DEWS2004 4-A-01

並列分散ストレージにおけるフラットなデータ管理への階層構造の導入手法

小原　俊樹[†]　渡辺　明嗣[††]　花井　知広[††]　山口　宗慶[††]　小林　大[††]

上原　年博[†††]　林　直人[†††]　田口　亮[†††]　横田　治夫[††††,††]

† 東京工業大学工学部情報工学科　〒 152-8552 東京都目黒区大岡山 2-12-1

†† 東京工業大学大学院情報理工学研究科計算工学専攻　〒 152-8552 東京都目黒区大岡山 2-12-1

††† NHK 放送技術研究所　〒 157-8510 東京都世田谷区砧 1-10-11

†††† 東京工業大学学術国際情報センター　〒 152-8550 東京都目黒区大岡山 2-12-1

E-mail: †obara@de.cs.titech.ac.jp, ††{aki,hanai,muu,daik}@de.cs.titech.ac.jp,

†††{uehara.t-jyg,hayashi.n-gm,taguchi.r-cs}@nhk.or.jp, ††††yokota@cs.titech.ac.jp

あらまし　データの管理構造として，負荷分散・耐故障性の観点からはフラットな構造が望ましく，人間がデータをアクセスするという観点からは階層構造が取れることが望ましい．この相対する要求の両立が望まれる．我々が，高信頼で高機能な並列分散ストレージとして提案してきた自律ディスク及び，その応用である自律 Web サーバはフラットな管理構造である．一方，分散環境下の Web コンテンツ編集の標準プロトコルとして規定された WebDAV では，階層構造のサポートを要求している．本稿では，自律 Web サーバで WebDAV の階層構造を実現することを通して，フラットなデータ管理構造と階層構造を両立させる方法を提案する．

キーワード　ストレージシステム, Web とインターネット, アクセスパス, 分散 DB, 並列 DB

An Introduction Method of Hierachical Structure to Flat Data Management in Distributed-Parallel Storage

Toshiki OBARA[†], Akitsugu WATANABE[††], Tomohiro HANAI[††], Munenori YAMAGUCHI[††], Dai KOBAYASHI[††], Toshihiro UEHARA[†††], Naoto HAYASHI[†††], Ryo TAGUCHI[†††], and Haruo YOKOTA[††††,††]

† Department of Computer Science, Faculty of Engineering, Tokyo Institute of Technology
2-12-1 Oookayama Meguro Tokyo, 152-8552 Japan

†† Department of Computer Science, Graduate School of Information Science and Engineering,
Tokyo Institute of Technology

††† NHK Science & Technical Research Laboratories　1-10-11 Kinuta Setagaya Tokyo, 157-8510 Japan

†††† Global Scientic Information and Computing Center, Tokyo Institute of Technology

E-mail: †obara@de.cs.titech.ac.jp, ††{aki,hanai,muu,daik}@de.cs.titech.ac.jp,

†††{uehara.t-jyg,hayashi.n-gm,taguchi.r-cs}@nhk.or.jp, ††††yokota@cs.titech.ac.jp

Abstract　From the viewpoint of load balancing, flat data management structures are suitable, while hierarchical data management structures are useful to be accessed from users. It is important to satisfy both of conflicting demands. We have proposed the Autonomous Disk as a highly reliable and functional distributed intelligent parallel storage, and Autonomous Web Server as an application of the Autonomous Disk, which uses a flat data management structure. On the other hand, the WebDAV proposed as a standard protocol for managing Web contents assumes hierarchical data management structures. In this paper, we propose methods to implement the hierarchical data management structure of the WebDAV on the flat data management structure in Autonomous Web Server.

Key words　storage system, Web and the Internet, access path, distributed database, parallel database

1. はじめに

コンピュータで扱う情報量は，近年急激に増加し続けている．特に，昨今のネットワーク技術の発達やストレージコストの低下により，コンピュータ内に蓄積されるデータ量は著しく増加している．性能や信頼性の面から，大容量ストレージは並列分散構成されることが前提となるが，その構成要素である個々のストレージの間でのアクセス負荷の分散や容量の分散が重要である．その結果，ストレージにおけるデータの管理コストの増加が大きな問題となっている．

データの格納方法として，ユーザの観点としてはユーザの概念構造を反映した階層構造が取れることが望ましい．一方，負荷分散・容量分散の観点からはフラットな構造で管理することが望ましい．これは，負荷分散・容量分散のためにデータの移動において概念構造に基づく階層構造があると，データ移動時に構造が制約になり，管理コストが高くなるからである．ユーザには階層構造を提供しつつ，データ管理においてフラットなデータ移動ができることが望まれる．

我々が，高信頼で高機能な並列分散ストレージとして提案してきた自律ディスク [1] 及び，そのアプリケーションである自律 Web サーバ [2] はフラットな管理構造を前提としている．一方，分散環境下の Web コンテンツ編集の標準プロトコルとして規定された WebDAV（Web-based Distributed Authoring and Versioning）[3] では，階層構造を前提としている．

本稿では，自律 Web サーバで WebDAV の階層構造を実現することを通して，フラットなデータ管理構造を持った並列分散ストレージを用いながらユーザに階層構造を提供する方法を提案する．そして，提案方法を実際に PC 上へ Java で実装された自律ディスク上で WebDAV サーバとして実装し，性能の比較を行った．

2. 背景技術

2.1 自律ディスク

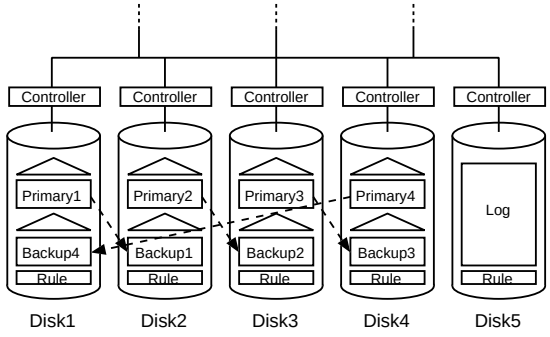

図 1　自律ディスク

我々は以前より，高信頼で高機能な並列分散ストレージとして自律ディスクを提案してきた．図 1 はその標準的構成である．自律ディスクはネットワークに接続してクラスタ化することを前提としている．クライアントは自律ディスク内のデータのアクセスするために，クラスタ内の任意のノードにリクエストを発行する．クラスタ内のノードは局所的な通信を行い，協力してこのリクエストを対処する．データに対するアクセスは，分散ディレクトリをトラバースしリクエストを適切なノードに転送することにより行われる．この自律ディスクの性質として，データ分散，ホストからの均質的なアクセス，同時実行制御，アクセス偏り制御，耐故障性，異種性が挙げられる．異種性とはストリームインターフェースを持つことを意味しており，このインターフェースを介してやり取りされるデータの識別子をストリーム ID という．

自律ディスクのアプリケーションとして，自律ディスクに HTTP インターフェースを付加したものである自律 Web サーバやそのマルチメディアコンテンツのための拡張であるマルチメディアサーバ [4] を提案している．

表 1　WenDAV で提供されるメソッド

メソッド名	処理内容
GET	リソースに関する情報の取得
HEAD	リソースに関するヘッダ情報の取得
POST	データ送信
PUT	非コレクションリソースの作成・更新
DELETE	リソースの削除
PROPFIND	プロパティの取得
PROPPATCH	プロパティの変更
MKCOL	コレクションの作成
COPY	リソースの複製
MOVE	リソースの移動
LOCK	リソースのロック
UNLOCK	リソースのロック解除

2.2 WebDAV

WebDAV [3] は，HTTP/1.1 の拡張で，分散環境下における Web コンテンツの編集を行うことを目的として規定されたプロトコルである．

WebDAV においては階層構造をサポートし，その中間ノードにあたるコレクションに対する操作も規定している．

以下，WebDAV の規定で使用されている用語のうち，データ格納に関するものについて記述する．

リソース　WebDAV で操作・管理の対象となるもの．一般的なファイルシステムにおけるファイルにあたるものである．

コレクション　複数のリソースをまとめて管理するための概念で，一般的なファイルシステムにおけるディレクトリにあたるものである．コレクションもリソースの一種である．

プロパティ　リソースの性質を名前と値の対で定義したものである．プロパティには，サーバが管理しているデータとクライアントで管理するものの両方が格納できる．

WebDAV で提供する操作は表 1 の通りである．これらのメソッドを利用して，ユーザは分散編集を行う．

3. 階層構造の提供

前章で述べたように，自律ディスクはそのアプリケーションとして，HTTP を拡張したインターフェースを持つマルチメディアサーバが存在する．そして今後，分散編集やバージョン管理

が行えるようになることが必要である．一方で，Web も分散編集可能な高信頼サーバが要求されてきている．したがって，自律ディスクが WebDAV インターフェースを持つことは非常に有用である．

このとき，自律ディスクに WebDAV を導入するにあたり問題となるのは，自律ディスクがフラットなデータ管理構造であるのに対して，WebDAV では階層格納構造を前提としていることである．以下，フラットなデータ管理構造をもった並列分散ストレージに階層格納構造を導入する方法を検討する．

階層構造の提供に際して，階層構造をフラットなデータ管理のストレージ上にマッピングを行うというアプローチを取る．そのためには，フラットなストレージ上へのマッピングである格納方式とその格納されたデータへのアクセス方法を考える必要がある．これらをまとめて，本稿ではアクセス・格納方式と表記する．また，フラットなストレージ上にマッピングされたデータの識別子を内部 ID と書く．

本稿では階層構造について記述する場合にリソースやコレクションという WebDAV 用語を用いるが，これらは WebDAV に依存しない普遍的な概念である．リソースは階層構造を木構造で捉えたときの葉ノードにあたり，コレクションはそれ以外のノードに相当するものである．また，コレクションは直下のリソースの名前とその格納位置を把握しているものとする．

3.1 リソースへのアクセス・格納方式

フラットなストレージ上にマッピングされたリソースへのアクセス・格納方式については，大別して以下の 2 種が存在する．

コレクション・トラバース法（CT） コレクション階層をたどってアクセスする．

アクセスパス・マッピング法（APM） コレクション階層をたどるアクセスパスを利用して直接アクセスする．

本節では，これらについて述べる．

3.1.1 コレクション・トラバース法（CT）

コレクション・トラバース法は，コレクションが持っている参照情報を元に階層を降下しながらアクセスしてゆくものである．例えば，図 2 の (c) の位置にあるデータにアクセスする場合，次のような手順を踏む．

（1） まず，(a) にアクセスし，(a) が持っている参照情報から (b) の内部 ID を取得する．

（2） 次に，(b) にアクセスし，(b) が持っている参照情報から (c) の内部 ID を取得する．

（3） そして，(c) にアクセスする．

この方法は，階層構造を提供するストレージにおいて一般的なアクセス方法である．

この方式はアクセスコントロールと親和性が高い．アクセスコントロールは，リソースごとのプロパティとしてアクセス権限リストを記録しておき，コレクション階層を降下するとときに，そのコレクションにアクセス権限があるかを問い合わせることにより実現できるが，これはコレクション・トラバース法のアクセス方式そのものである．したがって，コレクション・トラバース法とアクセスコントロールは親和性が高い．

このアクセス方法に適した格納方式の条件としては，別のリソースと内部 ID が重複しないことの他に，親コレクションの内部 ID に依存しないことが存在する．これは，コレクションに対する名前変更や移動といった構造変化に対する耐性を持たせるためである．これらの条件を満たす内部 ID の付与方法としては，挿入時刻や順番等に依存して付ける方法が挙げられる．

3.1.2 アクセスパス・マッピング法（APM）

コレクション階層におけるアクセスパスのみで導出できるような内部 ID を採用すれば，階層を降下していく過程を省略し，直接データにアクセスすることができる．

例えば，図 2 の (c) の位置にあるデータにコレクション・トラバース法でアクセスする場合，(a) → (b) → (c) の順にアクセスするが，これはコレクション階層内のアクセスパスが “/A/b” であるということである．このアクセスパスは，階層構造上のリソースそれぞれについてユニークである．従って，このアクセスパスをそのまま内部 ID にすれば，(a) と (b) にアクセスすることなく，(c) にアクセスできる．このようなデータ格納方式を採用すれば，コレクション階層の構造変化を伴なわないような操作についてはレスポンスタイムの短縮を見込める．

このアクセス方法に適した格納方式の条件としては，別のリソースの内部 ID と重複しないことの他に，アクセスパスからのみで導出できることが加わる．この条件のために，あるコレクションの移動や名前変更といった構造変化がそのコレクション配下全てのリソースに影響してしまう．しかし，Web コンテンツなどでは一般に，コレクションの移動や名前変更といったアクセスが読み取りアクセスに比べ発生頻度が低いため，変更コストよりアクセスコストを重視したこの方式の方がコレクション・トラバース法に比べ，多くの場合有利である．

アクセスコントロールについては，3.1.1 で述べた方法では実現できないが，対象リソースのアクセス権限の他に自分の親にあたるコレクションに至るまでのアクセス権限もプロパティとして記録することで実現できる．このとき，コレクションのアクセス権限の変更には大きなコストを要するが，コレクションの移動や名前変更同様にアクセス権限の変更も読み取りアクセスに比べ発生頻度が低いことから，アクセスパスマッピングの方がコレクション・トラバース法に比べ，多くの場合有利である．

ここでは内部 ID としてアクセスパスそのものを採用する．

3.2 並列分散ストレージへの導入

本節では，自律ディスクのような並列分散ストレージにおいて，コレクション・トラバース法とアクセスパス・マッピング法を実現する際の長短について考える．

ネットワークに接続してクラスタ化されている並列分散ストレージでは，リソースにアクセスするときに階層構造を降下していくことは通信コストの増加に繋がり，また階層構造におけるルートや上位コレクションへのアクセス集中を招く．Fat-Btree [5] のような分散ディレクトリ構造では，上位ノードへのアクセス集中に対して複製というアプローチで対処しているが，上位コレクションの複製というアプローチでは負荷均衡・容量均衡のためのデータ移動を行う際に構造を考慮した移動を行う必要が生じてしまうため，本稿では採用しない．そのため，

コレクション・トラバースではなく，アクセスパス・マッピング法を採用する．

図 3 は図 2 のような階層構造をアクセスパス・マッピング法で格納したときに，分散ディレクトリとして Fat-Btree を使用した場合のインデックス構造の例である．この構造が実際のデータの階層構造に依存していないことに注意されたい．

しかし，この方式には次のような問題が発生しうる．自律ディスクのようなデータ分散に値域分割戦略を用いている並列分散ストレージでは，アクセスパスをそのまま内部 ID とするとあるコレクション配下のリソースは近接した領域に配置される．これは，そのようなリソースは内部 ID の前部が同一となるためである．そして，あるコレクション配下のリソース（例えば，図 2 の網掛部）にアクセスが集中するような場合には，そのコレクションに対応する領域（図 3 の網掛部）のアクセス頻度が高くなる．値域分割戦略を採用する並列分散ストレージにおいてはアクセス頻度の高いディスクの境界に配置されているリソースを隣接ディスクに移動することにより負荷の均等化を行うため，負荷の集中するコレクションを格納するディスクの記憶領域使用量が極端に少なくなる可能性が出てくる．

そこで，この問題を考慮した格納方法である，ハッシュ値付きアクセスパス・マッピング法（hAPM）とサイズ考慮アクセスパス・マッピング法（sAPM）を提案する．hAPM は，アクセスパスのハッシュ値をアクセスパスの先頭に付加したものを内部 ID とする方式である．一方，sAPM はデータサイズの小さい非コレクションリソースをそのリソースを持つコレクションと同一または付近のディスクに格納するという方式である．

3.2.1 ハッシュ値付きアクセスパス・マッピング法（hAPM）

アクセスパス・マッピング法の改良として，アクセスパスのハッシュ値をアクセスパスの先頭に付加したものを内部 ID とする．

このようにすれば，同じコレクション内のリソースを広範囲のディスクに分散させることが可能となる．前述したような，あるコレクション配下にだけアクセスが集中するという状況では，アクセス偏り除去後の定常状態における各ディスクの記憶領域使用量の偏りを抑えることが期待できる（図 4；斜体部はハッシュ値）．採用したハッシュ関数やアクセスパターンによっては，コレクション内のリソースとしては分散しているにも関わらず，アクセス頻度の高いリソースが一つのディスクに集まってしまい，同様に記憶領域使用量の偏りが発生することも考えられる．しかし一般的には，同一コレクションのリソースにアクセスが集中する可能性が高いと判断できるので，本方式の効果が期待できる．

ただし，パス名からは各ディスクの記憶領域使用率が推測できないため全ディスクに均等に分散するとは限らない．また，サイズの小さいリソースに関しては，ディスク使用率均等化によって得られる効果が通信コストに比べ小さくなり，却って特に小さいリソースを多数含むようなコレクションに対するコピーや移動などの操作時にレスポンスが悪くなる可能性もある．

なお，ハッシュを利用しているため，一度格納されたリソースに対するハッシュ関数の変更・再適用は多大なコストがかかる．

3.2.2 サイズ考慮アクセスパス・マッピング法（sAPM）

ハッシュ値付きアクセスパス・マッピング法における問題点のうち，サイズの小さいリソースに関してレスポンスが悪くなる可能性が大きい点に関しては，工夫の余地がある．ハッシュ値付きアクセスパス・マッピング法に対して，データサイズの小さい非コレクションリソースについてはそのリソースを持つコレクションと同一または付近のディスクに格納する，すなわちそのコレクションと同一のハッシュ値を付加するという変更を施す．この格納方法を採用することにより，小さいリソースに対しては通信コストに起因するレスポンスタイムの悪化を回避しつつ，データ分散によるレスポンスタイム短縮の効果を得ることができる（図 5；斜体部はハッシュ値）．

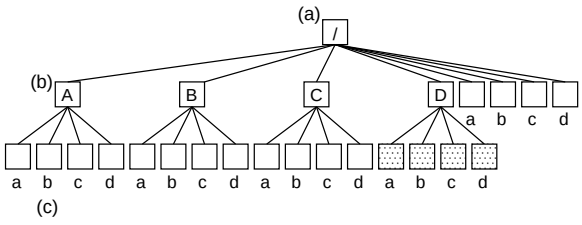

図 2　階層構造の例

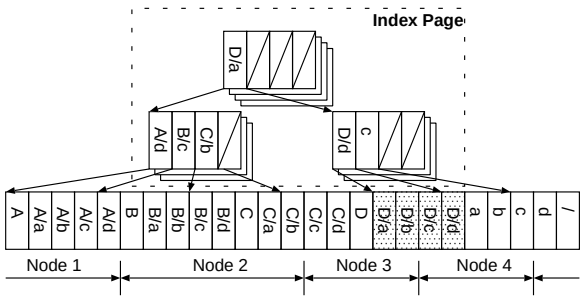

図 3　アクセスパス・マッピング法（APM）

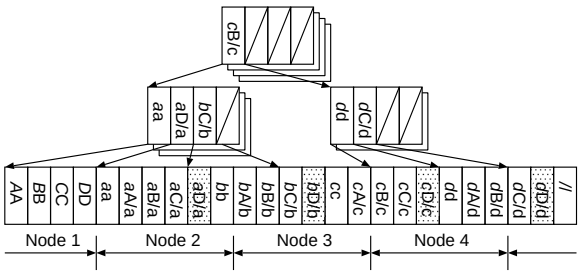

図 4　ハッシュ値付きアクセスパス・マッピング法（hAPM）

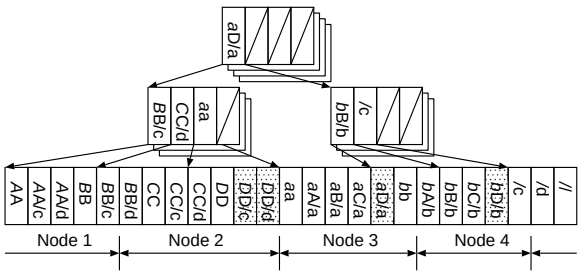

図 5　サイズ考慮アクセスパス・マッピング法（sAPM）

表 2　本稿における提案手法

	構造変化への耐性	アクセスコスト	アクセス偏りへの耐性 *	コレクション操作 † の効率
CT	○	大	×	○
APM	×	小	×	△
hAPM	×	小	○	×
sAPM	×	中	○	○

* : コレクション配下のリソースへアクセスが集中した時の容量均衡の取り易さ
† : コレクション配下の複数リソースへ一度にアクセスする操作

アクセス方式としては，非コレクションリソースに適用すべきハッシュ関数の引数がアクセスパスだけでは決定できないことから，アクセスパスからだけでは内部 ID を導出できなくなる．そのため，親コレクションに格納位置を問い合わせる必要が発生する．つまり，アクセスパス・マッピング法のアクセス方法は使用できない．

しかし，コレクションリソースに関してはアクセスパス・マッピング法同様のアクセス方法が使用できるため，以下のような手順を踏むことにより，非コレクションリソースに対してもコレクション・トラバース法より効率のよいアクセスができる．

（1） 当該リソースを持つコレクションにアクセスパス・マッピング法でアクセスする．

（2） 当該コレクションの持つエントリテーブルから要求するリソースの格納位置を取得する．

（3） 取得した格納位置へアクセスする．

サイズ考慮アクセスパス・マッピング法はデータ量の均衡が取れる可能性は高いが，あるディスクにアクセスが集中する可能性は大いにあるという問題がある．この問題に対しては，サイズが小さいリソースに関しては，ネットワーク中のプロキシにキャッシュとして残っている可能性が高く，そのキャッシュをクライアントが利用するといった運用を行うことでアクセス回数を減少させることが期待できる．

本節における議論をまとめると表 2 のようになる．

4. 評 価 実 験

提案手法の有効性を評価するために，我々が現在 PC 上に Java を用いて実装を行っている自律ディスクの試作システム上に，提案手法を実装し，評価実験を行った．

4.1 実験システム

実験システムは，マルチメディアコンテンツサーバ [4] の実装を元に，コンテンツを WebDAV のデータ構造に合わせ，かつ WebDAV のメソッドに対応するように追加実装を行ったものである．

以下に，その変更した部分の概要を示す．

a) データ構造に関する拡張

[4] のコンテンツデータと WebDAV リソースは形式的に対応する部分が多い．具体的には, [4] のメディアデータ表現と WebDAV における非コレクションリソースの実データは対応する．また, [4] のアノテーションやサムネイルと WebDAV のプロパティは対応している．しかし, [4] の実装ではアノテーション情報の種類が固定されているモデルを採用しているのに対し，WebDAV ではユーザ定義のプロパティも設定することができる．そこで，マルチメディアコンテンツサーバにおけるアノテーション情報にあたる部分は可変構造と変更した．

コレクションリソースについては，以下のように格納する．

- データ本体には，直下のリソース名とそのリソースの内部 ID へのマッピングを格納する．
- プロパティ中のリソースの種類を表すプロパティ（DAV:resourcetype プロパティ）に，コレクションである（値として，`<Collection/>`と書き込まれている）と記述する．

これは，前章で前提とした，コレクションは自身の直下のリソースへの格納位置を把握しているという設定を実現したものであり，また [3] で要求されていることに準じている．

なお，プロトタイプとして, [6] で提案された間接ディレクトリではなく，直接ディレクトリ方式のみを利用するもので実装した．この方式ではサイズの大きいマルチメディアデータを扱えないが，現在でも Web 上ではサイズの小さいテキストデータがかなりの割合を占めていること，そして本研究で特に注目している目的リソース発見までのオーバーヘッドの観測においてデータ本体の大小は本質的な問題ではないため，このような選択をした．

b) コレクション階層の探索に関する拡張

前章で述べたアクセス・格納方式を実現するためには，コレクション階層の探索を行う部分が必要であるためその部分を追加した．この拡張はルールの部分に施した．

4.2 評 価 実 験

提案手法の評価を行うために，複数の階層構造を用意し，その階層構造中のリソースに対する取得にかかる時間（レスポンスタイム）を計測する実験を行った．

計測対象のリソースは次の通りである．各構造とも非コレクションリソースはデータサイズが大きいもの，小さいものを同数配置してある．

（1） 構造偏りの無い，浅い階層のリソース（図 6 網掛部）
（2） 構造偏りの無い，深い階層のリソース（図 7 網掛部）
（3） 構造偏りの有る，浅い階層のリソース（図 8 網掛部①）
（4） 構造偏りの有る，深い階層のリソース（図 8 網掛部②）

このようなデータ構造を用意したのは，

- アクセスパスマッピングによる性能向上を示すため．
- ハッシュによるリソースの分散の効果を調べるため．

である．

アクセス・格納方法については，

（1） CT（内部 ID は挿入時刻の先頭にハッシュ値を付け，配置の偏りを抑えたもの）
（2） APM
（3） hAPM
（4） sAPM

の 4 種類を計測した．

表 3 に実験システムの性能諸元を示す. クライアントもクラスタの各ノードと同じ性能を持っており，クラスタ–クライアント間もギガビットイーサネットを用いて接続されている．

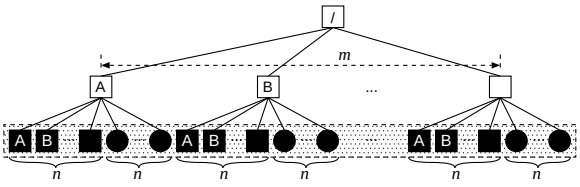

図 6　構造偏りの無い，浅い階層のリソース

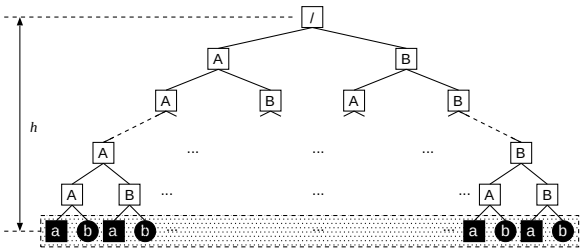

図 7　構造偏りの無い，深い階層のリソース

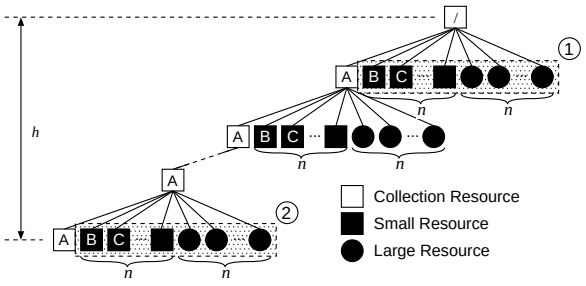

図 8　構造偏りの有るリソース

表 3　実 験 環 境

ノード数	6 台
CPU	Intel Pentium III 933 MHz
チップセット	Intel i815 (ATA100)
メモリ	PC133 SDRAM 32MB
HDD	Seagate Barracuda IV 20.4GB 7200rpm
OS	Linux 2.2.17, glibc 2.1.3
ネットワーク	1000BASE-SX, Switching Hub
Java	Sun JDK 1.4.1 ServerVM
HTTP サーバ	Apache Version 1.3.29
クライアント台数	1 台

測定条件は以下の通りである．

- クラスタ内の各ノードについて WebDAV の GET リクエストを発行する．リクエストは，大きいリソース，小さいリソース，それぞれ 1024 回ずつ発行する．
- 大きいリソースは 2KB，小さいリソースは 512 バイトである．
- sAPM の閾値は 1KB とした．
- 図 6 の $n = 16, m = 32$，図 7 の $h = 10$，図 8 の $n = 16$, $h = 32$ とした．
- どのノードにリクエストを行うかはランダムである．
- どのリソースにアクセスするかについては，これまでに記述した制約を除いて，ランダムである．
- アクセスパスの違いやデータ配置の偏りの違いを比較するために，負荷分散機構は停止してある．

4.3 結　　果

それぞれの構造における平均レスポンスタイムを図 9〜図 12 に示す．実際の計測において，Java の JIT（Just-In-Time）コンパイラが結果に影響を与えていることが判明したため，別に 256 リクエストを与えた後に計測を開始した．また，GC（Garbage Collection）によりレスポンスタイムの分布に大きなばらつきが生じたため，長いもののうち 10% を除外した平均レスポンスタイムを採用した．

4.4 考　　察

4.4.1　リソースサイズの違い

リソースサイズの違いによる差は小さかった．この原因として，512 バイトと 2KB では差が小さすぎて自律ディスクのレスポンスタイムの分散にかき消されてしまっている可能性がある．さらに大きいデータで実験を行う必要があるとともに，間接ディレクトリ方式の実装を行い，Java の影響が小さくなるような工夫を行う必要がある．

なお，sAPM の効果が顕れるのは，コレクション配下の多数のリソースに一度にアクセスが発生するような操作である．したがって，今後このような操作を行うメソッド実装し，そのメソッドのレスポンスタイムについて計測する必要がある．

4.4.2　階層構造の偏りと深さによる影響

図 9〜図 12 を比較すると，階層構造の偏り（図 9 と図 10，図 11 と図 12 をそれぞれ比較）よりも深さ（図 9 と図 11，図 10 と図 12 をそれぞれ比較）の方がレスポンスタイムに影響を与えていることがわかる．これは，アクセスパスをフラットなストレージにマッピングすることにより，階層構造の偏りを除去できたと判断できる．

一方で，階層構造が深くなるに従ってレスポンスタイムが長くなる傾向がある．特に，APM や CT での性能劣化が目立つ．この原因について考察する．CT はコレクション階層を降下するそのコストのために，レスポンスタイムが長くなる．しかし，コレクション階層の降下は行わない APM でも性能劣化が生じている．これは，負荷分散機構が動作していない条件下では，フラットなストレージにマッピングすることにより階層構造の偏りは除去できても，依然階層構造の制約がデータ配置に残ってしまうことに起因している，つまり，コレクションの配下のリソースは全て同じノードに配置されてしまうため，分散ディレクトリのインデックスが非常に大きくなってしまい，検索に時間がかかっているのではないかと推察する．

4.4.3　ハッシュによる分散の効果

ハッシュによりリソース分散を図った hAPM, sAPM に関しては，どの場合にも良好な結果を収めた．ハッシュによるリソース分散の効果は顕著に現れていると結論づけられる．

ハッシュを利用する 2 手法のうちでも，sAPM より hAPM の方が好成績を収めた．これは，sAPM ではアクセス時に親コレクションを経由しなければならないが，そのオーバーヘッドのためである．

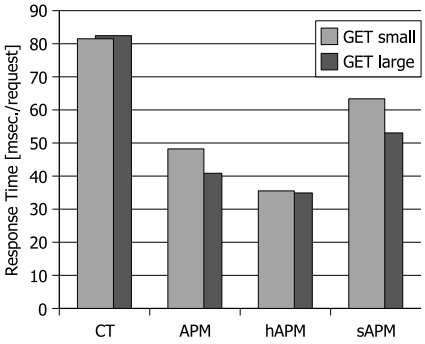

図 9　構造偏りの無い，浅い階層におけるリソースの平均レスポンスタイム

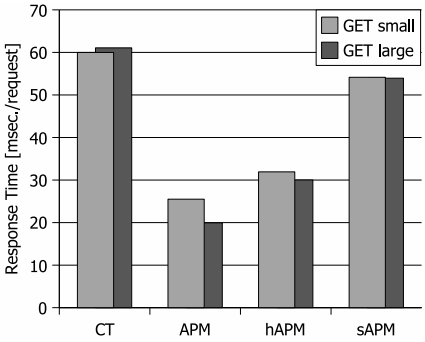

図 10　構造偏りの有る，浅い階層におけるリソースの平均レスポンスタイム

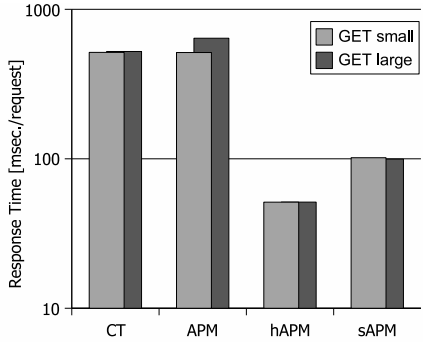

図 11　構造偏りの無い，深い階層におけるリソースの平均レスポンスタイム

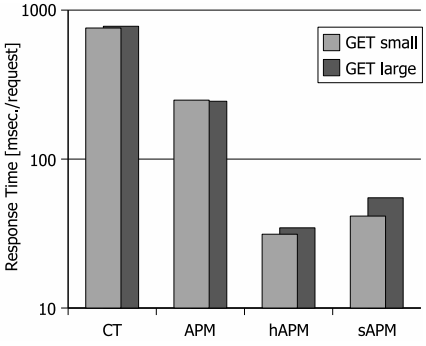

図 12　構造偏りの有る，深い階層におけるリソースの平均レスポンスタイム

5. 関 連 研 究

5.1 階層構造の提供

フラットな管理構造で階層構造を提供するということは多くのファイルシステムや OS で行われている，以下に，そのいくつかについて述べる．

MFS（Macintosh File System）は Apple Macintosh でかつて使用されていたフラットなファイルシステムである．MFS では，Finder というシェルプログラムの機能により仮想的に階層構造を提供していた．

また，ReiserFS [7]，JFS [8]，XFS [9] といった Linux 上のファイルシステムや Apple Macintosh で使用されている HFS, Novell NetWare 5 で使われている NSS, HP（旧 DEC）OpenVMS で使用されている Spiralog といったファイルシステムは，フラットな構造として管理を行っていながら，ユーザには階層構造を提供している．ただし，これらのファイルシステムでは，並列分散構成における負荷分散や容量分散については考慮されてはいない．

5.2 Apache HTTP Server での WebDAV 提供

HTTP のハンドリングを行う Apache HTTP Server [10] ではモジュールとして組み込むことで WebDAV を扱うことができる．

Apache 1.3 においては，mod_dav というモジュールを組み込むことで一般のファイルシステムを用いて WebDAV を提供することができる．このとき，ファイルシステム側で NFS を用いることで，並列分散構成をとることが可能である．しかし，この構成の場合，自律ディスクで提供している負荷分散機能や耐故障機能は提供できない．

一方，Apache 2.0 においては，mod_dav はプロトコルの処理のみを行うフロントエンドのモジュールと実際のデータストアへの操作を行うバックエンドのモジュールに分割された．mod_dav はフロントエンド部のモジュールである．そして，実際にファイルシステムに読み書きを行うのは mod_dav_fs というモジュールで行うように改められた．この変更で，関係データベース上にリソースを格納するようなことも実現可能となった．

Subversion [11] および Catacomb [12] はともに，Apache 2.0 の mod_dav のバックエンドモジュールとして利用できる WebDAV の実装である．これらはバックエンドとしてデータベース管理ソフトウェアを用いてフラットな管理を行っているが，WebDAV で前提としている階層構造をユーザに提供している．ただし，これらのモジュールでは特に並列分散構成に関して考慮されてはいない．

5.2.1 Subversion

Subversion [11] は，WebDAV に加え，DeltaV（RFC3253 [13]）のサブセットを実装したバージョン管理システムである．Subversion は CVS の置き換えのために設計・実装され，また DeltaV を定めるにあたって参照にされた実装でもある．Subversion は，Apache HTTP Server のような HTTP サーバと協調して動作させる場合と，Subversion 独自のサーバで動作させる場合の 2 通りの使用方法がある．

Subversion は，Barkeley DB [14] を利用している．この Barke-

ley DB はファイル型データベース管理ライブラリであり，ハッシュデータベース関数や，二分木やレコードといったデータ構造体を提供している．

5.2.2 Catacomb

Catacomb [12] は，関係データベース管理システムである MySQL [15] をストレージに利用した，Apache WebDAV モジュールのバックエンドの実装である．WebDAV，DeltaV に加え, [16] で規格の策定が進んでいるリソースの検索に関する拡張の DASL（DAV Searching and Locating）にも対応しており，また将来の実装計画にアクセス制御に関する拡張である ACL（Access Control List）も含まれている．

6. まとめと今後の課題

6.1 ま と め

本稿では，フラットなデータ構造上に階層構造を実現する手法として，コレクション階層をたどることでアクセスするコレクション・トラバース法（CT）とアクセスパスのみで導出可能な識別子をマッピングし直接アクセスするアクセスパス・マッピング法（APM）の 2 種類に大別し，並列分散ストレージには APM が適していることを示した．さらに，並列分散ストレージ向けに APM を改良したハッシュ値付きアクセスパスマッピング法（hAPM）および，サイズ考慮アクセスパス・マッピング法（sAPM）を提案した．そして，これらの手法を自律ディスクプロトタイプ上の WebDAV サーバとして実装し，性能の比較を行った．

その結果，多くの場合コレクション・トラバース法に比べてアクセスパス・マッピング法が優れていることを示した．さらにハッシュ値付きアクセスパスマッピング法およびサイズ考慮アクセスパス・マッピング法が性能を大きく改善することを示した．また，取得時の性能では，ハッシュ値付きアクセスパスマッピング法が最も優れていた．

6.2 今後の課題

まず，コレクション配下の複数のリソースへ一度にアクセスするような操作についての評価を行う必要がある．例えば，COPY, MOVE メソッドや Depth ヘッダを “1” あるいは “infinity” に設定したときの PROPFIND メソッドといった操作を，コレクションに対して行った場合における性能評価である．これらの操作は，サイズ考慮アクセスパス・マッピング法が有効であるような操作である．

次に，今回の実験ではサイズによる大きな違いは見られなかった．さらに大きなサイズのリソースで実験を行う必要がある．このためには，間接ディレクトリ方式での実装を行い，Java の影響が小さくなるような工夫を行う必要がある．

また，更なる評価として，さまざまなリソースに対する挿入，削除，移動，複製といった操作時のレスポンスタイムの測定も行いたい．

最後に，DeltaV [13] は WebDAV のバージョニング拡張であるが，まず分散編集の仕組みについて定めるために WebDAV が規定され，その後その枠組みを用いながらバージョニング機能について定められたという経緯を持つ．したがって，本稿で提案した方式はバージョニングにおいても応用可能である．さらに研究を進め，バージョニングに向いたアクセス・格納方式についても提案を行いたい．

謝　　辞

本研究の一部は，文部科学省科学研究費補助金特定領域研究 (15017233)，情報ストレージ研究推進機構 (SRC)，独立行政法人科学技術振興機構 CREST，および 21 世紀 COE プログラム「大規模知識資源の体系化と活用基盤構築」の助成により行なわれた．

文　　献

[1] Haruo Yokota. Autonomous Disks for Advanced Database Applications. In *Proc. of International Symposium on Database Applications in Non-Traditional Environments (DANTE'99)*, pp. 441–448, Nov. 1999.

[2] 花井知広 , 横田治夫. 自律ディスクを用いた Web サーバーの構成. 第 13 回データ工学ワークショップ論文集, DEWS2002 C3–4. 電子情報通信学会データ工学研究専門委員会, 2002.

[3] Y. Goland, E. Whitehead, A. Faizi, S. Carter, and D. Jensen. “HTTP Extensions for Distributed Authoring – WEBDAV”, Feb. 1999. RFC2518.

[4] 渡邊明嗣, 花井知広, 山口宗慶, 横田治夫. 自律ディスクを用いたマルチメディアコンテンツサーバ. 信学技報，電子情報通信学会, DE2002-86, DC2002-22, October 2002.

[5] Haruo Yokota, Yasuhiko Kanemasa, and Jun Miyazaki. Fat-Btree: An Update-Conscious Parallel Directory Structure. In *Proc. of the 15th Int'l Conf. on Data Engineering*, pp. 448–457, 1999.

[6] 花井知広 , 横田治夫. 自律ディスクを用いた Web サーバにおける負荷偏りの影響. 情処学会研究会報告, データベースシステム DBS-128-29. 情報処理学会, 2002.

[7] H. Reiser. ReiserFS. http://www.namesys.com/v4/v4.html, 2001.

[8] S. Beat. JFS overview. http://www-106.ibm.com/developerworks/linux/library/l-jfs.html, 2000.

[9] Inc. Silicon Graphics. XFS – File System. http://www.sgi.com/software/xfs/.

[10] The Apache Software Foundation. Apache HTTP Server. http://httpd.apache.org/.

[11] Subversion Project. Subversion. http://subversion.tigris.org/.

[12] Sung Kim. Catacomb. http://www.webdav.org/catacomb/.

[13] G. Clem, J. Amsden, T.Elison, C. Kaler, and J. Whitehead. “Versioning Extensions to WebDAV (Web Distributed Authoring and Versioning)”, Mar. 2002. RFC3253.

[14] Sleepycat Software. Berkeley DB. http://www.sleepycat.com/.

[15] MySQL AB. MySQL. http://www.mysql.com/.

[16] IETF WEBDAV Working Group. http://www.ics.uci.edu/pub/ietf/webdav/.