1A3-OS-17b-3

P2P 技術を用いたファイルシステムの実装とその性能評価

On the implementation of P2P File System and its performance evaluation

小林孝史[*1] 重兼史尚[*2] 田中英昭[*3]
Takashi KOBAYASHI Fumihisa SHIGEKANE Hideaki TANAKA

[*1]関西大学総合情報学部 [*2]有限会社 咲楽屋 [*3]関西大学大学院総合情報学研究科
Faculty of Informatics, Kansai University SACRAYA Inc. Graduate School of Informatics, Kansai University

When the data resources, such as files or directories, are stored as the local resources, we cannot correspond to the various failures. Therefore, we use the network storage system, like as file servers, so the data resources are available to access from anywhere, on anytime. However, the current network storage systems have a single point of failure problem. Therefore, we develop the P2P file system to solve that problem, and we evaluate its performance.

1. はじめに

ファイルシステムは，コンピュータで使用するファイルを格納するための論理的な装置である．ハードウェア等の差異を OS の機能によって隠蔽し，コンピュータの本体等に付属しているハードディスク等の物理的な記憶領域やネットワーク上に配置された記憶領域の差異を OS の機能等によって隠蔽し，それらの異なる記憶領域を統一的に扱うことも可能となっている．近年では，ビッグデータという複雑で膨大なデータを蓄積・処理するために，膨大な記憶容量を持つファイルシステムの需要が高まるとともに，その信頼性，可用性の要求レベルも高くなってきている．

コンピュータ本体等に付属しているハードディスクなどを利用する場合，ハードディスクなどの物理的な媒体を使用することから，その物理的な故障を避けてとおることはできない．その故障問題を緩和するために，それらの物理的な装置を複数台用意することにより，全体での故障確率を下げることが可能になっている．

また，ネットワーク上に配置された記憶領域を利用する場合，ファイルシステムへのアクセスをファイルシステムを提供するサーバとの通信に置き換える機能により，コンピュータ本体等に付属する記憶領域と同様のアクセスを可能にしている．

一方，ファイルシステムには障害が発生することは容易に想定できる．ファイルシステムにトラブルが発生した場合，障害時の対策を行なっていないシステムでは，そのファイルシステムへアクセスができなくなる．起こりうる障害としては，ファイルシステムを構成する物理的な装置の故障，ネットワーク上のファイルシステムを利用する場合ではネットワークのトラブルやサーバのネットワーク的なトラブルが考えられる．そのようなトラブルが発生した場合，障害時対策のできていないシステムではそのファイルシステムにアクセスすることができなくなる．ネットワーク上にあるファイルシステムを利用する場合，そのファイルシステムを提供しているサーバ自体に障害が発生した場合，そのサーバを使用していたクライアントに影響が及ぶことになる．

そこで本研究では，既存ファイルシステムの問題点を解決するために，P2P 型のネットワークを用い，対障害性能の高いファイルシステムの構築を目指している．本論文では，P2P ファイルシステムを構成しているストレージ部と P2P ネットワーク部についての実装について述べ，その性能評価の結果について説明する．

連絡先: 小林孝史：関西大学総合情報学部，〒 569-1095 大阪府高槻市霊仙寺町 2-1-1，taka-k@kansai-u.ac.jp

2. 既存のファイルシステムの問題点

既存のファイルシステムには，単一故障点（Single point of failure）の問題を抱えている．ファイルシステムの構成部品であるストレージ領域，ファイルシステムを提供するためのアクセスサーバなどである．それらの対策として，ストレージを複数台用意したり，それらを制御するコントローラを複数設置しているが，どれもその構成費用を 2 倍以上にしてしまう要素となっている．そして，現状のファイルシステムでは，導入初期段階で構築されている構成をシステムを止めずに変更したいような要求には対応することはできない．例えば，複数の物理的な記憶領域を使用したストレージ領域にさらに記憶領域を追加したい場合には，一度構築済みのストレージ領域を解体して新しい記憶領域を追加し，そののちにストレージの再構築を行い，格納されていたデータを回復する必要がある．

また，ファイルシステムを提供しているサーバ機器を更新する際，現状と同等またはそれ以上の構成のサーバ機器を併設する必要があり，設置場所の問題も発生する．旧サーバに格納されているデータを新サーバに移行することも必要で，その時にデータのコピー作業には非常に長い時間が必要となる．その間にも旧サーバには参照・更新のためのアクセスが発生しており，完全に移行するためにはクライアント等のアクセスが発生しない状況においた上で作業を行う必要がある．新旧のサーバ機器を入れ替える際にも，一旦どちらのサーバ機器もネットワークから切り離すなどの処理が必要で，この時に若干の停止時間が発生することになる．

3. 関連研究

3.1 bigpool

本研究のファイルシステムの性能評価を行ったが，比較対象として，本研究と同様の構造型 P2P アーキテクチャを採用している金子らによる分散ファイルシステム bigpool[金子 2009] を選定した．

bigpool ではディレクトリ構造の表現に，ext2/3 等のファイルシステムでも利用されている i ノードを模した管理手法を用いている．bigpool では，i ノード及びディレクトリエントリを XML で記述し，これらの i ノードファイル，ディレクトリ

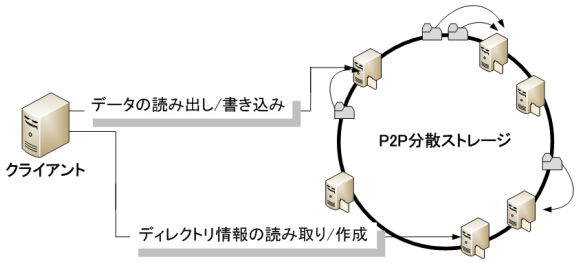

図 1: P2P ファイルシステムの概要

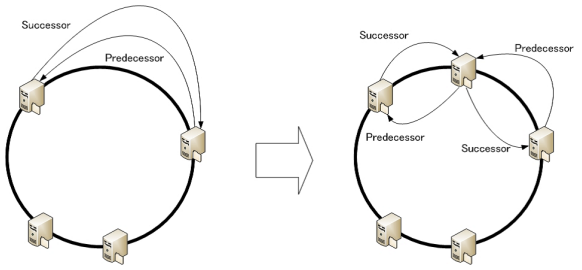

図 2: ノードの新規参加による前後関係の更新

ファイルとして扱っている．i ノードファイルには，その i ノードが示すファイルの名称とそのファイルの種類を記述している．ファイルの種類は通常のファイルまたはディレクトリファイルである．ディレクトリファイルには，そのディレクトリ直下に配置される i ノードファイルの名称，ユーザーに提示するディレクトリやファイルの名称を記述する．

一つのファイルを発見するためには，ファイルシステムのルートディレクトリに存在する i ノードファイルから順にエントリを辿って XML データを解析する必要がある．

3.2 pNFS，GFS

pNFS（parallel Network File System）[pNFS] や GFS（Google File System）[西田 2008] では，メタデータを保持するノードが存在し，各データがどのノードで保持しているかを管理している．pNFS では，クライアントはメタデータサーバに対して問い合わせを行い，その後ストレージへとアクセスを行う．ストレージ中にはデータが分散して保持されており，クライアントは各データサーバに並列にアクセスを行うことが可能である．

GFS では，ファイルをチャンクと呼ばれる一定のサイズに分割し，これをチャンクサーバと呼ばれるデータサーバへ分散して配置する．また，pNFS と同様にマスターサーバと呼ばれる管理サーバがあり，クライアントはこのサーバへ問い合わせを行い，データの所在を知ることができる．

これらの分散ファイルシステムでは，ストレージへの負荷分散やクライアントへのデータを提供し続ける仕組みが実現されているが，管理サーバに問題が生じることでデータへのアクセスができなくなる可能性がある．

4. P2P ファイルシステム

本研究では，これらの既存ファイルシステムの欠点を補うために，ファイルシステムのストレージ部分を分散管理し，それらを P2P ネットワークで接続．また，それらへのアクセスに対するインタフェースも分散化する，P2P 型のファイルシステムを提案している（図 1）．以下の節ではそれぞれの機構について説明する．

4.1 P2P ネットワーク部

本研究のファイルシステムでは，ストレージ部に属するノード群を統一的に管理するノードは存在しない．どのノードもファイルシステムへのアクセスが可能なノードとして扱われており，従来のファイルシステムのような単一故障点となるようなこともない．これらのノードを P2P ネットワーク上に配置することにより，どのノードにアクセスしても同じ情報にアクセスすることが可能になっている．

この P2P ネットワークは Chord[Stoica 2003] を参考にした手法によって構築されている．Chord は構造化 P2P ネットワークを構成する手法の一つで，各ノードはそれぞれ固有の識別子を持っており，その大小関係によって論理的な環状のネットワークを構成する．このネットワーク上で，ある識別子から時計回りに辿って最初に発見されるノードをその識別子の Successor と呼び，反時計回りに辿って最初に発見されるノードをその識別子の Predesessor と呼ぶ．ノードの参加や脱退に伴って各ノードが Successor，Predecessor の情報を更新することにより，環状の P2P ネットワークを保持し続けることが可能になる（図 2）．

また，本研究では，P2P ネットワーク上で各ノードの位置の偏りを緩和するため，実ノード上に仮想的なノードを配置している．仮想ノードもそれぞれ固有の ID を持ち，P2P ネットワークに 1 つのノードとして参加する．現在の実装状況では，実ノードの IP アドレスにパラメータを付加し，これを下にしたハッシュ値から仮想ノード ID を生成している．仮想ノードでも実ノードと同様に Successor，Predecessor，Finger Table を保持している．

ノードの IP アドレスを元に ID を生成し，各仮想ノードの Successor，Predecessor，Finger Table の初期化を行う．新たに参加するノードが P2P ネットワークにおいて 1 台目のノードである場合は，各仮想ノードの持つ Finger Table に各 Key に対応したノードを登録し参加処理は終了する．2 代目以降のノードの参加においては各仮想ノードの Successor，Predecessor の検索を行い，これらノードに対してそれぞれの Predecessor，Successor の更新リクエストを送信し P2P ネットワークへ参加する．また，各仮想ノードの Finger Table の作成及び既存ノードの Finger Table の更新処理を行う．また，ノードの脱退時には後続ノードの Finger Table の更新処理を行い，Successor，Predecessor それぞれの Predecessor，Successor を更新しネットワークからの脱退を行う．

4.2 ストレージ部

ファイルサイズに関係なく，1 つのファイル全体を 1 ノードで管理することとし，絶対パスから求めたハッシュ値を ID として，ファイルの配置ノードを決定する．配置先のノードでは，そのファイルのディレクトリによる区別なく，ストレージ用のディレクトリ配下に配置していく．

ディレクトリについては，特殊なファイルとして通常のファイルと同様にハッシュ値を求めて配置ノードを決定する．ディレクトリファイルの内容はそのディレクトリに存在するファイルの ID 一覧となる．

本システムでは，クライアントにおけるマウントポイントを頂点としたファイルシステムを提供する．ディレクトリ構造を表現するため，ファイルシステム内において作成されたディレクトリに対応したファイルをストレージ中に保持する．し

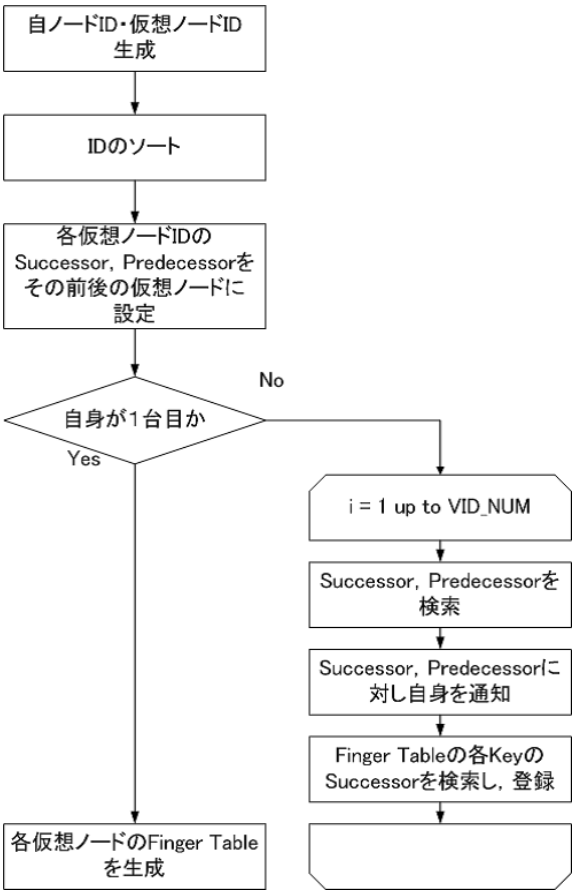

図 3: 新規参加ノードの初期処理

たがって，ストレージにはデータそのものであるファイルと，特定ディレクトリ下のエントリ情報を保持するファイルの２種類が保持される（図 4）．このファイルにはディレクトリへのファイルシステム中における絶対パス名から得たハッシュ値が付与されており，それぞれのディレクトリ下のエントリ情報を保持する．すなわち，ルートディレクトリを含めたファイルシステム中におけるディレクトリと同数の，ディレクトリ情報を保持するファイルがストレージ中に保持される．

4.3 インタフェース部

FUSE（Filesystem in USEr space）[FUSE] をインタフェースとして使用することによって，クライアント上では P2P ファイルシステムを通常のファイルシステムと同様に扱うことができる．マウントポイントへの各種システムコールが FUSE のカーネルモジュールによって P2P ファイルシステムへのアクセスとして置き換えられる．したがって，クライアントの OS では FUSE カーネルモジュールをサポートしている必要がある．

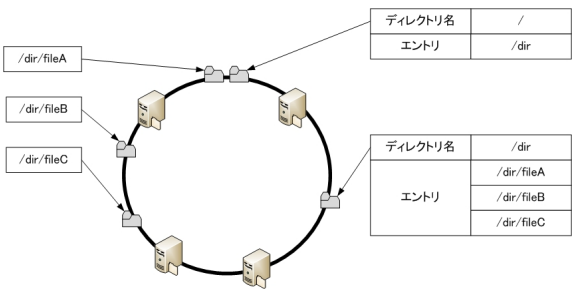

図 4: エントリ情報の保持

FUSE では，`read`，`write`，`getattr` 等の UNIX 向けシステムコールの代替となる処理を記述することができる．本研究では，`getattr`，`readdir`，`open`，`read`，`write`，`mknod`，`unlink`，`utime`，`mkdir`，`rmdir` の各実装を行なっている．これらの実装によって，ファイルやディレクトリへのアクセスに関する一揃いの機能が実現できていることになる．

ここでは，その一部 `getattr` についての実装について述べる．このシステムコールは，ファイルやディレクトリに関する属性情報を取得する際に呼び出されるものである．引数はファイル名やディレクトリ名である．引数に与えられたパスがルートディレクトリである場合，ディレクトリであることを示すフラグをセットするとともに，モード，ハードリンク数を返答用のバッファへ登録する．ここで返答した値は，マウントポイントとなるディレクトリそのものの属性として使用される．

ルートディレクトリ以外のファイルやディレクトリの属性情報の取得を行う場合，パス名からその直上のディレクトリまでのパスを取得し識別子を生成する．例えば，指定されたパスが “/dir/fileA” である場合，“/dir” までのパス名からハッシュ関数を用いて識別子を生成する．

次に，生成された識別子の Successor の検索を行う．Successor の発見後，このノードに対して READ_DIR サービスを呼び出し，属性情報の取得を行う．READ_DIR サービスにおいては，指定されたディレクトリ下のエントリ情報を保持するファイルから情報の読み出しを行い，クライアントへ返答する．返答されたエントリが指定されたものである場合，その属性情報をバッファへ書き込み，処理は終了する．

5. 性能評価

性能評価の指標として，ファイルの作成・削除，ディレクトリの作成・削除にかかる時間を採用している．

図 5 は 1000 個までのファイル作成時の 1 個あたりの所要時間，図 6 には同様に 1000 個までのファイルの削除の際の 1 個あたりの所要時間である．ファイルの作成・削除ともに bigpool はファイル数が多くなるに従って所要時間も増えていくが，本研究の構成では所要時間はファイルの個数に関係なく，ほぼ一定となっている．

この違いは，bigpool がファイル等の情報管理に XML を利用しているのに対して，p2pfs についてはパスから得られた ID 情報のみでファイルを配置しているため，XML での記述を走査する時間が大きく出たものと考えられる．一般的に，XML ファイルの走査にはかなりの時間がかかると言われており，この比較実験はその影響が顕著な例であると言える．

次に，既存のファイルシステムとの比較を行った結果が図 7 及び図 8 である．本研究で提案している P2P ファイルシステムでは，ファイルシステムへのアクセスに必ずネットワーク通信及びピア間の通信が含まれるため，既存のファイルシステムと比較してある程度の性能低下が見込まれる．図 7 と図 8 はその性能低下の度合いを示すことになる．

これらの図より，ローカル資源を用いている ext3 ファイルシステム，ネットワーク上にあるファイルサーバへアクセスする NFS についてはかなり短い時間でファイルの作成・削除が可能になっているが，p2pfs についてはそれらよりも 6〜7 倍

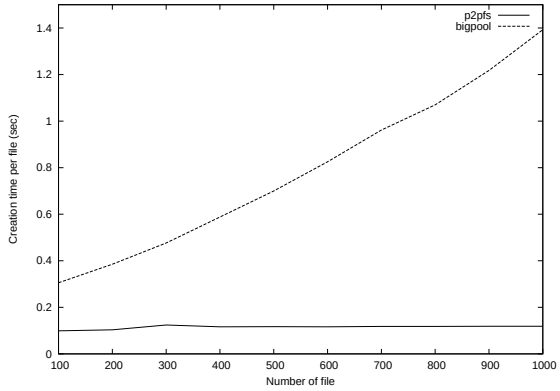

図 5: ファイル作成時間の比較

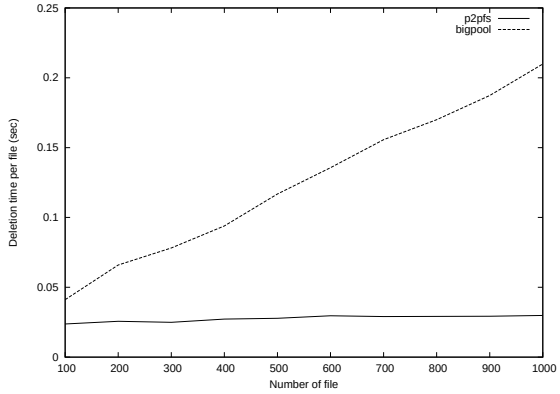

図 6: ファイル削除時間の比較

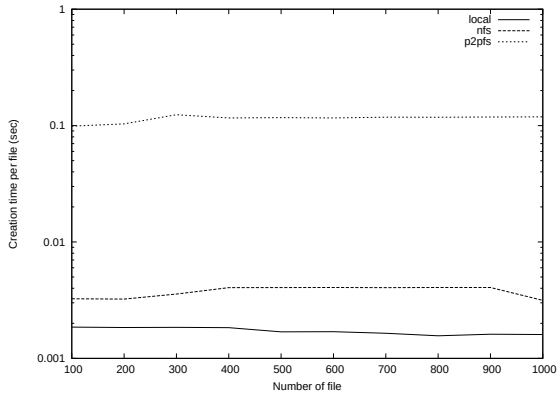

図 7: ext3，NFS 及び p2pfs でのファイル作成時間の比較

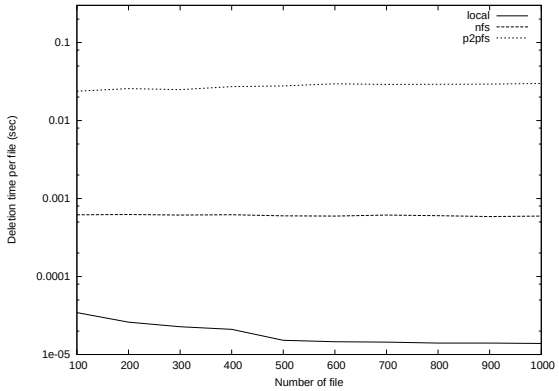

図 8: ext3，NFS 及び p2pfs でのファイル削除時間の比較

も遅いということがわかる．

6. おわりに

本論文では，P2P ファイルシステムの実装とその性能評価について述べた．本システムでは，既存の分散ファイルシステムの問題点を緩和するための機能として，管理ノードを省くことができる P2P 型のストレージ部によるデータ管理手法と，それを通常のファイルシステムと同様に扱うことが可能になるインタフェース部を有している．

本システムは，ファイルシステムとしての最低限の機能を実装しているが，実際に使用できる領域のサイズを調べることができないなど，一部の未実装の部分があるため，今後も継続的に開発を行なっていく予定である．

参考文献

[FUSE] ”FUSE: Filesystem in Userspace,” http://fuse.sourceforge.net/ (2012 年 2 月 10 日確認).

[pNFS] ”pNFS,” http://www.pnfs.com/ (2012 年 2 月 10 日確認).

[Stoica 2003] Ion Stoica, Robert Morris, et. al., ”Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Apprications,” Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications, pp.149–160 (2003).

[金子 2009] 金子豊, 黄民錫, 竹内真也, 和泉吉則, ”構造型 P2P を使った分散ファイルシステムにおける分散ディレクトリシステム,” 情報科学技術フォーラム講演論文集 8(4), 213-216 (2009).

[西田 2008] 西田圭介, ”Google を支える技術,” 技術評論社, 2008.