．これはファイル，セクション毎に行い，それらのアクセス先をスケジュールする際に同時に行う．クエリの対象となったファイル，セクション，ファイルアクセスのクエリを行ったクライアントや実際にスケジュールされた I/O ノード，時刻を記録する．これらの情報を基に，次のようにファイルアクセスの集中を検知する．まず，ファイル，セクションに対してあらかじめモニタリングを行う時間 (モニタリング時間，$monitoring\_time$) を設定する．次に，ファイル，セクションに最後にアクセスされた時刻から，モニタリング時間内に同じファイル，セクションへのアクセス数をカウント ($count$) し，そのアクセス数をモニタリング時間で割ったものをアクセス状況を表す値 ($state$) とし，アクセス状況があらかじめ設定した閾値を超えた場合をアクセス集中状態と判定する．以下に式を示す．

$$state = \frac{count}{monitoring\_time} \quad (3)$$

3.4.2 ファイル複製の作成

3.4.1 節でアクセス集中状態が判定された後，そのときモニタリングしていたファイルとセクションに関して複製の作成を行う．作成先は次のように決定する．

- アクセス集中が判定された際，そのときのファイルアクセスのスケジュールでファイルアクセスを要求するノードとスケジュールされたファイルアクセス先のノードの属するグループが同じ場合，その同じグループに属するノード間でファイル，セクションを複製する．
- アクセス集中が判定された際，そのときのファイルアクセスのスケジュールでファイルアクセスを要求するノードとスケジュールされたファイルアクセス先のノードの属するグループが異なる場合，ファイルアクセスを要求するノードの属するグループとそれとは異なるグループに属するノードの間でファイル，セクションを複製する．

すなわち，3.2 節で構成したノードのグループにおいて，グループ内にあるファイル，セクションへの参照が多い場合は，そのグループ内で積極的に複製を作成し，グループ内にないファイル，セクションへの参照が多い場合は，他のグループからノードの近傍へ複製を作成する．

3.4.3 ファイル複製の消去

3.4.1 節のファイルアクセス集中の検知において，そのときに対象としていたファイル，セクションがアクセス集中状態と判定されなかった場合，そのときのアクセス状況の値に応じて，ファイル，セクションの存在を保証した上でファイル複製の削除を行う．この操作により，複製が頻繁に作成されストレージ上に恒久的に存在する状態を回避する．

4. プロトタイプの実装

3 章で述べた提案手法のプロトタイプを Grid Datafarm アーキテクチャの参照実装である Gfarm に機能を追加することで実現する．現在のプロトタイプは，Gfarm version 1.2 を基盤に利用している．

4.1 構　　成

図 2 にプロトタイプの構成を示す．クライアント，メタデータサーバ (gfmd，LDAP)，I/O ノード (gfsd)，および，ファイルシステムモニタ (gfads)，I/O ノードモニタ (gfsd_am) から構成される．メタデータサーバにおいて，gfmd はユーザがリクエストした並列プロセスの管理に用いられ，ファイルシステムのメタデータは LDAP により管理される．ファイルシステムモニタとしてメタデータサーバが動作するノード上で gfads が動作するが，これはファイルシステム上のファイルアクセスのモニタリングだけでなく，クライアントのファイルアクセス先の決定を行うスケジューラやファイルシステム上のデータ管理を行うためのイ

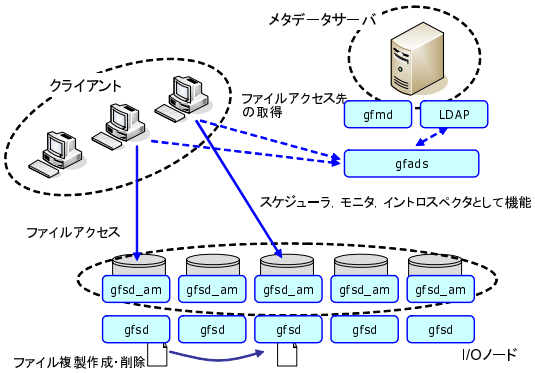

図 2 プロトタイプの構成

ントロスペクタ として動作する．また，I/O ノードでは，gfsd_am と呼ばれる I/O ノードモニタが動作し，CPU の負荷やストレージ容量などのリソースに関してモニタリングを行う．

4.2 クライアントからのファイルアクセス動作

ファイルシステム上のファイルへのアクセスはクライアント上で Gfarm の提供する Gfarm Parallel I/O API を呼び出すことで行う．クライアントはアクセス先の I/O ノードを決定するために，gfads に対して Gfarm 上のファイルとそのファイルに付随する情報であるセクションを通知する．セクションは，ファイルが実行形式の場合は I/O ノードのアーキテクチャ名を表し，通常のファイルの場合は，ファイル断片のインデックス番号を表す．クライアントからの通知を受け，gfads のスケジューリング機能により，3.3 節で述べた手法をもとにアクセス先の I/O ノードが決定される．現在の実装では，同一グループにファイル複製を持つノードが複数存在する場合は，それらの I/O ノードに対して負荷平均のクエリを行い，その最も低い I/O ノードをクライアントのアクセス先として決定する．アクセス先の I/O ノードの決定後，クライアントはスケジュールされた I/O ノードに対しファイル操作の要求を行う．

4.3 ファイルアクセスのモニタリング

gfads のモニタリング機能により，クライアントから I/O ノードへのファイルアクセスに関するモニタリングを行う．gfads はファイルアクセスのスケジューリング時に同時に，クエリの対象となったファイル，セクション，ファイルアクセスのクエリが行われたクライアントや実際にスケジュールされた I/O ノード，ファイル操作の種類，時刻を Postgresql DB 上へ記録する．これらの情報はファイルシステム上のデータ管理のために用いられ，一定時間の経過の後にローテーションにより消去される．

4.4 ファイルシステム上のデータ管理

gfads は，4.3 節で述べたのファイルアクセスのモニタリングの後に，gfads のイントロスペクタ機能によりファイルシステム上のデータ管理を行う．まず，ファイル，セクションに対して，3.4.1 節の手法を基にファイルアクセスの集中を検知する．現在の実装ではモニタリング時間は 180(秒) と設定している．その後，アクセス集中状態と判定されたファイル，セクションに対して，ファイル複製の作成を行う．ファイル複製の作成元と作成先は，3.4.2 節の手法により決定され，実際のファイル複製の作成は，Gfarm の提供するファイル複製作成のための API(gfarm_url_section_replicate_to および gfarm_url_section_replicate_from_to) を呼び出すことで行う．グループ内に複数の複製が存在する場合の複製元の選択は，ファイルアクセスのスケジューリング時と同様に，I/O ノードの平均負荷を元に行う．ファイル，セクションがアクセス集中状態と判定されなかった場合は，3.4.3 節の手法を基にそのファイル，セクションの存在を保証した上で，ファイル複製の消去を行う．

4.5 ファイルシステムを構成するノードの管理

ファイルシステムを構成するノードの管理は次のように行う．まず，4.4 節のファイル複製の作成時に同時に，そのファイル，セクションの転送時間を計測する．その後，ファイル，セクションのファイルサイズをメタデータサーバより取得してバンド幅を算出し，ファイル複製の作成元，作成先の情報とともに記録する．この動作をファイル複製の作成を行う毎に行う．その後，3.2 節の手法により，ファイルシステムを構成するノードをグループへ分割する．現在の実装では，距離の最小値の閾値を 0.01 と設定し，また，遅延の影響を考慮していない．

5. 実　　験

5.1 概　　要

提案手法の有効性を検証するために，ファイルアクセス集中の検知と広域環境でのファイルシステム上のデータ管理の 2 点に関して実験を行った．テストベットとして，東京都目黒区にある東京工業大学 PRESTOIII クラスタと茨城県つくば市にある産業技術総合研究所 KOUME クラスタ) をプロトタイプファイルシステムを用いて構成した．この際，ファイルシステムのメタデータサーバを PRESTOIII クラスタ 1 ノードに割り当て，その他の PRESTOIII クラスタと KOUME クラスタのノードをファイルシステムのクライアント及び I/O ノードとして構成した．図 3 にテストベッドの構成，表 1，表 2 に各計算機クラスタの性能とネットワークの性能を記す．また，PRESTOIII クラスタ上ではジョブスケジューリングシステムである Condor[8] を動作させ，PRESTOIII クラスタの 1 ノードにジョブスケジューラとして，その他の PRESTOIII クラスタのノードをジョブ投入ノード及びジョブ実行ノード

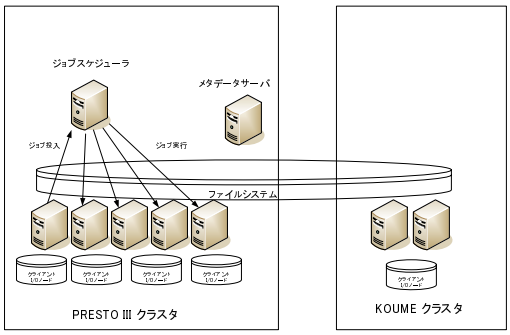

図 3 テストベッドの構成

表 1 各計算機クラスタの性能

	PRESTO III	KOUME
ノード数	60 (120CPU)	5 (10CPU)
CPU	Opteron 242 ×2	Pentium III 1400MHz ×2
Memory	2GBytes	2GBytes
OS	Linux 2.4.27	Linux 2.4.20
Network	1000Base-T	100 Base-T

表 2 ネットワーク性能

	PRESTO III ノード間	KOUME ノード間	PRESTO III - KOUME
RTT [ms]	0.083	0.055	6.60
バンド幅 [Mbits/sec]	973	908	73.0

として配備した.

5.2 ファイルアクセスの集中の検知

3.4 節で述べたアクセス集中の検知のためのパラメタのファイルシステム上のデータ管理への影響を調べるために，PRESTOIII クラスタ上で 1 つのスイッチに接続されたノード (19 台) を用いて次のような実験を行った．1 つのノードからジョブスケジューラに対し全てのファイルの内容の read を行う (open，read，close) ジョブを 100 回連続投入する．ファイルは 1 つのスイッチに接続された PRESTOIII クラスタのノード上に配備し，ファイルシステム経由で各ジョブ実行ノードからアクセスされる．この実験では，全てのノードを同一グループとして扱い，ファイルシステムを以下の 5 種類の状態に設定したものを行った.

(1) アクセス集中検知及びファイルシステム上のデータ管理を行わない (no_rep)
(2) アクセス集中検知のパラメタを 0.5 と設定し，ファイルシステム上のデータ管理を行う (p0.5)
(3) アクセス集中検知のパラメタを 0.25 と設定し，ファイルシステム上のデータ管理を行う (p0.25)
(4) アクセス集中検知のパラメタを 0.1 と設定し，ファイルシステム上のデータ管理を行う (p0.1)
(5) あらかじめファイル複製を各ノードのローカルディスクに作成し，ファイルシステム上のデータ管理を行わない (oracle)

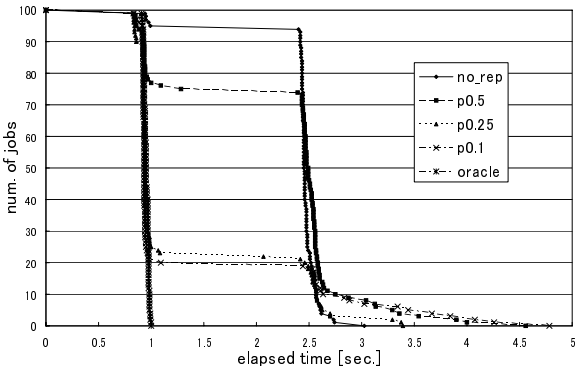

図 4 ファイルアクセスの集中の検知の実験結果

アクセス対象のデータとして 128MBytes のファイルを用いた.

図 4 に実験結果を示す．図中の x 軸は 1 つのジョブの実行時間 (秒) を示し，y 軸は累積ジョブ数を示す．理想的なシナリオの場合 (oracle)，全てのジョブの実行時間が 0.8〜1.0 秒程度要するのに対し，最悪のシナリオの場合 (no_rep)，95%のジョブの実行時間が 24〜3.1 秒の範囲に収まり，残りの 5%が 0.8〜1.0 秒の範囲に収まった．この際のジョブの最大実行時間は 3.02 秒だった．最悪のシナリオの場合で 5%のジョブの実行時間が 0.8〜1.0 秒で終了している理由は，ジョブがファイルを保持するノードに投入され，ローカルディスクに存在するファイルへアクセスするためである．一方，理想的なシナリオの場合は全てのジョブがローカルディスクに存在するファイルへアクセスする.

アクセス集中検知に関する結果は図中の p0.5，p0.25，p0.1 により示される．22%(p0.5)，75%(p0.25)，79%(p0.1) のジョブの実行時間が 0.8〜1.0 秒の範囲に収まることを確認した．これらの結果からパラメタを下げるに従い頻繁にファイルアクセス集中状態と判定されファイル複製が作成される効果が伺える．しかし，9%(p0.5)，3%(p0.25)，7%(p0.1) のジョブの実行時間が，最悪なシナリオでのジョブの最大実行時間 (3.02 秒) よりも大きい値を示した．この際の各ジョブの最大実行時間はそれぞれ 4.56 秒 (p0.5)，3.39 秒 (p0.25)，4.79(p0.1) 秒であった．これはファイルアクセス集中の判定及びファイル複製の作成が適切に行われていないことに加え，ファイルシステム上のデータ管理の処理のオーバーヘッドにより，いくつかのジョブではファイルアクセス先のスケジューリングが適切に行われていないことが要因である.

5.3 広域環境でのファイルシステム上のデータ管理

広域環境においてファイルシステム上のデータ管理のジョブへの影響を調べるために，PRESTOIII クラスタ (60 ノード) と KOUME クラスタ (5 ノード) を用

いて次のような実験を行った．PRESTOIII クラスタ上の 1 つのノードからジョブスケジューラに対しファイルの open，read，close のみを行うジョブを 100 回連続投入する．アクセス対象のデータは KOUME クラスタ上のノード上に配備し，ファイルシステム経由で各ジョブ実行ノードからアクセスされる．ファイルシステムを以下の 5 種類の状態に設定して実験を行った．

(1) アクセス集中検知とその後のファイルシステム上のデータ管理を行わない (no_rep)
(2) 全てのノードを同じグループに設定し，ファイルシステム上のデータ管理を行う (same_group)
(3) ノードのグループを提案手法により動的に構成し，ファイルシステム上のデータ管理を行う (auto_config)
(4) ノードをいくつかのグループに分け，ファイルシステム上のデータ管理を行う (diff_group)
(5) 予めファイル複製を各ノードのローカルディスクに作成し，ファイルシステム上のデータ管理を行わない (oracle)

(4) において，ノードのグループ分割は，KOUME クラスタ上の全てのノードを 1 つのグループ，1 つの同じスイッチに接続された PRESTOIII クラスタ上のノードを 4 グループ (7 台，17 台，16 台，20 台) として構成した．また，(2)，(3)，(4) の実験で用いたアクセス集中検知のパラメタは 0.1 とし，アクセス対象のデータは 128MBytes のファイルを用いた．

図 5 に実行結果を示す．図中の x 軸は 1 つのジョブの実行時間 (秒) を示し，y 軸は累積ジョブ数を示す．理想的なシナリオの場合 (oracle) 全てのジョブの実行時間が 0.8〜1.0 秒の範囲に収まるのに対し，最悪のシナリオの場合 (no_rep) 全てのジョブの実行時間が 29.8 秒以上を要した．これは，理想的なシナリオの場合，全てのジョブがローカルディスクに存在するファイルへアクセスするのに対し，最悪のシナリオの場合では全てのジョブが KOUME クラスタのノード上に存在するファイルへアクセスするためである．

ノードのグループ分割の効果は，図中の same_group，diff_ group，auto_group により示される．26%(same _group)，28%(diff_group) のジョブの実行時間が 0.8〜1.0 秒の範囲に収まり，また，10%(same_group)，11%(diff_group) のジョブの実行時間が 29.8 秒以上を要し有意な差がみられなかった．しかしながら，34%(same_group)，52%(diff_group) のジョブの実行時間が 1.0〜4.0 秒の範囲に収まることから，ノードのグループ分割により空間的に遠方に存在するファイルへのアクセスが抑えられることが確認できる．

次に，ファイルシステムを構成するノードを動的に構成した場合の結果を auto_config に示す．37% のジョブの実行時間が 0.8〜1.0 秒の範囲に収まり，same_group，diff_group と比較して良好な結果を示し，ノードのグループの動的な構成及びそれによるファイルアクセスの局所化が有効に動作していることが伺える．一方で，18%のジョブの実行時間が 29.8 秒以上を要した．これは，ファイルシステム上のデータ管理の処理のオーバーヘッド及びファイルシステムを構成するノード管理のオーバーヘッドにより，いくつかのジョブではファイルアクセス先のスケジューリングが適切に行われていないためである．

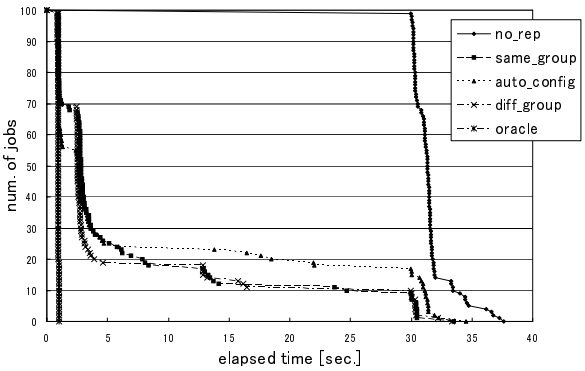

図 5 広域環境でのファイルシステム上のデータ管理の実験結果

表 3 各シナリオでのジョブの平均時間及び最大時間の比較 [sec.]

	no_rep	same_ config	auto_ config	diff_ config	oracle
average	31.6	6.50	8.50	6.19	0.954
max	37.6	33.5	34.5	33.3	0.997

最後に，(1)〜(5) の各シナリオでのジョブの平均時間及び最大時間を比較した．結果を表 3 に示す．平均時間の比較において，最終的に本手法は最悪なシナリオに対して最大 3.7 倍の性能向上を示した．一方で，理想的なシナリオに対しては 8.9 倍の性能低下を示した．これは，今回の実験が遠隔サイトへのデータアクセスの性能がネットワーク性能により著しく低下するような設定であったためジョブの最大時間により平均時間が増大するためである．このため，ファイルアクセスの際のアフィニティを実現するために，ファイルアクセスのワークロードのモニタリングによるアクセス集中検知アルゴリズムの改善やファイル複製作成の効率化が必要である．また，動的なノード構成手法 (auto_config) では，静的なノード構成手法と比較して (same_config，diff_config) 平均時間，最大時間においてオーバーヘッドが見られた．しかし，静的なノード構成手法の実験により (same_config，diff_config) 正確なノード構成の把握自体には効果が見られた．

6. 関 連 研 究

いくつかの分散ファイルシステムはデータ管理機構を備える．Batch-Aware Distributed File System[9] は，ストレージの制御機構を外部スケジューラから扱

えるようにしたファイルシステムである．ファイルシステムの内部制御を外部にさらすことで，スケジューラと協調動作し，I/O インテンシブなジョブ実行をサポートする環境を実現する．しかし，ユーザによるワークロードの明示的な記述が必要であり，ジョブの I/O に関する静的な知識を要求する．Google File System[5] は，ストレージへのファイル断片の格納の際に複製を作成する．しかし，主として耐故障性の実現を目的としており，広域環境での HPC を目的とした本手法とは異なる．また，複製の動的な管理機構は備えてはいない．

Content Derivary Network(CDN) では，read mostly なデータのインターネット上への配布を行う．これらは主にコンテンツ配信を対象にし遅延及びバンド幅に基づくオーバーレイネットワークをインターネット上に構築し 1 対 N のデータ転送をサポートする[10][11][12]．一方，データインテンシブコンピューティングにおけるファイルシステムのワークロードは，N 対 M のデータ転送などのより複雑なワークロードを対象にする．

ファイルシステムを構成するノードの管理に関してはネットワークトポロジの動的な検知に関する研究が関連する[13]．これらの研究の多くは遅延を考慮して構成される．本手法では，ファイル複製を頻繁に行うという特性を活かし，ファイルシステム上のデータ管理に十分な程度の精緻なネットワークトポロジの構成を目指す．

7. おわりに

グリッド上のファイルシステムにおいて，1) クライアントからある特定のノード及びある特定のファイルへのアクセスの際に時間的に近接した状態が発生する，2) ファイルシステム上のファイルへのアクセスが空間的に 遠方に存在するファイルへのアクセスとなる，などの要因によるファイルアクセス性能が低下を回避するため，データインテンシブコンピューティングのためのグリッドファイルシステム上で動的なデータ管理機構を提案した．また， 提案手法をグリッド上のファイルシステムの 1 つである Gfarm を基盤として適用し，グリッド上でジョブスケジューラによりファイルシステム上のファイルへアクセスを行うジョブを複数投入して動作させた結果，最大で 3.7 倍の性能向上を確認した．今後の課題としては，ファイルアクセス集中検知のアルゴリズムの精緻化，様々なデータインテンシブアプリケーションによる評価，より広範囲で大規模なグリッド環境での評価が挙げられる．

参 考 文 献

1) Maziéres,D.: *Self-certifying File System*, PhD thesis, Massachusetts Institute of Technology (2000).
2) 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).
3) 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.
4) Lustre, `http://www.lustre.org`.
5) 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.
6) 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).
7) Gfarm, `http://datafarm.apgrid.org/`.
8) Condor Project Homepage, `http://www.cs.wisc.edu/condor`.
9) Bent, J., Thain, D., Arpaci-Dusseau, A., Arpaci-Dusseau, R. and Livny, M.: Explicit Control in a Batch Aware Distributed File System, in *Proceedings of the First USENIX/ACM Conference on Networked Systems Design and Implementation (NSDI)*, San Francisco, CA (2004).
10) Jannotti, J., Gifford, D. K., Johnson, K. L., Kaashoek, M. F. and James W. O'Toole, J.: Overcast: Reliable multicasting with an overlay network, in *Proceedings of the 4th Symposium on Operating System Design and Implementation* (2000).
11) Bozdog,A., Renesse,van R. and Dumitriu,D.: SelectCast: a scalable and self-repairing multicast overlay routing facility, in *Proceedings of the ACM Workshop on Survivable and Self-Regenerative Systems* (2003).
12) Banerjee, S., Bhattacharjee, B. and Kommareddy,C.: Scalable Application Layer Multicast, in *Proceedings of the ACM Sigcomm* (2002).
13) Coates,M., Castro,R., Nowak,R., Gadhiok,M., King,R. and Tsang,Y.: Maximum Likelihood network topology identification from edge-based unicast measurements, in *Proceedings of ACM Sigmetrics* (2002).