L-033

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管理手法
Decentralized Directory Management on Distributed File System based on Structured P2P

金子 豊† Yutaka Kaneko　黄 民錫† Minsok Hwang　竹内 真也† Shinya Takeuchi　和泉 吉則† Yoshinori Izumi

1. はじめに

放送局のテープレス化が始まり，現在 VTR テープで収録している番組素材や完成番組は，将来はすべてファイル化する予定である．放送局に存在する膨大な素材ファイルを共有利用する方法として，複数のストレージサーバを仮想的な 1 つのストレージとして扱うことができる分散ファイルシステムが有効である．筆者らは構造型 P2P を使った DHT(Distributed Hash Table)による広域分散ファイルシステムの検討を進めており，これまでに OneHop を拡張したサーバの物理的な位置を考慮したファイル管理方式を提案した[1].

現在のパーソナルコンピュータでは階層ディレクトリによるファイル管理が利用されている．そのため，互換性を持たせるには分散ファイルシステムでも，ユーザが互いに共通のディレクトリ名でファイルへアクセスできることが必要である．大規模な広域分散ファイルシステムとして開発された GFS[2]や Grid Datafarm[3]では，ディレクトリ管理をメタデータサーバで集中管理している．しかし，集中管理機構のない P2P 型の分散ファイルシステムに，集中管理方式を用いることは P2P を導入したメリットが薄れる．

本報告では特定のメタデータサーバを必要としない分散ディレクトリ管理手法を提案する．また，提案したディレクトリ管理手法を実装した分散ファイルシステムを試作し，実験による性能評価を行ったので報告する．

2. 分散ディレクトリ管理

BSD から派生した UNIX 系 OS で利用されている FFS(Fast File System)や ext2/3 などのファイルシステムでは，図 1 (a)に示すように，ファイル（ディレクトリエントリやユーザデータ）に対して 1 つの i ノードを割り当て，ディスク上のデータを管理する[4]．ディレクトリエントリには，i ノード番号とファイル名が保存される．ファイルシステムに保存したファイルは，ルート i ノードを起点に，ディレクトリエントリと i ノードを順に検索することで，階層ディレクトリの中から見つけることができる．

提案する分散ディレクトリ管理手法では，この i ノードおよびディレクトリエントリの関係を利用する．提案方式では，図 1(b)に示すように i ノードおよびディレクトリエントリをそれぞれ XML で記述したファイルとする．これらを i ノードファイルおよびディレクトリファイルと呼ぶ．これらのファイルはユーザデータのファイルとともに，分散ファイルシステム内のサーバに保管する．

i ノードファイルには，その i ノードが示すファイルのファイル名とタイプ（データファイルかディレクトリエントリか）を保存する．ディレクトリファイルにはディレクトリエントリとして，i ノードのファイル名とユーザが利用するディレクトリ名またはファイル名を格納する．

†日本放送協会 放送技術研究所, NHK

分散ファイルシステムに保管したファイルは，ルートの i ノードファイルから順に i ノードファイルとディレクトリファイルの XML データを解析することで見つけることができる．

3. 分散ファイルシステム

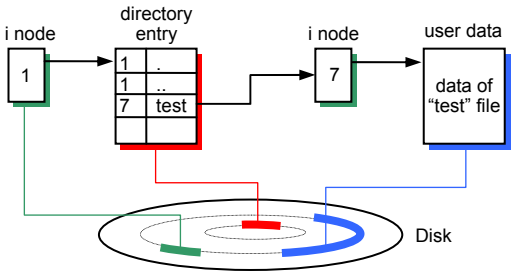
本章では提案する分散ディレクトリ管理手法を実装した構造型 P2P による分散ファイルシステムについて述べる．

3.1 全体構成

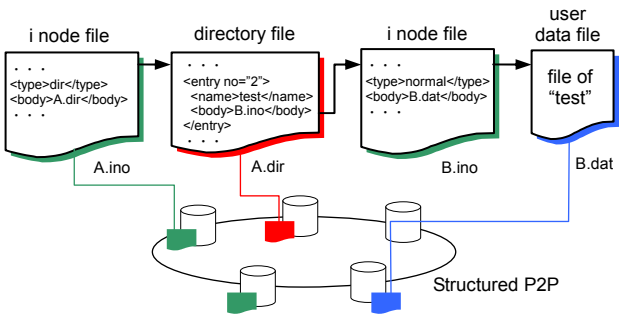
図 2 に試作した分散ファイルシステムの全体構成を示す．分散ファイルシステムは複数のサーバが構造型 P2P によるネットワークオーバレイを構成する．構造型 P2P の方式には OneHop 拡張方式[1]を用いる．

保管する i ノードファイル，ディレクトリファイル，ユーザデータファイルは，分散ファイルシステム内で一意に特定するためにファイル生成時に各サーバが ID を生成する．ID には UMID[5]を用い，UMIID をファイル名として各サーバのローカルファイルシステムに通常ファイルとして保存する．UMID は SMPTE で規格化された 32 バイトのユニーク ID であり，サーバの MAC アドレスや日時情報などを使って生成する．

各サーバに格納されたファイルは，OneHop 拡張方式による<key, value>ルックアップテーブルを検索することで保管先サーバを知ることができる．すなわち，ファイル名のハッシュ値を key，サーバアドレスを value とし，各サー



(a) i ノードによるディレクトリ管理



(b) 分散ディレクトリ管理(提案方式)

図 1 i ノードによるディレクトリ管理と提案方式

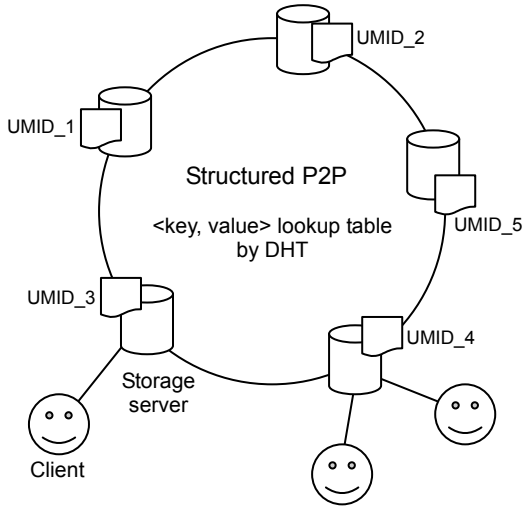


図２ 分散ファイルシステムの全体構成

バが保管しているファイルのファイル名をルックアップテーブルに登録することで，ファイル名から格納先サーバを検索できる．

分散ファイルシステムのクライアントはサーバのインタフェースを介してファイルの操作を行う．クライアントとサーバ間は NFS と同様，コネクションレスのファイル操作とし，サーバ側では状態を保持しない．サーバとクライアント間の通信には CORBA-IIOP を用いた．

3.2 ストレージサーバ

サーバの主なインタフェースを図 3 に示す．インタフェースは低レベルインタフェースと高レベルインタフェースに分かれる．低レベルインタフェースはサーバが保管しているファイルへの操作関数であり，UMID のファイル名で指定する．高レベルインタフェースはディレクトリを含む分散ファイルシステム内のファイルへの操作関数であり，ディレクトリ階層を含む絶対パス名で指定する．

低レベルインタフェースの get_key 関数は，<key, value> ルックアップテーブルの検索用関数である．get_stat, rea_file, wri_file, del_file の各関数は指定したファイル名(fname)のファイル情報の取得，読み出し，書き込み，削除を行う．

高レベルインタフェースの bp_get_stat, bp_rea_file, bp_wri_file, bp_mk_file, bp_rm_file の各関数は，指定された絶対パス名(path)のファイルのファイル情報の取得，読み出し，書き込み，作成，削除を行う．同様に，bp_mk_dir, bp_rm_dir, bp_rea_dir は絶対パス名(path)のディレクトリの作成，削除，ディレクトリ情報の読み出しを行う．また，bp_rename はファイルまたはディレクトリ名を変更する．bp_get_ior 関数は絶対パス名のファイルまたはディレクトリの UMID のファイル名と保存先のサーバアドレス(IOR)を取得する．

高レベルインタフェースでは，指定されたパス名から実際のファイルを検索するため，パス検索処理を実行する．パス検索処理は，ルートの i ノードファイルを起点に i ノードファイルおよびディレクトリファイルのダウンロードと，XML の解析を繰り返すことで行う．本システムはコネクションレスのファイルシステムであるため，関数の呼び出し毎にパス検索が実行される．

```
// 低レベルインタフェース
get_key(in string key, out StringList values);
get_stat(in string fname, out FileStat stat);
rea_file(in string fname, in ULONG offset, in ULONG size,
        out BinaryData buf);
wri_file(in string fname, in ULONG offset, in ULONG size,
        in BinaryData buf);
del_file(in string fname);

// 高レベルインタフェース
bp_get_stat(in string path, out FileStat stat);
bp_rea_file(in string path, in ULONG offset, in ULONG
        size, out BinaryData buf);
bp_wri_file(in string path, in ULONG offset, in ULONG
        size, in BinaryData buf);
bp_mk_file(in string path, in ULONG mode);
bp_rm_file(in string path);
bp_rea_dir(in string path, out StringList dnames);
bp_mk_dir(in string path, in ULONG mode);
bp_rm_dir(in string path);
bp_rename(in string from_path, in string to_path);
bp_get_ior(in string path, out string iname, out string body,
        out string ior_inode, out string ior_body);
```

図３ サーバの主なインタフェース(OMG IDL)

ファイルの新規作成時にファイルを保管するサーバは，参加しているサーバからラウンドロビンで選択する．

3.3 クライアント

クライアントは任意のサーバのインタフェースを介して分散ファイルシステム内のファイルにアクセスする．クライアントからサーバを利用する形態には図 4 の方法Ⅰ～Ⅲの３通りが考えられる．

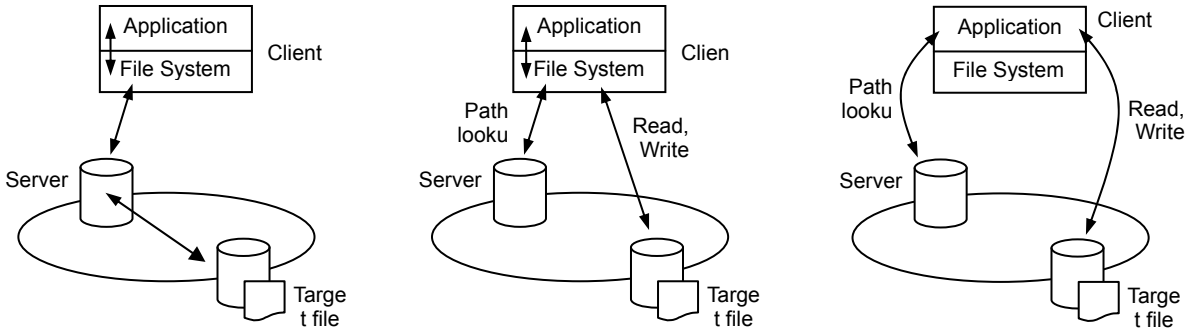
方法Ⅰではクライアントのファイルシステムから特定のサーバを介してすべてのファイル操作を行い，方法Ⅱではファイルのデータへの読み書きは保存先サーバと直接行う．方法Ⅲはクライアントのファイルシステムは介さずアプリケーションから直接サーバのインタフェースを実行する．

方法ⅠおよびⅡはファイルシステムを介して分散ファイルシステムにアクセスするため，既存のアプリケーションをそのまま利用できるメリットがある．方法Ⅲはクライアントのファイルシステムのオーバヘッドが削減できるが，アプリケーションを変更する必要がある．一方，方法Ⅰでは，クライアントは１台のサーバとのみ接続できればファイルの操作が可能であるが，方法ⅡとⅢはクライアントが全てのサーバと接続できなければならない．

試作した分散ファイルシステムのクライアントは，FUSE(Filesystem in Userspace)[6]を用いて実装した．FUSE はユーザが作成したユーザ空間のファイルシステムを実行するファイルシステムモジュールである．

4. 性能評価

本章では表１の４台のサーバを使った実験結果について述べる．



(a) 方法 I　(b) 方法 II　(c) 方法 III

図 4 クライアントからのアクセス方法の分類

4.1 ファイル，ディレクトリの作成と削除

分散ファイルシステムのルートディレクトリへ 1,000 個のファイルの作成と削除を行った場合の 1 ファイルあたりの処理時間を図 5 (a)に示す．同様にディレクトリの作成と削除を行ったときの処理時間を図 5 (b)に示す．ファイルの作成には touch，削除には rm，ディレクトリの作成には mkdir，削除には rmdir を用いた．

ファイル，ディレクトリの作成，削除ともに，すでに保存されているファイル数が多くなるほど，処理時間が長くなる．これは，同一ディレクトリ内にファイルが存在するほど，ディレクトリファイルの XML 解析に時間がかかるためである．ディレクトリファイルの XML から目的のファイル名を検索するために，リニアサーチを用いている．削除時の処理時間は，XML データのサーチ時間に影響され，処理時間は測定した最小値(min)と最大値(max)の間となる．

ディレクトリの作成に比べファイルの作成に時間がかかるのは，touch コマンドでは，内部で mknod, write, utimes など，複数のファイル操作のシステムコールが呼び出されているためである．

図 5 (c)に 100 階層のディレクトリの作成，削除をしたときの 1 ディレクトリあたりの処理時間を示す．ディレクトリ階層が深くなるほど，パス検索のためにファイルの取得と XML 解析が繰り返し実行されるため，処理時間が長くなる．

NFS での同様の実験ではファイル数や階層数にほぼ影響なく 10msec 以下と提案方式に比較して高速であった．提案方式の高速化は今後の課題であるが，処理速度の向上にはパス検索の高速化が必要であり，そのためには i ノードファイルおよびディレクトリファイルのキャッシュの利用，XML 解析の高速化が有効と考えられる．

4.2 ファイルの読み書き

read, write システムコールにより 1.5G バイトのファイルを転送したときの転送速度を図 6 に示す．

図 6 (a)は FUSE をデフォルトの状態で用いた場合である．比較として NFS の実験結果を示した．FUSE および NFS ともに，内部的にバッファキャッシュを用いてリモートと通信しているため，read，write の呼び出しメッセージサイズとは関係なくほぼ一定の転送速度となった．バッファキャッシュとして NFS では 32K バイト，FUSE では読み出しに 128K バイト，書き込みに 4K バイトが使われている．提案

表 1 実験に用いたサーバのスペック

CPU	Opteron(DualCore, 3.0GHz) x 2
Memory	8GB
HDD	2.5inch SAS (73GB, 15,000rpm) 5 台による RAID0
OS	Linux 2.6.22.16
Local File System	ext3
FUSE	2.7.4

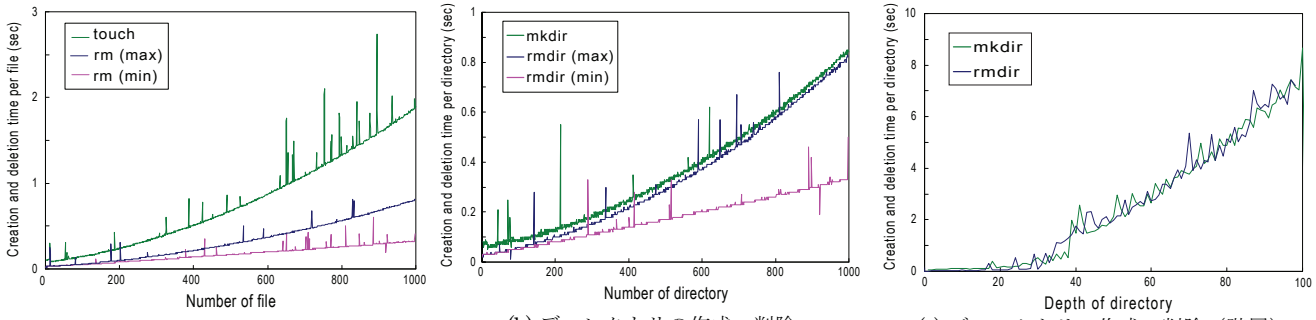
のディレクトリ管理方式では，インタフェースの呼び出し毎にパス検索をするため，方法 I では転送速度が大幅に劣化することがわかる．方法 II では，FUSE の先読みの機構(read-ahead)もあり 930Mbps と，NFS の 860Mbps よりも高速な転送速度となった．しかし，書き込みはメッセージサイズに無関係に 4K バイト単位となるため，書き込み時の速度は NFS の 760Mbps に比べて 90Mbps と極端に遅い結果となった．

図 6 (b)は FUSE の direct_io モードを利用した場合の結果である．direct_io モードでは，バッファキャッシュや先読みを行わず，read, write のメッセージサイズで読み出し，書き込みが行われる．そのため，メッセージサイズが大きいほど転送速度が速くなる．方法 III では FUSE のオーバヘッドが無いため方法 II に比べて良い結果となった．

以上の結果から，クライアントが全サーバと接続可能な状況では，方法 II および方法 III が有利であり，特に読み出しに関しては FUSE の direct_io を使用せず方法 II を用いるのが有効である．ただしその場合，書き込み速度のパフォーマンスを改善するために，バッファサイズを変更するなどの改善策が必要である．

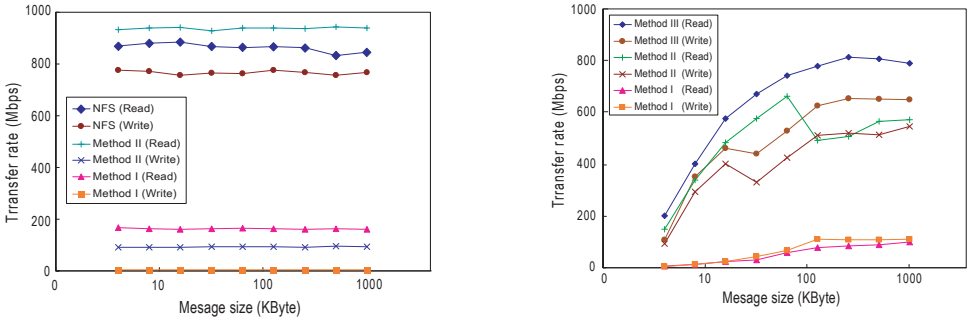
5. 拡張機能

ファイルの操作は一般には read, write, seek などの抽象化された基本的な関数で操作しているが，ファイルシステムに固有の機能を追加することで，処理の効率化や利便性の向上が期待できる．提案方式は i ノードとディレクトリエントリの情報を XML で記述したファイルとしてサーバ内で通常のファイルとして保存しているため，特にファイルサイズに制限は無い．これらの XML 記述に情報を保存し利用することで，様々な固有機能が実現できる可能性がある．以下，検討中の機能について述べる．



(a) ファイルの作成，削除　(b) ディレクトリの作成，削除（同一ディレクトリ内）　(c) ディレクトリの作成，削除（階層）

図５ ファイル，ディレクトリの作成，削除の処理時間



(a) デフォルト(direct_io=off)　(b) direct_io モード(direct_io=on)

図６ データの読み出し，書き込み速度

5.1 メタデータの付加

放送局では映像や音声などのデータに各種メタデータを付加したいケースが多い．開発中の分散ファイルシステムでは，ユーザがファイルに自由にメタデータを読み書きできるように，ｉノードファイルへのメタデータの読み出しと書き込みインタフェースを追加する予定である．

5.2 マルチレコード

１つのファイルを複数のレコードの集まりとして扱える構造を検討している．放送局のテープレス化では，大容量ファイルの高速転送が課題の１つとなっている．文献[7]では，ファイルにブロック単位のメッセージダイジェスト値を付加し，変更箇所だけを高速に検出，転送する手法を提案した．データに対してメッセージダイジェストデータを付加するなど，ファイルに対して付加データを一緒に保存したい場合，マルチレコードのファイル構造が有効である．また，GFS や Grid Datafarm では，１つのファイルを複数のチャンクに分割して保存しているが，マルチレコードのファイルシステムを用いることで，１つのファイルを複数のレコードで構成することも実現できる．

5.3 高速な部分挿入・削除

テープレス化の課題として映像や音声などのファイル内の部分変更の高速化がある．文献[8]では，ext4 ファイルシステムに挿入・削除機能を付加し，MXF ファイルの高速編集手法を提案した．このような機能を分散ファイルシステムにも必要であると考えている．

6. まとめ

構造型 P2P による分散ファイルシステムを構築するため，特定のメタデータサーバを必要としない分散ディレクトリ管理手法を提案した．また，試作した分散ファイルシステムの概要と，基本的な性能測定結果について述べた．提案した分散ディレクトリ管理方式により通常のファイルシステムと同等に利用できることを確認した．今後は複数のクライアントから利用したときの性能評価を行う予定である．また，試作した分散ファイルシステムは NFS に比較して速度性能の面で不十分な点があるため，今後パス検索処理の性能改善などが必要である．

参考文献

[1] 金子, 竹内, 南, 和泉, “OneHop-P2P 拡張方式の実装方法と性能評価”, 信学技報, NS2008-52, pp.57-62, 2008.
[2] S. Ghemawat, H. Gobioff, S.-T. Leung, “The Google File System,” Proc. SOSP2003, pp.20-43, 2003.
[3] 建部, 森田, 松岡, 関口, 曽田, “広域大規模データ解析のための Grid DataFarm アーキテクチャ”, 情処技報, HPC87-31, pp.177-182, 2001.
[4] M.K.McKusick, G.V.Neville-Neil, “BSD カーネルの設計と実装－FreeBSD 詳解－”, アスキー, 2005.
[5] SMPTE 330M-2004, “Unique Material Identifier (UMID). ”
[6] http://fuse.sourceforge.net/
[7] 南, 金子, 竹内, 藤沢, 和泉, “番組ファイルの高速差替手法”, 映メ技報, BCT2008-89, pp.1-4, 2008.
[8] 金子, 南, 黄, 竹内, 和泉, “挿入削除機能付きファイルシステムによる MXF ファイル編集の性能評価,” 映情年大, 2009.