社団法人　情報処理学会　研究報告
IPSJ SIG Technical Report

2005－ARC－162 （2）
2005－HPC－101 （2）
2005／3／7

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が, 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, ファイルへアクセスを分散する手法を提案する. 本稿では, グリッドファイルシステムである Gfarm を用いてファイルへのアクセスが集中する例, 理想的なアクセスパターンの例について検証を行った. また, 提案手法を Gfarm に実装したプロトタイプを用いて, ファイルへのアクセスの自動分散の有効性を確認した.

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 claim an automatic file distribution scheme using the access concentration detection on the file system and the file replication. We implement this prototype on Gfarm and evaluate its performance. The results showed that our prototype is better performance than Gfarm in the file concentration situation.

1. はじめに

近年, HPC クラスタ技術やグリッド技術等の発達により, 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, 地震工学などような様々な科学技術計算の分野において, 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行われている. 例えば, 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, 処理する必要があるとされる. このような環境を実現するために, 基盤となる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は, 統一的なファイルビューを提供するためのファイルシステムを利用することが有効であると考えられるが, HPC クラスタやグリッドなどの並列計算環境において既存の分散ファイルシステムを用いた場合は, アプリケーションによっては, アクセス集中のためにアプリケーションのファイルアクセスの実行性能が低下してしまう問題がある. 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

† 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

ザへの負担が大きいことが問題となる.

本稿では, ファイルシステム側で, アクセス集中を検知し, 適切にファイル複製を作成することで, アクセス分散を行う手法を提案し, 提案手法のプロトタイプを Gfarm を用いて実現した. また, Gfarm を用いて, 実際にファイルへのアクセスが集中する例, ファイル複製を作成した場合の理想的なアクセスパターンの例に関して検証を行い, その後, Gfarm とプロトタイプとの比較により, ファイルへのアクセスの自動分散が有効であることを示す.

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統一的なビューでファイルにアクセスができため, 高いユーザ利便性を提供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, 1) 異なるサイト間でも安全なデータ共有ができること (安全性) 2) 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 (高性能性), 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.

安全性を実現した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の認証技術である 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 の拡張であるため高性能性の点で問題があり, 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.

ファイルをストライピングすることで高性能性を実現するファイルシステムとして, PVFS[6], Lustre[7], Google File System[8] などのようなクラスタ型計算機での使用を想定した並列ストライピングファイルシステムが挙げられる. これらのファイルシステムは, 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構成された環境を想定することは望ましくない. グリッド環境は, 既存の計算環境をなるべくそのまま統合して実現するのが望ましい. また, I/O バンド幅がネットワークのバンド幅に制限されるなどの問題もある. チャンクがネットワークに流出することによりネットワークに負荷を与えたり, チャンクデータの安全性を実現しなければならない.

ディスク I/O を積極的に利用することで高性能性を実現するファイルシステムとして, Gfarm[9] が挙げられる. Gfarm は, 並列ストライピングファイルシステムの拡張したものであるが,1 つファイルは, 複数のファイルから構成され, ファイル単位でストレージに分散格納される. Gfarm は, 大規模データを対象にしたデータ・インテンシブ・コンピューティングと呼ばれる科学技術計算を対象にしている. このような計算では, 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ため, Gfarm では, CPU とディスクを統合し, ファイルのあるノードで計算を行うことで, ディスク I/O を積極的に利用し, ネットワーク I/O のバンド幅の制限を避ける.

並列ストライピングファイルシステムでは, ファイルをチャンク単位に分散格納するため, ファイル読み出し時に複数のディスクから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. また, Gfarm ようなファイル単位の分散格納を行うファイルシステムでは, 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の実体へのアクセスをコントロールし, 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複製を用意する. ファイルアクセスの際は, プログラムは, 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, 複製のうちのいづれかにアクセスするようにする. この動作を図を用いて説明する. 図 1 では, 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図る. 提案システムで想定する構成としては, 並列ストライピングファイルシステムで見られるような, 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 ノードの 3 点からなるファイルシステムとする. このようなファイル

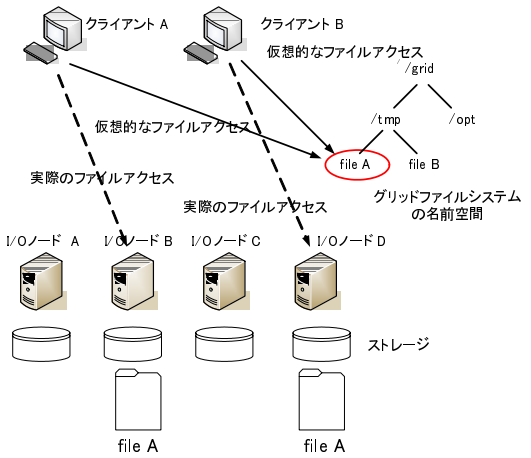

図 1 提案手法によるファイルへのアクセス分散

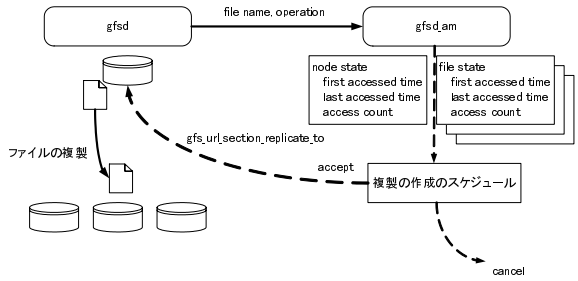

図 2 プロトタイプシステムの構成図

システムでは, ファイルアクセスの際に, 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, メタデータサーバより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, ファイルアクセスの際, ノードへのアクセス集中が発生した場合に, ファイルの実体へのアクセスをコントロールすることで, アクセス集中を制御することができ, 性能向上が期待できると考える.

4. プロトタイプの設計と実装

3 章の提案手法のプロトタイプを Gfarm 上に実装する. これは, Gfarm が, グリッド上のファイルシステムでの実現項目であるユーザ利便性, 安全性, 高性能性を備えている点, ファイル複製作成機能を備えている点, また, 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ためで, 提案手法の実現に適していると判断した.

4.1 プロトタイプの設計

図 2 にプロトタイプの構成図を示す. 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. クライアントが実際にファイルへアクセスする場合は, クライアントが, 複製を持つ I/O ノードへアクセス状況のクエリを出し, 取得した値を元に, アクセスの集中していないノードを選択して, ファイルの実体へアクセスする. また, I/O ノードでは, 自発的にアクセスの集中を検知し, アクセスの集中を検知した場合は, I/O ノード上でアクセス頻度の高いファイルに対し複製を作成するよう指令する.

4.2 プロトタイプの実装

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, Gfarm ファイルシステムを構成する. プロトタイプでは, I/O ノード上で, この gfsd とは別に, ファイルのアクセス状況を管理する gfsd_am デーモンを動作させる. gfsd にその I/O ノード上のファイルに対するオペレーションのリクエストがきた際に, その通知をキャッチして, そのオペレーションとファイルの名前を gfsd_am に通知する. それに対し, gfsd_am は, ノード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, そのノードに最後にアクセスした時刻, 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, そのファイルに対して, 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, アクセス回数を保持し, 通知された情報を記録する. 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以下のように算出する. 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.

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

ファイルアクセスのオペレーションの通知を受ける毎に, ノードアクセス, ファイルアクセスのいずれに関しても, 最後にアクセスした時刻, アクセス回数を更新を行う. クライアントが, 実際にファイルアクセスを行うノードを決定する際に, gfsd は, クライアントからのリクエストを受け, この情報を gfsd_am から取得し, クライアントへ送信する. また, ファイルアクセスのオペレーションの通知を受ける毎に, 複製の作成を試み, 上で定義したアクセス状況の値が閾値を超えるような場合に, アクセスが集中していると判断し, ファイルを他のアクセス状況が低いノードへ複製することを試みる. ファイルの複製は,Gfarm の提供する `gfarm_url_section_replicate_from_to` API を呼ぶことで実現する. ある一定時間内に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情報をキャンセルする. このようにすることで, 各 I/O

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

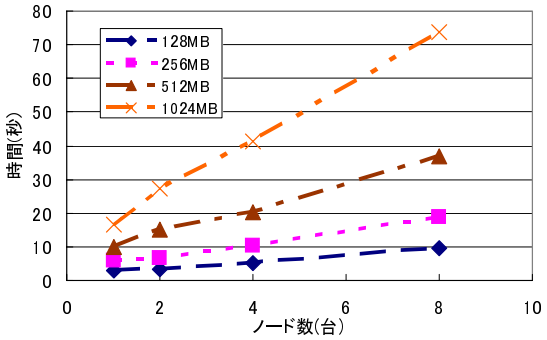

図 3　複数クライアントからの 1 つのファイルへの一斉アクセス

ノードのアクセス状況の均一化を図る.

5. 実　　験

本研究の前提となっているアクセス集中の問題点と提案手法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.

5.1　ファイルアクセス集中の検証

まず, 本研究の前提となっているファイルアクセス集中に関する検証を Gfarm を用いて行う. ファイルを Gfarm 上に 1 つおき, そのファイルに対して, 複数のクライアントから一斉に open, read, close のアクセスを行う. buffer のサイズは 1MByte とした. 結果を図 3 に示す. ノード数が増加するにつれ, リニアに実行時間が増加していることが確認できる. このときのファイルアクセス処理に費やされた時間の内訳を表 2 に示す.　ただし, ファイルサイズを 1024MBytes の場合とする. 結果から実行時間の大部分は,gfs_pio_read に費やされており, gfs_pio_open や gfs_pio_close の際のメタデータサーバへのクエリのためのアクセスの集中の影響はないことがいえる. また, 複数のクライアントから一斉に単一ノードへ一斉アクセスした際のネットワークのスループットを netperf を用いて測定した. 結果を表 3 に示す. 図より, アクセスを行うクライアントが n 台の場合は, クライアントと I/O ノード間のスループットを $MaxThput$ MBytes/sec としたとき, $MaxThput/n$ MBytes/sec 程度しかでていないことが伺える. 以上のことから, ファイルアクセスの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, ファイルアクセスの性能が著しく低下すると考えられる.

表 2　複数クライアントからの単一ノード上の 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
gfs_pio_open	0.00738	0.00105	0.0160	0.0270
gfs_pio_read	16.7	27.1	41.2	73.4
gfs_pio_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

5.2　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を検証するために, 次のような実験を行った. 特定の大きさのファイルを Gfarm 上におき, そのファイルに対して, 8 台のプロセスから一斉にアクセスを行い, このとき, 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制御した. すなわち, クライアントが n 台で, ファイル複製が m 個である場合, ファイル複製 1 つが, n/m 台からのクライアントからのアクセスを受け持つ. 結果を図 4 に示す. 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, 図 3 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, 複製の数が 8 の場合, 各クライアントが各々別の I/O ノードへアクセスが分散しているため, 実行時間の増加が抑えられている. このときのネットワークのスループットを netperf を用いて, 上記の実験と同じようなアクセスになるよう制御して測定した. 結果を図 4 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 (128MBytes/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

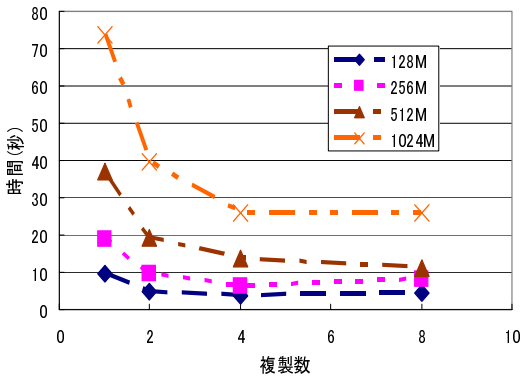

図 4 8 クライアントからのファイルアクセスと複製数の関係

表 4 8 クライアント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

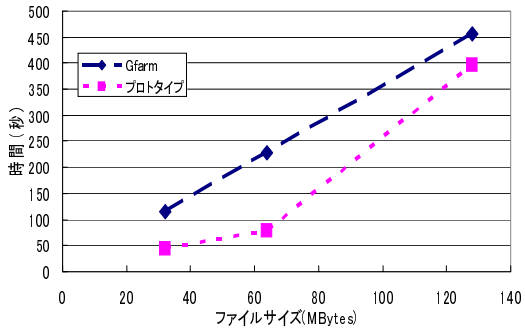

図 5 8 ノードから 1 つのファイルへアクセスを行った際の実行時間 (リモートホストへの複製)

と考えられる.

5.3 他の I/O ノードへのファイル複製の自動作成の効果

他の I/O ノードへのファイル複製の自動作成の効果を検証するために, 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のサイズは,32MBytes,64MBytes,128MBytes とした. 結果を図 5 に示す. Gfarm では, 常に 8 台のクライアントから単一 I/O ノード上の 1 つのファイルへアクセスが集中し, 実行時間が大幅に増加するのに対し, プロトタイプでは, I/O ノードでアクセス集中を検知して自発的にバックグラウンドでファイル複製の作成をすることにより, アクセスの分散が実現でき, ファイルアクセスのための実行時間を抑えることに成功している. 実験では, 32MBytes と 64MBytes のサイズファイルのアクセスの場合は 7 ノードに, 128MBytes のサイズのファイルのアクセスの場合は 3 ノードにファイルを複製していることを確認した. このため, 128MBytes のサイズのファイルのアクセスの場合は, 少数ながら, ファイルアクセスの割り当てが重なってしまい, Gfarm システムにおける 1 つのファイルへの集中アクセスの場合に漸近した結果となっている. 一方, 32MBytes, 64MBytes のサイズ場合は, ファイル複製が多く作成されているため, アクセスが分散して, 良好な結果を示している. 32MBytes と 64MBytes サイズのファイルへのアクセスの場合は, ファイル複製の数が同じであるにもかかわらず, ファイルシステム上の 1 つのファイルへアクセスした場合と比較した実行時間に違いがある. この違いは, ファイル複製へのアクセスの割り当てに偏りがあり, 64MBytes のファイルへのアクセスの場合は, ファイルアクセスが分散して割り当てられているのに対し, 32MBytes のファイルへのアクセスの場合は, ファイルアクセスの割り当てが集中したためであると考えられる.

6. 議　　論

5.1 節のベンチマークより, I/O ノードへのアクセス集中を故意に発生させ, その際, 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ためであることを確認した. 今回の実験ではディスク I/O のオーバヘッドは確認できなかったが, これは, ある特定のファイルを使用したことにより, ディスクのキャッシュが効いている可能性が考えられる. 例えば, ある I/O ノード上の複数のファイルにアクセスが集中した場合, 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 の性能に依存した性能低下が発生することが予測できる. 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, この関係性に関しての検証も必要である.

次に, 5.2 節のベンチマークより, ファイル複製が存在する場合, 理想的には, 1 台のクライアントがファイルへリモートアクセスする程度の時間でファイルアクセスが実現できることを確認した. これは, 今回の実験では, 1 つのスイッチに接続されたクラスタノードを用いて実験を行ったためこのような結果になったと考えている. しかしながら, 現実のグリッド環境は, あるサイト A とあるサイト B を高速なネットワークで結んで構成されることが想定される. このような場

合において, サイト A の I/O ノード上のファイルに対し, サイト B のクライアント群から一斉アクセスした場合には, 5.2 節のように, 仮にファイル複製を作成し, ファイルのアクセス先を上で述べたような理想的な状態に設定したとしても, ネットワーク輻輳が発生し, 1 台の I/O ノードへ複数のクライアントから一斉にアクセスした状況が起きるため, ファイルアクセスの実行時間の増加が発生する予測できる. 実際, スイッチを跨ぐように接続したクラスタノード間において, 5.2 節のような実験を行った場合, 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パラレル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 3 より, 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, ネットワーク輻輳を避けることが必要であると考えられる. 例えば, グリッドを構成するネットワークのトポロジーを把握してネットワークの近さによって I/O ノードをグルーピングし, ファイル複製をそのグループ毎に作成し, グループ内のクライアントからのファイルアクセスは, そのグループ内の複製へ割り当てるようにすることで, ネットワークの輻輳を回避することなどの対策が必要である.

また, 5.3 節の実験より, プロトタイプにおいて, ファイルアクセスの集中の検知が行え, ファイル複製が自動的に作成できることを示した. また, ベンチマークより作成されたファイル複製へアクセスが分散するため, ファイルアクセスが性能向上することを確認した. この際のファイル複製の作成は, 送信元, 受信先が一対一転送で行われていた. 多くのファイル複製を作成を必要とする場合は, 一対多転送で効率良く行うことにより, 更なる性能向上を見込めると予測できる.

7. おわりに

7.1 まとめ

本稿では, ファイルシステム側で, アクセス集中を検知し, 適切に, ファイル複製を作成することで, ファイルへのアクセスの自動分散を行う手法を提案し, 提案手法のプロトタイプを Gfarm を用いて実現した. また, Gfarm を用いて, 実際にアクセス集中を発生させ, その問題点を検証した. クライアント数, 及び, ファイルサイズの増加に応じた実行時間の増加を確認し, アクセス分散の必要性を確認した. 次に, Gfarm を用いて, 理想的なアクセスパターンを検証した. リモートアクセスの場合, ファイル複製数と実際のファイルのアクセス先を制御することで, 1 台のクライアントが I/O のノード上ファイルへアクセスするのと同等の時間でファイルへのアクセスが実現できることを確認した. また, Gfarm とプロトタイプとを用いて, ファイル複製の自動作成の効果について検証した. プロトタイプにおいて, アクセス集中を検知し, バックグラウンドで, 自動的にファイル複製を作成し, それらのファイル複製へアクセスすることで, ファイルへのアクセスの性能向上を確認した.

7.2 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。

- ファイルへのアクセス分散のポリシーを決定するためのグリッドシミュレータを用いたファイルへのアクセス分散のシミュレーション
- 適切なファイルアクセス先の割り当てを実現するためのノードのアクセス状況の一元的なモニタリング
- ファイル複製作成のポリシーに応じたグリッド中への効率の良いデータ転送手法の実現

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).