マルチメディア通信と分散処理　59－4
（１９９３．１．２９）

高速ＯＳＩディレクトリ情報ベース（ＤＩＢ）の実装

西山 智　　小花 貞夫　　堀内 浩規　　鈴木 健二

国際電信電話株式会社　研究所

ＯＳＩディレクトリは、メッセージ通信システム（ＭＨＳ）などＯＳＩ通信システムで必要な通信相手の接続番号や通信能力に関する情報を提供するために標準化が進められてきた。近年、ＯＳＩディレクトリを用いてユニバーサルパーソナル通信（ＵＰＴ）等のサービスで必要なネームサーバ機能を提供することが考えられているが、そのためには高速なＯＳＩディレクトリサービスが必要となる。筆者らはＯＳＩディレクトリのデータベースであるディレクトリ情報ベース（ＤＩＢ）を、ＤＩＢのデータモデルと操作を直接提供する専用データベース管理システム（ＤＢＭＳ）を用いて実装し、ＯＳＩディレクトリの高速化を図った。本稿ではこの高速なＤＩＢの実装について報告する。ＤＩＢ専用ＤＢＭＳでは、開発とＤＢＭＳ内部の最適化の容易性を考慮して、ツールキット方式の拡張可能ＤＢＭＳの構築技法を用いて階層的にモジュール化を行なった。ＤＩＢの従来の実装で高速化の妨げとなっていた１）名前解読処理、２）エントリ格納方式、３）フィルタ処理に対して、それぞれ１）木構造のアクセス手法に格納された識別名インデックス、２）高速なＡＳＮ．１符号化規則を格納形式とする直接クラスタリングによるエントリの格納、３）識別名インデックスと属性インデックスによるフィルタ処理、を実現することによって高速化を図った。ＵＮＩＸワークステーション上で評価した結果、１万件程度のエントリが格納されたＤＩＢで検索操作が12ミリ秒程度、更新操作が25ミリ秒程度で実行でき、ＵＰＴなどの高度な交換通信サービスのネームサーバとしても適用できることを示した。

Implementation of High-Performance OSI Directory Information Base (DIB)

Satoshi NISHIYAMA　　Sadao OBANA　　Hiroki HORIUCHI　　Kenji SUZUKI

KDD R & D Laboratories

2-1-15, Ohara, Kamifukuoka-shi, Saitama 356, JAPAN

OSI Directory has been standardized to provide information on telecommunication users and OSI communication systems such as MHS. Recently, it is proposed that OSI Directory can be applied as a name server for advanced telecommunication services such as Universal Personal Telecommunication (UPT). Since these are real-time switching services, the response time of the name server should be as small as possible. We consider that a key to high-performance OSI Directory used for such name server is a dedicated DBMS which directly supports the data model and operations of OSI Directory Information Base (DIB) as its data model and operations. In this paper, the implementation of high-performance DIB using the DBMS is discussed. The DBMS software is divided into modules hierarchially, based on the toolkit approach, one of the techniques for the extensible DBMSs, from the view point of easiness of design and further customization. To reduce the response time, the DBMS uses 1) the distinguished name (DN) index, stored by 'tree structure' access method, 2) direct clustering based on Light Weight Encording Rules of Abstract Syntax Notation (ASN).1, and 3) attribute indexes and DN index for filtering without accessing the stored entries. The evaluation result of the DIB on a UNIX workstation shows that it can execute Read operation and ModifyEntry operation in about 12 millisecond and 25 millisecond, respectively, when DIB stores about 10 thousand entries.

1. はじめに

ネットワークの高度化に伴い、ネットワークの効率的な利用や運用を支援する通信支援データベースが重要になってきた。代表的な通信支援データベースとして、通信相手の接続番号（アドレス）や通信能力等に関する情報を提供する開放型システム間相互接続（ＯＳＩ）ディレクトリ[1]が標準化されている。これまでＯＳＩディレクトリはメッセージ通信システム（ＭＨＳ）等の通信システム用のネームサーバや電子電話帳としての適用がなされてきたが、近年ＵＰＴ（ユニバーサルパーソナル通信）等の高度な交換通信サービスを実現するためのネームサーバとしての適用も検討されている[2]。

ＯＳＩディレクトリを電子電話帳やＭＨＳのネームサーバとして適用する場合、ディレクトリサービスの高速性はそれほど重要視されてこなかった。これに対し、ＵＰＴ等のサービスは即時交換型であり、呼接続処理時にネームサーバを使用するためネームサーバの応答速度が呼接続遅延に直接影響する。従って、ネームサーバとしてのＯＳＩディレクトリに対して数十ミリ秒でアクセスを可能とする高速なディレクトリサービスの提供が要求される。

これまでに幾つかのＯＳＩディレクトリの実装例が報告されている[3, 4, 5]が、これらは汎用的なディレクトリ機能の実現を主目的としたもので、高速性の実現を主目的としたものではなかった。ＯＳＩディレクトリで管理される情報はディレクトリ情報ベース（ＤＩＢ）と呼ばれる論理的なデータベースとして表現されるが、特にリレーショナル型やオブジェクト指向型等の汎用データベース管理システム（ＤＢＭＳ）を用いてＤＩＢを実現した従来の実装[3, 4]では、ＤＩＢのデータモデル、操作とＤＢＭＳのデータモデル、操作とのマッピングが必要となり、高速化に限界があった。

筆者らは、ＤＩＢのデータモデルと操作を直接提供する専用ＤＢＭＳを開発し、そのＤＢＭＳを用いてＤＩＢを実現することで、ＯＳＩディレクトリの高速化を図った。ここではその専用ＤＢＭＳを用いたＤＩＢの実装について報告する。

2. ＤＩＢの特徴

まずＯＳＩディレクトリの標準[1]で規定されるＤＩＢの特徴を述べる。

2.1 データモデル

ＤＩＢは、各オブジェクトの名前管理（明確に識別できる名前を与えること）のために、また、複数のＤＳＡにまたがってＤＩＢを分散管理できるようにするために、ディレクトリ情報木（ＤＩＴ）と呼ばれる木構造で論理的に表現される（図1）。ＤＩＴ木構造における木の節はエントリを、また木の枝はエントリ間の従属関係を表す。一つの節から出る木の枝にはそれぞれ他の枝と異なる名前（相対識別名：ＲＤＮ）が付与される。木構造の根から特定のエントリに至る相対識別名の並びは識別名（ＤＮ）と呼ばれ、そのエントリを一意に識別する。エントリは属性の並びからなり、また各属性は任意

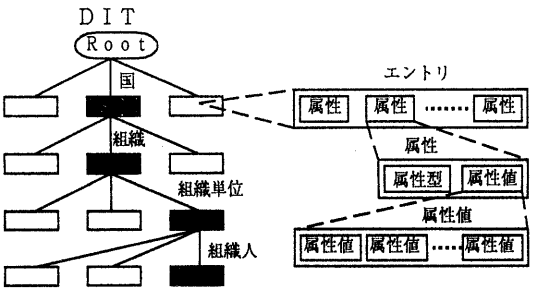

図 1: ディレクトリ情報木（ＤＩＴ）の構造

の数の属性値を持つことができる。属性値の型は抽象構文記法１（ＡＳＮ．１）[7]により規定される。

2.2 データ操作

ＤＩＴに対して、識別名からエントリの情報を読み出すRead操作、特定の条件に合致するエントリを検索し情報を読み出すSearch操作、エントリを追加するAddEntry操作等、９種類の操作が定義されている。Search操作では、検索範囲の部分木とフィルタと呼ばれる検索条件が指定できる。フィルタは各属性値に対する条件（値と比較条件）を任意に組み合わせたものである。

また上記のディレクトリ操作に対し、ＤＩＢは以下のような性質を持つものとして規定される。

- 一時的なデータの不整合を許容している。
- 全ての操作がエントリの単位で操作される。
- 更新操作は操作単位でのアトミック性を要求している。
- 検索系の操作に比較して更新操作の頻度は少ない。

3. 従来のＤＩＢ実現における問題点の明確化

これまでにいくつかＯＳＩディレクトリの実装例が報告されている[3, 4, 5]（表1）。これらはＤＩＢのモデルとは異なる汎用あるいは独自のＤＢＭＳやＯＳの提供するファイルシステムをデータ管理機能に用いてＤＩＢを実現しており、ＤＩＢのデータモデル、操作と使用したデータ管理機能のデータモデルや操作とのマッピングのために以下の点で高速化に問題となっている。

(1) 名前解読処理
任意の深さのＤＩＴに対して識別名から検索対象のエントリを得るための効率的な処理（名前解読処理）方法がない。例えば、[3]ではＤＩＴの上下関係をリレーショナル型ＤＢＭＳ（ＲＤＢ）の表形式にマッピングしておりＲＤＮを１段（或いは複数段）名前解読を行なう毎にＲＤＢ操作を実行する。また、[4]、[5]はＲＤＮを１段名前解読を行なう度にエントリ内容あるいはＵＮＩＸのディレクトリ内容を読む。

(2) エントリの格納
[3]では多数の属性値を含むエントリに対しては操作のマッピングが複雑となる。[4]、[5]はエントリ毎にＵＮＩＸファイルに格納しているため、エン

表 1: ＯＳＩディレクトリの実装例

	文献 [3]	文献 [4]	文献 [5]
データ管理機能	リレーショナル型ＤＢＭＳ	独自オブジェクト指向ＤＢＭＳ（ＵＮＩＸファイルシステムを格納に使用）	ＵＮＩＸファイルシステム
名前解読処理	ＤＩＴ専用の表にエントリ間の上下関係を格納し、複数のＲＤＮを一度に解析	各エントリ内に下位エントリのＲＤＮのリストを保持。各エントリ内容を読んで処理	ＤＩＴをＵＮＩＸの木構造管理にマッピング。ＵＮＩＸのディレクトリ情報を読んで実現
エントリ格納	クラス毎の表に格納。属性値が溢れた場合のみ属性毎の表に格納。独自符号化を使用。	ＵＮＩＸファイルに直接クラスタリング。ＡＳＮ．１基本符号化を使用。	ＵＮＩＸファイルに直接クラスタリング。独自符号化方式を使用。
フィルタ処理	検索範囲のエントリ内容を読んで処理	検索範囲のエントリ内容を読んで処理	（機能なし）

表 2: ＤＩＢ専用ＤＢＭＳの機能概要

拡張可能性（4.1節）	実現方式	階層的モジュール化
	モジュール構成	９レイヤ、１１モジュール
名前解読処理（4.2.1節）	実現方式	木構造アクセス手法を持つ識別名インデックスによる
エントリ格納（4.2.2節）	格納方式	エントリ単位の直接クラスタリング
	格納符号化形式	ＡＳＮ．１ＬＷＥＲ（注）
	格納の効率化	フラグメンテーションにより固定長化
	アクセス手法	Ｂ－木
フィルタ処理（4.2.3節）	実現方式	識別名インデックスと属性インデックスを使用
	属性インデックスの作成	ディレクトリのクラス共通で属性型毎に作成。値を正規化。Ｂ－木に格納
多重処理（4.3.1節）	実現方式	１プロセス多重処理
	多重処理方式	オブジェクト単位で多重処理
排他制御（4.3.2節）	制御方式	施錠（更新ロック）
	制御単位	ページ単位
障害回復（4.3.3節）	対象障害	アプリケーション障害、システム障害
	実現方式	ページ単位、後イメージ
バッファリング（4.3.4節）	実現方式	優先度付きＬＲＵ

（注） ＬＷＥＲ：ＯＳＩの抽象構文記法１（ＡＳＮ．１）のライトウェイト符号化規則

トリを読む度にＵＮＩＸのファイルオープン処理が必要となる。

(3) フィルタ処理方法

[3]、[4] は Search 操作を実現しているが、探索範囲の全てのエントリの内容を読み込んでフィルタ条件に合致するエントリを探索する。従って、操作に要する時間は結果の件数ではなく探索範囲に比例する。

4. ＤＩＢ専用ＤＢＭＳの実装

ＯＳＩディレクトリの高速化を図るためにはＤＩＢのデータモデル、操作と使用するデータ管理機能のデータモデル、操作とのマッピングをなくすことが必要であり、そのためにはＤＩＢのデータモデル、操作を直接提供するＤＩＢ専用のＤＢＭＳが必要となる[8, 9]。筆者らはこのＤＩＢ専用ＤＢＭＳをＵＮＩＸワークステーション上にＣ言語を用いて実装を行なった。以下ではその実装について示す。また表 2に機能の概要をまとめる。

4.1 拡張可能ＤＢＭＳの構築技法の採用

一般にデータモデル、操作あるいは内部の実現方法を拡張できるＤＢＭＳは拡張可能ＤＢＭＳと呼ばれる。拡張可能ＤＢＭＳには、拡張部分を解釈実行できる汎用的なＤＢＭＳを提供する方式[10]とＤＢＭＳのソフトウェアを部品化しアプリケーションに応じたＤＢＭＳを生成するツールキット方式[6, 11]の２種類がある。ツールキット方式では、ＤＢＭＳソフトウェアを機能的、階層的にモジュール化することでモジュール単位でＤＢＭＳの機能を拡張可能とする。ＯＳＩディレクトリ用ＤＢＭＳにおいてもこのようなモジュール化を行なうことによりモジュール単位で設計や実装が行なえ、開発の容易性が向上する。また、ＤＢＭＳ内部のアクセス手法やバッファリング等のアルゴリズムレベルでも最適化が可能となる。従ってツールキット方式の拡張可能ＤＢＭＳ構築技法を用いてＯＳＩディレクトリ用のＤＢＭＳソフトウェアを階層的にモジュール化した。

ここではＤＢＭＳの階層として、ファイル管理システムに相当する下位７レイヤ、インデックスを用いたフィルタ処理のためのインデックスレイヤと、ディレクトリ操作を実現するディレクトリサービスレイヤに分けた（図 2、表 3）。レイヤ毎にモジュールインタフェースを統一しモジュールの追加、変更を容易とした。

4.2 従来の実装における問題点の対処

3.章で明確化した３つの問題点：１）名前解読処理、２）エントリ格納、３）フィルタ処理の高速化に対する実装での対処内容について以下に示す。

4.2.1 名前解読処理

名前解読処理の高速化を図るために、図 3に示すように識別名に対してインデックスを生成し、インデックスレイヤでエントリ内容を読むことなしに識別名インデックスを用いて識別名から対象となるエントリを特定する。木の深さに制限を設けずにＤＩＴを表現でき、かつ検索範囲として指定された部分木に含まれるエントリを容易に特定できるように、識別名インデックスはＤＩＴの木構造に対応した木構造アクセス手法により格納

表 3: 各レイヤとモジュールの機能

レイヤ	機能	モジュール	モジュールの機能
ディレクトリサービス	ディレクトリ操作（Read 操作、Search 操作等）の実現	ＤＳＡ	ＡＳＮ．１ＬＷＥＲ符号化を使用
インデックス	フィルタ処理の実現	ＭＪ	識別名インデックスと属性インデックスの検索結果によりフィルタ処理を実現
ファイルマッピング	ファイル構造の変換	Ｆｒａｇ	固定長に分割
		Ｉｎｔ	無変換
アクセス手法	論理ブロックを用いて高速なアクセス手段を持つファイルを提供	木構造	識別名インデックスのためのＤＩＴに適したアクセス手法
		Ｂ－木	Ｂ－木を実現
論理ブロック	論理的なブロックを提供	ＯＣ	複数の物理ブロックを用いて無限長のソートされた論理ブロックを提供
物理ブロック	物理的なブロックへの格納方法を規定	ＯＦＵ	固定長のブロックにソートして格納
トランザクション	トランザクションのアトミック性を提供	ＡＩ	コミットまで主記憶上で後イメージを保持
バッファリング	バッファリングによりＩ／Ｏを減少	ＬＲＵ	優先度付きＬＲＵアルゴリズム
Ｉ／Ｏ	システム依存のＩ／Ｏ操作を共通化	ＵＮＩＸ	ＵＮＩＸのシステムコール

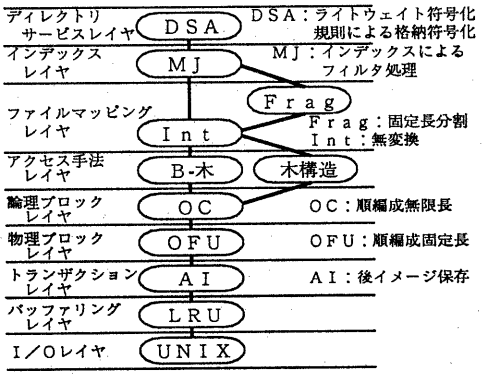

図 2: ＤＩＢ専用ＤＢＭＳのモジュール構成

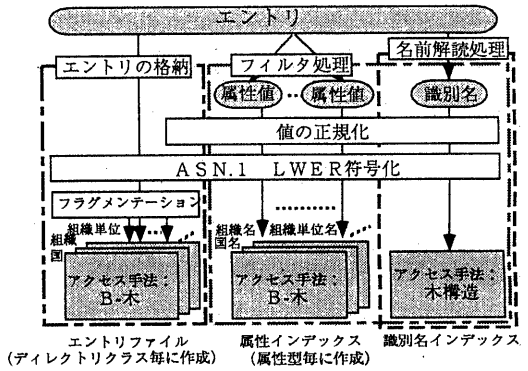

図 3: エントリとインデックスの格納

する。

この木構造アクセス手法を図4に示す。各エントリに対応するノードは、下位レイヤ（論理ブロックレイヤ）が提供するソートされた無限長の論理ブロックにより実現される。論理ブロックに格納されるレコードはキーとデータ内容からなり、キーとして下位エントリの相対識別名を用いる。また、データ内容をさらにポインタとフラグに分け、ポインタには通常下位エントリのノードへの論理ポインタが格納される。その際、格納領域の効率化のために、その下位エントリが葉である場合は、下位エントリに対応するノードを設けずにポインタに下位エントリの実体を指すオブジェクト識別子を格納した(例えば図4の節Z、A2)。フラグは、ポインタに格納される論理ポインタとオブジェクト識別子を区別する。

また、特殊なキーの値（ここでは０）を用いてそのエントリ自身の情報を表した。この場合ポインタにはエントリの実体を指すオブジェクト識別子が格納される。

4.2.2 エントリの格納

エントリ格納の概略を図3に示す。ＯＳＩディレクトリでは、エントリ単位の操作のみが定義されるため、エントリ単位でレコードとして格納する（直接クラスタリング)。ＯＳＩディレクトリでは属性の型としてＡＳＮ．１の任意の型が定義できることから、ＡＳＮ．１による符号化をクラスタリングの格納形式とした。さらに符号化の高速化を図るためにＡＳＮ．１の基本符号化規則（ＢＥＲ）ではなく、ライトウェイト符号化規則（ＬＷＥＲ）[12]を用いた。なお、符号化／復号処理部分についてはＣ言語の符号化／復号関数を生成するＬＷＥＲコンパイラ[13]を使用した。インデックスレイヤ以下のレイヤではデータの格納形式に依存しない構造とするために、この符号化はディレクトリサービスレイヤで行なわれる。さらに格納効率を向上するため、符号化したエントリをファイルマッピングレイヤが一定の大きさに分割（フラグメンテーション）する。分割されたエントリはさらに各エントリに対して内部で付与するオブジェクト識別子（ＯＩＤ）をキーとして、Ｂ－木を用いて格納される。

4.2.3 フィルタ処理

- フィルタを高速に実現するため図3に示すように、検索条件として指定可能な属性型毎に属性インデックスを生成する。Search 操作での検索範囲はＤＩＴ部分木で指定され、その範囲内には複数のディレクトリオブジェクトのクラスのエントリが含まれうる。従って属性インデックスはクラス毎ではなく、全てのクラスに共通に作成される。インデックスレイヤが特定の符号化形式に依存しないようにディレ

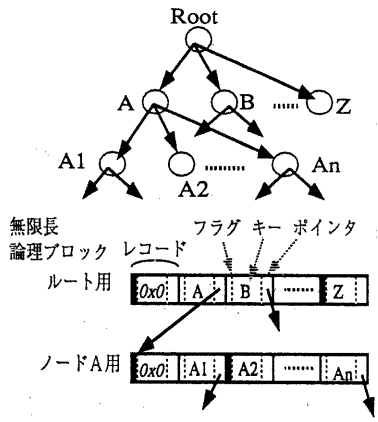

図 4: 木構造のアクセス手法

クトリサービスレイヤがインデックスを作成し、インデックスレイヤに通知する。なお属性インデックスはＢ－木をアクセス手法として用いる。

- Search 操作実行時はインデックスレイヤが作成された属性インデックスと識別名インデックスを用いて、実際のエントリ内容を読むことなしにフィルタによるエントリの絞り込みを行なう。
- 属性値の格納方式についてもＡＳＮ．１のＬＷＥＲ符号化を用いることとした。ＯＳＩディレクトリでは各属性に対して意味を考慮した比較規則（例えば大文字小文字を無視とか、電話番号のハイフンを無視する等）を定めている。検索時に単なるバイト列の比較が行なえるように、属性インデックスに対しては符号化結果が一意となるようディレクトリサービスレイヤが正規化を行なうこととした。例えば、電話番号に対しては含まれるハイフンを除去する処理を行なう。

属性インデックスを用いた検索例を以下に示す。図５のＤＩＴに対して {国:JP、組織:B} 以下の全部分木を検索範囲としてフィルタ（通称＝ａ）を持つ Search 操作を発行したとする。識別名インデックスから該当エントリへの論理ポインタの集合ＯＩＤ={3、6、7} が得られ、また通称の属性インデックスからＯＩＤ={4、6} が得られる。両方の集合の積をとることで検索結果のエントリへの論理ポインタの集合（この場合ＯＩＤ={6}）が得られる。

4.3 多重処理、排他制御、障害回復、バッファリング

以下ではＤＢＭＳを実現する場合に必要な多重処理、排他制御、障害回復、バッファリングについて述べる。

4.3.1 多重処理

一般に多重処理の実現方法として利用者毎に生成される複数プロセスでＤＢＭＳを構成し並行処理する方法と、１プロセスで多重処理する方法の２種類が考えられる。前者では利用者毎にサーバプロセスが生成され多数の利用者がアクセスするデータベースでは資源面で不利であるため、後者の１プロセス内で多重処理する方

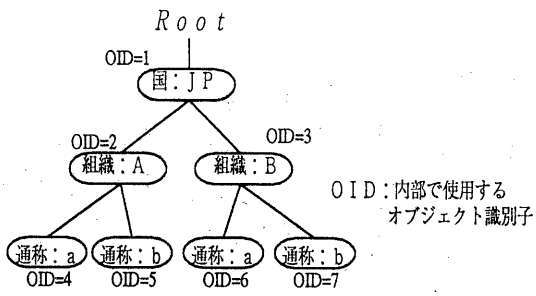

図 5: ＤＩＴの例

法を用いることとした。ディレクトリサービスレイヤがエントリ単位の処理を行なう度に複数のディレクトリ利用者の要求を切替えて多重処理を行なう。

4.3.2 排他制御

2.2節で述べたようにＯＳＩディレクトリは一時的なデータの一貫性のない状態を許容している。この点を考慮し高速化を図るために、更新ロックにより排他制御を行なうこととした。ロックの単位はページ単位としアクセス手法レイヤで実施する。4.2.3節で述べたように各属性に対するインデックスを全てのオブジェクトクラスに共通に作成するため、更新時にファイル全体にロックを行なう通常のロック手法を用いた場合、その属性型を持つエントリ全てに対する更新操作がロック待ちを引き起こす。従って、Ｂ－木については更新で影響が及ぶ部分木のみをロックする手法[14]を用いることとした。

ロック待ちは操作単位とし、ディレクトリサービスレイヤが多重処理の一環として行なう。デッドロックが発生した場合は、後発の操作を中止してデッドロック解除を行なう。

4.3.3 障害回復

障害回復の対象をアプリケーション（ディレクトリ利用者エージェント：ＤＵＡ）障害並びにシステム障害とし、トランザクションレイヤでページ単位の障害回復処理を実施する。ＯＳＩディレクトリではトランザクション中で一度に更新されるエントリは少ないため、主記憶上に変更後のページのイメージをコミットまで保持する方法を用いることとした。

4.3.4 バッファリング

バッファリングはバッファリングレイヤでページ単位で実施することとした。バッファリングアルゴリズムとして、優先度「高」のページが一定割合以内の場合、優先度「低」のページから掃き出しの犠牲者が選択されるという高低２段階の優先度付きＬＲＵを使用した。アクセス手法レイヤがこの優先度の指定を行なうこととし、アクセスのランダム性が高いＢ－木の葉ノードは優先度が「低」で、他は全て優先度が「高」としてバッファリングさせた。

5. ＤＩＢとしての評価

実装した専用ＤＢＭＳを用いて評価用ＤＩＢを構築し、性能評価を行なった。

表 4: 測定条件

測定環境	Sun SPARC 670MP	
測定内容	利用者プロセスでの操作をＤＢＭＳに送信してから結果を受信するまでの応答時間	
格納内容	国	属性：クラス、国名の２件９バイト
	組織	属性：クラス、組織名の２件 23 バイト
	組織単位	属性：組織単位名等５件約 60 バイト
	組織人	属性：通称等６件約 70 バイト
操作内容	Read	指定エントリの全属性の読み出し
	Search	指定エントリ以下の全部分木を範囲とし一致条件のフィルタ１件。結果は１件

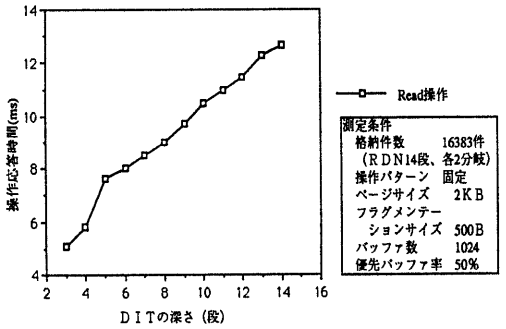

図 6: ＤＩＴの深さと検索操作応答時間

5.1 検索性能評価

3.章で挙げた１）名前解読処理、２）エントリ格納、３）フィルタ処理の３項目の問題点に対する（検索）処理性能の評価を行なった。表 4 に評価の測定条件を示す。

5.1.1 名前解読処理

名前解読処理の性能を見るために２分木、ＲＤＮの深さ１４段、エントリ件数 $2^{14}-1(=16,383$ 件) 件のＤＩＴに対して指定する識別名のＲＤＮ数を変化させて Read 操作の応答時間を測定した結果を図 6 に示す。ここではバッファのヒット率の変化を排除するために同じ操作を繰り返し実行した。グラフの傾きからＲＤＮ１段当りに必要な名前解読処理時間が約 0.6 ミリ秒であることがわかる。

5.1.2 エントリ格納

エントリ格納件数の変化に伴うエントリ検索性能を見るために、ＲＤＮの深さ４段（国、組織、組織単位、組織人）、２段目以下が各段で均等に分岐するＤＩＴに対して、各段での分岐数を変化させることでエントリ格納件数を変化させ、ランダムに選んだ組織人クラスの葉エントリに対する Read 操作、葉の１段上の組織単位エントリに対して表 4 で示す Search 操作を実行させた。これらの操作の応答時間を図 7 に示す。

5.1.3 フィルタ処理

フィルタ処理性能を見るために検索範囲を変化させて Search 操作の応答速度を測定した。アクセス手法の評価で用いた２分木、ＲＤＮの深さ１４段、エントリ件

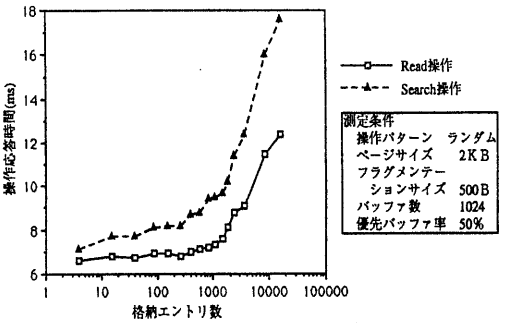

図 7: エントリ格納件数と検索操作応答時間

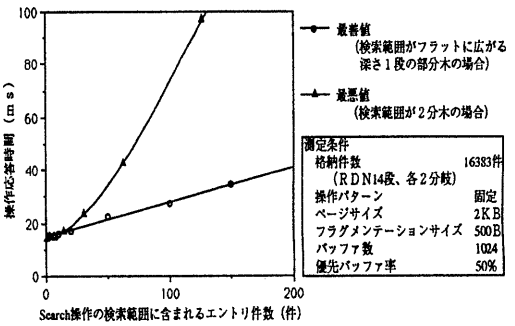

図 8: フィルタ検索範囲と検索操作応答時間

数 $2^{14}-1(=16,383$ 件) 件のＤＩＴを用いて、１段の部分木の下にフラットに分岐数が増えていくために検索範囲が広がる場合と、２分木で部分木の深さが増えていくことにより検索範囲が広がる場合を測定した結果を図 8 に示す。ここでも、バッファのヒット率の変化を排除するために同じ操作を繰り返し実行した。グラフの傾きから、検索範囲が１件広がることによる遅延は、１段にフラットに多数のエントリが存在する場合は 0.15 ミリ秒程度、また２分木の場合は 0.8 ミリ秒程度であることが分かる。

5.2 更新性能評価

一般にＯＳＩディレクトリは更新の少ないアクセスを想定しているが、高度な交換通信サービスのネームサーバとして使用される場合、更新が比較的多くなる場合も想定される。例えばＵＰＴでは各加入者に実際に現在通信できる端末のアクセス番号をディレクトリに格納するため、加入者の網の移動に伴いアクセス番号の更新が発生する。従って、更新性能の評価も行なうこととした。評価する更新操作の選択に当たっては、例えばＵＰＴネームサーバとして適用される場合更新はオブジェクトの追加削除ではなくオブジェクト内の属性値の更新に対応付けられる[2]ため、オブジェクト内の属性値の更新（ModifyEntry）操作を評価対象とした。エントリ格納

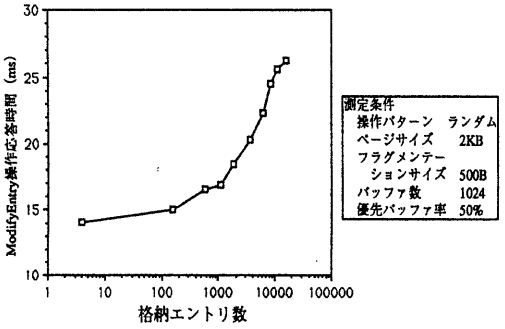

図 9: エントリ格納件数と更新操作応答時間

件数の変化に伴うエントリ検索性能評価で使用したＤＩＴを用いて、ランダムに選んだ組織人クラスの葉エントリの電話番号の更新を行なった場合の応答時間を図9に示す。

6. 考察

6.1 性能評価結果について

(1) 名前解読処理性能

評価結果からＲＤＮ１段当たりの名前解読処理時間は0.6ミリ秒程度である。現実的にはＤＩＴの深さは多くとも１０段程度と考えられ、最大でも名前解読処理時間は６ミリ秒程度に抑えられる。高速な名前解読処理の実現ができたと考えられる。

(2) エントリ格納性能

エントリの読み出し速度については、１万件程度格納した状態で実際の操作応答時間（12ミリ秒）から名前解読処理の時間（0.6ミリ秒/段×４段）を除くと、読み出しから符号化まで、10ミリ秒程度であり、十分高速化ができた。格納件数を変化させた場合の応答速度については、格納件数を M とするとＢ－木の特性である $O(logM)$ よりも悪い傾向を示している。評価内容がランダムなエントリに対する検索というＬＲＵアルゴリズムに最も悪い条件であり、エントリ格納件数の増加にしたがってバッファのヒット率が低下しているためと考えられる。バッファ数を増加してヒット率を１００％としたところ、応答時間が $O(logM)$ 近くまで改善された。従って、バッファリングアルゴリズムを改良することでさらに処理性能の向上が期待できる。

(3) フィルタ処理性能

識別名用インデックスに用いた木構造アクセス手法から、検索範囲１件当たりの遅延は１段にフラットに多数のエントリが存在する場合が最良の条件、２分木の場合が最悪の条件であると考えられる。評価結果から、フィルタ処理の遅延は検索範囲に比例して増加するものの、検索範囲１件当たりの応答時間の増加は最良の条件で0.15ミリ秒、最悪値で0.8ミリ秒程度と、検索範囲の全てのエントリの内容を実際に読む場合と比較して十分小さく、高速なフィルタ処理が実現できた。

(4) 更新処理性能

図9から１万件格納時のエントリの更新操作応答時間は25ミリ秒程度であり、高速な更新処理が実現できた。更新操作の応答時間の特性も検索操作の場合と同様に、エントリ格納件数に対してＢ－木の特性よりも悪い傾向を示す。検索操作の場合と同様にバッファリングアルゴリズムの改良で応答時間の特性が改善できると予想される。

6.2 ＵＰＴネームサーバとしての適用性について

6.1節で考察したように、性能評価の結果から本ＤＩＢの高速性が実証できた。ここでは、高度な交換通信サービスのネームサーバとしての適用性について、特に更新操作が多く処理能力が低くなるＵＰＴに適用した場合を例に考察する。仮にＵＰＴネームサーバに適用されたＯＳＩディレクトリに対する操作要求が、更新（ModifyEntry）操作と検索（Read）操作が１：１であったとする。図7と図9から１万件格納でのエントリの読み出し速度はそれぞれ12ミリ秒程度、25ミリ秒程度であるため、１操作当たりの平均処理時間は18ミリ秒程度となる。

ＵＰＴを含む高度な通信サービスであるインテリジェントネットワーク（ＩＮ）の規格の１つであるＡＩＮ[15]では、ＵＰＴでの番号変換など交換サービス制御を行なうＳＣＰ（サービス制御ポイント）の応答時間として平均150ms、最悪180msと定めている。今回実現したＤＩＢの応答時間はＳＣＰの応答時間の１割程度であるため、ＤＩＢをネームサーバとして用いても、ＳＣＰの応答時間の遅延にあまり影響する値ではない。このことから本ＤＩＢのＵＰＴ等の高度サービスへの適用性が実証できたと考える。なおこの値は約２０万操作／時間程度の処理能力に相当する。一般に大規模交換機の最繁時の１時間当りの呼処理能力が１００万呼程度であるから、ワークステーションを５、６台で十分大型交換機をサポートできると考えられる。

6.3 拡張可能性を用いたＯＳＩディレクトリの最適化

ＯＳＩディレクトリを例えばＵＰＴ等の特定のアプリケーションで使用する場合、そのアプリケーションの性質に応じた最適化を行なうことが考えられる。拡張可能性を持たない場合、予めパラメータ化された項目（例えばページサイズ、バッファ数）に対する最適化しか行なえないが、拡張可能性を持つことでアルゴリズムの変更等より広い範囲にわたる最適化が可能となる。

例えば、アプリケーションの性質により、エントリの大きさが固定的であればフラグメンテーションを行なわない、格納効率を向上させるため格納データを圧縮するアルゴリズムを追加する、バッファリングのアルゴリズムを変更するといった最適化が、他のレイヤやモジュールに影響を与えることなくモジュール振り分けの変更やモジュール追加で実現できる。

実験的に、バッファリングアルゴリズムをこれまでの

優先度付きＬＲＵから優先度なしのＬＲＵに置換した所、全ソフトウェアの内３００ステップ程のモジュール１箇所の変更で実現できることを確認した。

6.4 実現したＤＩＢ用ＤＢＭＳの汎用性について

ＯＳＩディレクトリでは、アプリケーション毎に固有のオブジェクトクラスや属性型が定義される。ＤＩＢ用ＤＢＭＳではオブジェクトクラス／属性型が追加、変更された場合、ディレクトリサービスレイヤ中に存在するオブジェクトクラスや属性型に関するデータを変更し、さらにＡＳＮ．１で記述されたオブジェクトクラス／属性型の構造定義をＡＳＮ．１のＬＷＥＲコンパイラにかけて再コンパイルすることで追加、変更に対処できる。従って、従来のＯＳＩディレクトリの実装例が実現していた静的なディレクトリスキーマへの対応機能を本ＤＢＭＳも具備している。一般にＯＳＩディレクトリに必要なオブジェクトクラスや属性型の種類は使用されるアプリケーションに依存するのであまり動的に変更されることは少なく、静的なスキーマ変更機能で十分であると考えられる。

7. むすび

本稿では、高速なＯＳＩディレクトリを実現するために、ＤＩＢのデータモデル、操作を直接提供する専用ＤＢＭＳを用いたＤＩＢの実装について報告した。

ＤＩＢ用ＤＢＭＳは、モジュール化によって設計、実装が容易になることと、拡張可能性によりＤＢＭＳ内部の最適化が容易になることを考慮し、ツールキット方式による拡張可能ＤＢＭＳの構築技法を用いて内部構造を９レイヤに階層的にモジュール化した。従来のＤＩＢの実装例で高速化に問題となっていた１）名前解読処理方法については、ＤＩＴの表現に適した木構造アクセス手法を用いる識別名インデックスを用いることで、２）エントリ格納方法については、エントリ単位に直接クラスタリングを行ない、格納形式としてＡＳＮ．１のＬＷＥＲ符号化を用いることで、３）フィルタ処理方法については、フィルタに指定可能な属性型毎に属性インデックスを設けて、エントリ内容を直接読むことなしに識別名インデックスと属性インデックスからフィルタ処理を行なうことで、高速化を図った。さらにＤＢＭＳとしての多重処理方式、排他制御方式、障害回復方式、バッファリング処理などにもＤＩＢの性質を考慮した実装を行ない高速化を図った。

評価の結果、１万件程度のエントリが格納されたＤＩＢで検索（Read）操作が 12 ミリ秒程度、更新（ModifyEntry）操作が 25 ミリ秒程度で実行でき、ＵＰＴなどの高速性を必要とするアプリケーションのネームサーバとしても適用できることを示した。最後に、日頃御指導頂くＫＤＤ研究所 小野所長、浦野次長、ならびに御討論頂いた同研究所通信網支援ソフトウェアグループ 浅見リーダに感謝します。

参考文献

[1] ISO/IEC 9594 1-8: Information Processing Systems – Open Systems Interconnection – The Directory (1988).

[2] 小花 他: ユニバーサルパーソナル通信（ＵＰＴ）へのＯＳＩディレクトリの適用と評価, 電子情報通信学会論文誌 Vol. J74-B-1, pp. 959-970 (1991).

[3] 小花 他: リレーショナルアプローチによるＯＳＩディレクトリのＤＩＢ（ディレクトリ情報ベース）の実装と評価, 情報処理学会論文誌 Vol.32, No.11, pp.1488-1497, (1991).

[4] 中川路 他: ＯＳＩディレクトリシステムにおけるＤＩＢ（ディレクトリ情報ベース）のオブジェクト指向による実現, 情報処理学会論文誌 Vol.32, No.3, pp.304-313 (1991).

[5] Robbins, C.J., et. al.: The ISO Developement Environment: User's Manual Vol.5:QUIPU, (1989).

[6] Batory, D.S. et al.: GENESIS: A Reconfigurable Database Management System, University of Texas at Austin, Tech. Report TR-B6-07, (1986).

[7] ISO 8824,8825: Information Processing System – Open Systems Interconnection – Specification of Abstract Syntax Notation One (ASN.1) / Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1), (1987).

[8] 西山 他: 拡張可能ＤＢアプローチに基づく通信支援用ＤＢＭＳの構築に関する提案, 第４４回情報処理学会全国大会論文集 3L-4, (1992).

[9] 西山 他: 高速ＯＳＩディレクトリ用ＤＢＭＳの設計, 第４５回情報処理学会全国大会論文集 3R-09, (1992).

[10] Stonebraker, M., and Rowe, L.: The Design of POSTGRES, Proceedings of 1986 ACM SIGMOD Conference on Management of Data, (1986).

[11] Carey, M.J., et. al.: The Architecture of the EXODUS Extensible DBMS, Proceedings of Object-Oriented Database Workshop, pp.52-65, (1986).

[12] ISO/IEC JTC1/SC21 N6131: Working Draft for Light Weight Encording Rules, (1991).

[13] 堀内 他: ＯＳＩ応用層プロトコル用ＡＳＮ．１ライトウェイト符号化規則のための符合化／復号処理系の実装と評価, 情報処理学会マルチメディア通信と分散処理研究会資料 DPS-55-2, pp.9-16, (1992).

[14] Bayer, R., Schkolnik, M.: Concurrency of Operations on B-Trees, Acta Informatica, (1977).

[15] Bellcore: Advanced Intelligent Network (AIN) Release 1: Service Logic Program Framework Generic Requirments, Issue 1, (1991).