データベースシステム 95－7
（1993．9．9）

OSI管理情報ベース（MIB）用データベースの設計と実装

西山 智　　堀内 浩規　　横田 英俊　　小花 貞夫　　鈴木 健二

国際電信電話株式会社　研究所

網管理のための基盤アーキテクチャであるOSI管理では、管理オブジェクト（MO）の集合は論理的に管理情報ベース（MIB）と呼ばれる。OSI管理に基づく網管理システムを実現する場合、検索の効率化等の理由によりMOの情報を管理システムや被管理システム上のデータベース（MIB用データベース）に格納する必要が生じる。これまで、汎用のDBMSを用いてMIB用データベースを実現する例が報告されているが、MIBのデータモデル・操作と使用するDBMSのモデル・操作のマッピングのために高速化が困難であった。筆者らは、拡張可能データベースの構築技法に基づき、MIBのデータモデルと操作をそれ自体のデータモデルと操作とする専用のデータベースASSIST/Mを設計、実装した。評価の結果、1つのMOに対するM-GET操作、M-SET操作がそれぞれ約20ミリ秒、約40ミリ秒で実現でき、汎用DBMSを用いて実現した場合と比較して2～10倍高速であった。また、既に開発したOSIディレクトリ用DBMSの拡張可能性を利用して、一部のモジュールを変更することで容易に専用データベースが実現できた。

Design and Implementation of Database Software for OSI Management Information Base (MIB)

Satoshi NISHIYAMA　　Hiroki HORIUCHI　　Hidetoshi YOKOTA
Sadao OBANA　　Kenji SUZUKI

KDD R & D Laboratories
2-1-15, Ohara, Kamifukuoka-shi, Saitama 356, JAPAN

This paper discusses on the design and the implementation of ASSIST/M: database software for OSI Management Information Base (MIB).

The collection of managed objects (MO) is logically called as MIB in the OSI management. A database which provides the data model and operations of MIB to store the data from MOs is the key to implement a network management system based on OSI management, This paper proposes dedicated database software which directly supports the data model and operations for MIB. It is designed as layered software based on the toolkit approach, one of the extensible database techniques, and is easily implemented by replacing modules of 3 layers of another dedicated DBMS for OSI Directory with the modules specific for MIB. The evaluation result on a UNIX workstation shows that it can execute a M-GET operation and a M-SET operation for a single MO within 20 millisecond and 40 millisecond, respectively.

1. はじめに

網管理のための基盤アーキテクチャであるOSI管理[1]では管理オブジェクト（MO:Managed Object）の集合は論理的に管理情報ベース（MIB:Management Information Base）と呼ばれる。MOの情報はMOに対応する実管理対象に存在する場合もあるが、障害管理等のアプリケーションからの操作効率を考慮して、通常MOの情報を管理システム（マネージャ）や被管理システム（エージェント）内のデータベース（MIB用データベース）に格納する。MIB用データベースは格納時におけるMOの情報欠落を防いだり、MOに対する場合と統一した操作が行なえるように、アプリケーションからはMIB用データベースをMIBのデータモデルで操作できることが必要である[2]。これまで、このMIB用データベースを汎用のデータベース管理システム（DBMS）で実現した例が報告されている[3, 4, 5, 6]が、MIBのモデルを実現するために、使用するDBMSとMIBのモデル・操作間のマッピングが必要となり、高速化が困難であった。そこで、本稿ではMIBのデータモデルと操作を直接扱える専用のデータベースを構築することを提案し、このMIB用データベースASSIST/M (Advanced Supervisary Supporting System for Integrated Services of Telecommunication/MIB) の設計と実装について報告する。

2. OSI管理情報ベース（MIB）

2.1 モデル

OSI管理では全ての管理対象は管理オブジェクト(MO)としてモデル化される。MOは管理対象間の包含関係によって木構造（包含木）に関係付けられ、MOの集合は木構造モデルを持つ仮想的なデータベースとみなすことができる。このデータベースは管理情報ベース（MIB）と呼ばれる。

包含木では、MOの全体は仮想的に存在するルートに包含されている。包含木における節はMOを表し、枝は同一のMOに包含されるMOを識別するための相対識別名(RDN)を表す。ルートから包含関係を示す枝をたどることにより個々のMOは識別される。途中の枝に対応するRDNの並びは識別名(DN)と呼ばれる。

各MOはそのMOの情報として複数の属性を持つことができる。また、属性には複数の属性値を持つものもある。

2.2 スキーマ定義

MIBのデータスキーマとなる管理オブジェクトや属性等の定義は7種類のテンプレートと呼ばれる抽象構文規則1（ASN.1）[8]のマクロにより定義される。表1にこれらのテンプレートを示す。

表1: MIBのテンプレート

テンプレート名	定義する内容
管理オブジェクトクラス	MOのクラスを定義する。オブジェクトクラスの名前、クラスの継承元、必須及びオプショナルなパッケージを記述する
パッケージ	MOの振舞いや含まれる属性を定義する。パッケージの名前、含まれる属性と許される操作、初期値とデフォルト値、含まれる属性グループ、動作、通知を記述する。
属性	属性を定義する。属性名、ASN.1定義あるいは他の属性への参照による属性の型、値の比較規則を記述する。
属性グループ	属性をまとめたグループを定義する。グループ名、含まれる属性を記述する。
名前結合	MOのクラス間の上位/下位関係を記述する。名前結合名、上位MOクラス、下位MOクラス及びRDNに使用される属性を記述する。
動作	M-ACTION操作による動作の内容を定義する。動作の名前、ASN.1の型定義によりM-ACTION操作で指定される情報とその応答の型を記述する。
通知	M-EVENT-REPORT操作による事象報告の内容を定義する。通知名とASN.1の型定義によりM-EVENT-REPORT操作で指定される情報とその応答の型を記述する。

表2: CMIS操作

M-GET	条件に合致するMOの情報を取得する。
M-CANCEL-GET	M-GET操作を中止する。
M-SET	条件に合致するMOの情報を更新する。
M-CREATE	条件に合致するMOを作成する。
M-DELETE	条件に合致するMOを削除する。
M-EVENT-REPORT	特定の事象が発生したことをCMIS利用者に通知する。
M-ACTION	条件に合致するMOに何らかの手続きを実行させる。

2.3 操作

MIBに対する操作は、OSIの応用層のサービスである共通管理情報サービス（CMIS）上の操作として表2に示す7種類が定義されている。

これらの操作の内、M-EVENT-REPORTはMIB側から自律的にMIBの利用者に通知されるものである。また、M-ACTION操作はMOに対して予め定めた操作を実行させるものであるが、その操作内容については勧告/標準の範囲外となっている。

また、M-CANCEL-GET、M-EVENT-REPORTを除く全ての操作で、包含木の部分木による操作範囲の指定と、フィルタと呼ばれる操作条件の指定ができる。フィルタは属性に対する条件の論理演算による任意の組合

せであり、条件は属性、値、比較規則からなる。比較規則としては一致、部分一致、複数の属性値に対する集合演算等10種類が定義されている。

3. MIB専用データベースの提案

3.1 専用データベースによる実現

OSI管理ではMOの情報はMOに対応する実管理対象に存在するが、MIBに対する操作効率を考慮して通常MOの情報をマネージャやエージェント内のデータベース（MIB用データベース）に複写する手法が一般的に行なわれる。

このデータベースは、複写時のMOの情報の欠落を防いだり、実際のMOと同一の形式で操作できるようにするため、MIBのデータモデルに従ってMOの情報を格納することが望ましい。これまで、MIB用データベースを汎用のデータベース管理システム（DBMS）を用いて実現した例（RDBMS[3, 4]、OODB[5, 6]等）が報告されているが、MIBのモデルを実現するために、使用するDBMSとMIBのモデル・操作間のマッピングが必要となり高速なMIBの実現が困難であった。特に汎用のDBMSを用いた場合、CMIS操作で指定される操作範囲とフィルタを使用するDBMSの操作に効率的にマッピングすることは難しい。

筆者らはMIBやOSIディレクトリ情報ベース（DIB）のような木構造モデルを持つデータベースの高速な実現方法として、専用のデータベース機能を拡張可能データベース構築技法に基づき実現する手法をこれまでに示した[7]。そこで、この手法を用いてMIBのデータモデル・操作を直接提供するMIB用データベースを構築することで、データモデル・操作間のマッピングを排除し、高速なMIB用データベースを実現することとする。

3.2 自律動作機能の具備

OSI管理では、M-EVENT-REPORTによる事象報告機能が定義されている。しかしながらOSI管理のアプリケーションでは、MIBに含まれるMO間、特に包含関係にあるMO間で、あるMOの事象が発生したことを他のMOに反映させる必要が生じる場合がある。例えば、コネクションがMOとして定義されており、上位のプロトコルモジュールのMOでコネクション数の管理を行なっている場合、コネクションの発生、消滅は上位のMOの属性値の変更を必要とする。同様に、回線終端装置ユニットで収容している個々の回線終端部が障害になった場合、ユニット自体の装置状態も部分障害に移行させる必要が発生する。データベース内部でこのような一貫性の保持を行なわせることで、MIBを利用するアプリケーションの実現がより容易になると考えられる。従って、MIB用データベースでは事象報告機能ではなく、より汎用的な自律動作機能を実現することとする。

4. MIB用データベースの設計

4.1 基本方針

MIB用データベースを専用のデータ管理機能として実現するに当たって、以下の設計方針をたてた。

- MIBのデータモデルと、CMIS操作をデータベースのモデル、操作とする。
- MIBの要求する事象報告機能をより汎用化した自律動作機能を実現する。
- 拡張可能データベース構築技法に基づき、階層的なモジュール構成をとるデータベースとして実現する。

以降の節では、MIB用データベースの設計について、特に外部仕様、データの格納方式、CMIS操作の実現方式、自律動作機能の実現方式について述べる。

4.2 外部仕様

4.2.1 モデル

データベースのモデルは以下に示すOSI管理が定めるMIBのモデルをそのまま使用する。

- 各MOは特定のMOクラスに属する。
- MO間に木構造の包含関係があり、包含木を構成する。MOは任意の数のMOを包含できる。包含可能なMOのクラスは属するMOのクラスにより定義される。
- MOは複数の属性を持ち得る。
- MOが持ち得る属性の種類はMOの属するクラスにより定義される。持ち得る属性には必須の属性とオプショナルな属性がある。
- 属性は通常1つの値を持つ。但し、複数の値を持つ属性も存在する。
- 属性値の型はASN.1により規定される。値の長さ制限はモデル上にはない。

4.2.2 データ定義機能

- 一般にMIBでは動的なスキーマ変更機能は不要と考えられるので、MIB用データベース中にデータとしてスキーマを格納し、特にデータ定義系の言語は持たない。
- 動的なスキーマ変更を行なわないため、スキーマアクセスの高速化の観点から、表1に示したテンプレートから継承関係やグルーピングを全て展開したMOのクラス定義と属性定義の2種類のデータを使用する（図1）。

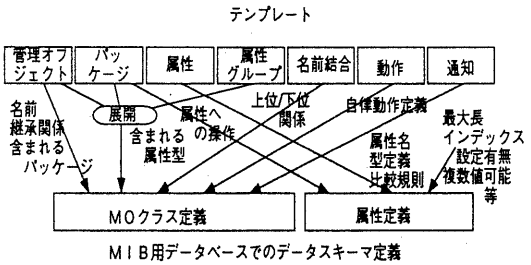

図 1: MIB データスキーマの表現

- MO のクラス定義には、その MO の名称やオブジェクト識別子に加えて、直接包含し得る MO のクラス、上位の継承関係にある MO のクラス、RDN として使用可能な属性型、その MO に含まれる必須とオプショナルな属性型、及び自律動作の定義を含む。自律動作の定義については 4.5節で示す。
- 属性型の定義には名前、オブジェクト識別子、CMIS 操作上の比較規則、その属性に許される操作、複数の属性値を許容するか否か、デフォルト値等のスキーマ情報に加えて、実装に依存する情報、例えば格納時の型、インデックスの有無、最大許容する属性値長を格納する。また、デフォルト値については ASN.1 の任意の型が記述可能なため、デフォルト値を設定する関数へのポインタを格納する。

4.2.3 データ操作機能

- MIB 用データベースへの操作は UNIX のソケットによるプロセス間通信を用いて行なう。
- 通信上の形式は ASN.1 の基本符号化規則（BER）により操作/応答を符号化したものとする。さらに操作の符号化長および操作/応答で複数の操作を識別するための操作識別子を付加する。
- 表 2に示した CMIS 操作の内、M-EVENT-REPORT、M-ACTION を除く操作をデータ操作言語とする。
- MIB 用データベースからの応答には自律動作の結果としての M-EVENT-REPORT、M-ACTION の返送を含む。(4.5節参照)

4.3 データ格納方式

基本的に筆者らが既に開発した OSI ディレクトリ用高速 DBMS:ASSIST/D [7] で用いたデータ格納方式を使用する。以下にその概略を示す。

4.3.1 基本方式

- 拡張可能 DBMS の構築技法に基づきデータ格納機能を機能別に 9 層に階層化（図 2）して実現する。各層と使用するモジュールの機能を表 3に示す。

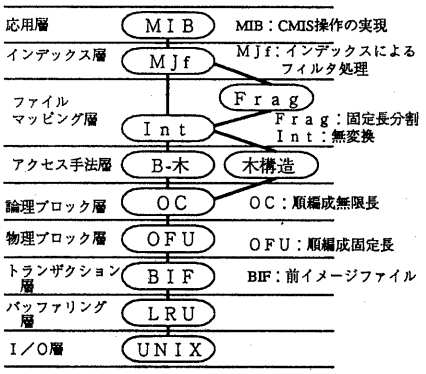

図 2: MIB 用データベースの構造

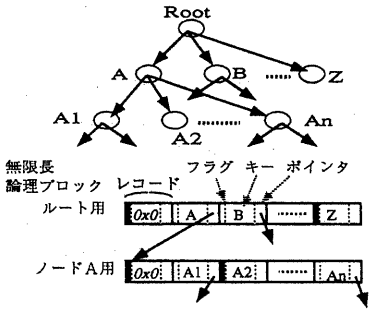

図 3: 包含木に対応する木構造アクセス手法

- 基本的なデータ格納機能として B+-木と包含木に対応した木構造アクセス手法（図 3）を提供する。
- 排他制御はページ単位の R/W ロックを用いる。
- 前イメージファイル（BIF）アルゴリズムを用いてトランザクションの原子性を実現する。
- LRU アルゴリズムによりページ単位のバッファリングを行なう。
- 実ファイルシステムは通常の UNIX のファイルを使用する。

4.3.2 包含木の格納

- 包含木情報を MO の格納から分離して、独立した情報として格納し、包含木に対する名前解析処理の高速化を図る（図 4）。包含木情報は応用層からは識別名をキーとする識別名インデックスとしてみえる。このインデックスはアクセス手法層で木構造アクセス手法（図 3）により格納する。
- 識別名の内部表現としては、RDN ごとに正規化処理を行ない、さらに ASN.1 の基本符号化規則（BER）で符号化したものを用いる。
- 各 MO には MO 毎に付与した内部通番（以下オブジェクト識別子と呼ぶ）を格納する。

表 3: 各層とモジュールの機能

層	層の機能	モジュール	モジュールの機能
応用	応用依存の操作の実現	MIB	CMIS 操作を実現
インデックス	フィルタ処理の実現	MJf	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。大量の中間結果は 2 次記憶上にバッファする
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	BIF	コミットまで 2 次記憶上で更新前イメージを保持
バッファリング	バッファリングにより I/O を減少	LRU	LRU アルゴリズムを提供
I/O	システム依存の I/O 操作を共通化	UNIX	UNIX のファイルを 2 次記憶とし、UNIX のシステムコールを使用する

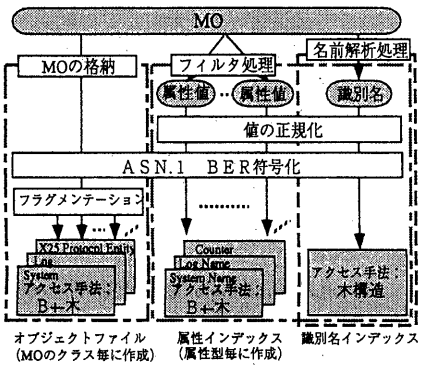

図 4: 包含木と MO の格納方式

4.3.3 MO の情報の格納

- MO 本体は、検索の高速化を考慮して、M-GET 操作の結果の型に変換して ASN.1 の BER で符号化し、B-木（B+-木）に格納する（図 4）。格納には MO のオブジェクト識別子をキーとして使用する。
- ファイルマッピング層で不定長の MO の符号化結果を固定長に分割し（フラグメンテーション）、格納の効率化を図る。
- フィルタ検索の高速化のため、MO に含まれるフィルタ条件となり得る属性の属性値に対して、B+-木によるインデックスを付与可能とする。作成の有無はデータ定義系で示した属性のスキーマ情報として指定する。

4.4 CMIS 操作の実現

CMIS 操作は操作対象となる MO を操作範囲の指定とフィルタから抽出し、それらの MO に対して操作を実行する、という 2 段階で実現する。以下では、操作対象 MO の抽出処理、大量の MO に対する操作の実現への対処ならびに同時実行制御方式について述べる。

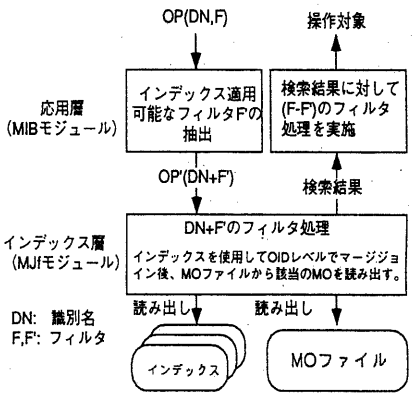

図 5: 操作対象エントリの抽出

4.4.1 操作対象 MO の抽出

操作範囲の部分木指定を単なる識別名に関するフィルタとみなし、MIB に対してフィルタ処理を行ない、結果の MO に対して操作実行を行なう。フィルタには集合演算を含むものもあるが集合演算はインデックス検索で実現できない。従って、図 5に示すようにインデックス層と応用層の 2 段階でフィルタ処理を実現する。

- 応用層はインデックス検索が可能なフィルタ条件を抽出し、識別名に対するフィルタと合わせてインデックス層に渡す。
- インデックス層は識別名インデックスを含むインデックスが適用可能な条件に対するフィルタ処理を実施する。インデックス層ではフィルタ処理は全てオブジェクト識別子の比較により行ない、実際の MO 本体は検索しない。
- 応用層はその他の条件に対するフィルタ処理、即ちインデックスの設定されていない属性に対する

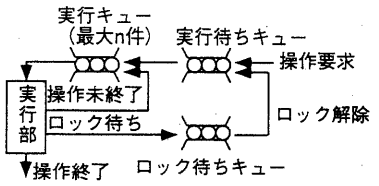

図 6: インターリーブ制御

条件あるいは属性値の集合演算操作による条件に対するフィルタ処理を行なう。インデックス層でのフィルタ結果の MO に対して実際の MO の内容を読んでフィルタ処理を行なう。

4.4.2 大量 MO に対する操作への対処

広い範囲に対してフィルタが指定された場合に備えて、フィルタ処理を行なうインデックス層では、オブジェクト識別子専用の中間バッファを持つ。フィルタの中間結果のオブジェクト識別子の集合がこのバッファを溢れた場合、外部ファイルに書き出してフィルタ処理を継続する。

また、MIB では 1 つの操作で複数の MO が更新できるため、トランザクション層では更新前のページを 2 次記憶領域に書き出し（BIF モジュール）、1 つの操作での大量の MO の更新に対処する。

4.4.3 同時実行制御方式

- インターリーブ制御
  M-GET 等の 1 つの CMIS 操作では、フィルタ結果により複数の MO に対する操作が必要になる場合がある。そこで、フィルタ結果として操作対象となった MO を単位とするインターリーブ制御を行なう。（図 6）同時実行数が増加すると平均的に応答速度が低下するため、実行用のキューとして実行キュー、起動待ちキュー、ロック待ちキューの 3 種類を設け最大同時実行数を制限する。
- 更新時の atomic、bestEffort 処理
  複数の MO に対する処理では 1 つでも失敗したら全て失敗とする（atomic）、失敗してもその MO だけ失敗にとどめ可能な限り他の MO に対して操作を継続する（bestEffort）の 2 つのトランザクション処理が選択できる。ここでは atomic 指定時は操作全体に対してトランザクションとし、また bestEffort 時は MO 毎にトランザクションとすることでこの 2 つの処理を実現する。
- ロックによる一貫性保持
  アクセス手法層で B+-木、木構造の両者に対してページ単位の R/W ロック管理を行なう。
- デッドロック処理
  実行中にロック待ちにかかった場合、応用層でロック待ちキューに繋ぐ。また、応用層ではロック待ち状態をトランザクション間の有向グラフで管理し、ループが発生することでデッドロック検出を行なう。デッドロックが発生した場合、最後にデッドロックの原因となったトランザクションをアボートし、デッドロックを解除する。

4.5 自律動作機能

4.5.1 自律動作の定義

- 自律動作定義（ルール）は、MO のクラス毎に定義する。特定の MO のみに自律動作を定義する場合は発火条件により指定する。
- ルールは発火条件と動作からなる。
- フィルタを用いて発火条件を記述する。なお、フィルタ条件における属性の指定の際に属性 ID のローカル形式を指定することで、比較対象としてその MO の属性を指定することを可能とする。また、この他にフィルタを拡張して MO 生成／更新／削除も発火条件として指定可能とする。
- CMIS 操作（M-SET、M-DELETE 等）を動作の記述に使用する。これにより、通常アプリケーションで記述しなければならない MO 間のデータの一貫性の保持動作を定義する。ここで M-EVENT-REPORT 操作を定義すれば標準/勧告で示す事象報告機能となる。また、ローカルな MO 指定用のパラメタを利用して、包含木における上位 MO を識別名を明示せずに指定することを可能とする。

4.5.2 自律動作の実現

- 全ての更新操作に対して、更新後の MO の値がその MO クラスの全てのルールの発火条件を満たすか否かを判定する。条件成立時はそのルールの動作（CMIS 操作）を取り出し、他の CMIS 操作と同様に操作待ちキューにつなぐ。
- この自律動作は他の CMIS 操作と同様に通常のインターリーブ制御により実行する。新たな種類のロック等によりその操作の正当性を保証することは行なわない。
- 操作は自律的に実行されるため、操作結果は MIB 用データベースの利用者には通知しない。操作が M-EVENT-REPORT、M-ACTION であれば、操作実行を行なわずデータベース利用者にその操作を通知する。

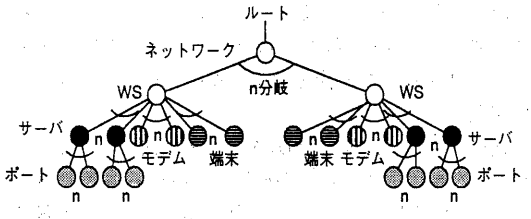

図 7: 評価に使用した MIB 構成

5. 実装と評価

5.1 実装

4.章で述べた設計を基に MIB 用データベース ASSIST/M の実装を SUN OS4.1.2 上で行なった。今回は専用データベースによる CMIS 操作の性能を評価するために基本的な CMIS 操作の実現のみを行なった。実装は OSI ディレクトリ専用 DBMS である ASSIST/D[7] が拡張可能 DBMS としてモジュール化されているため、ASSIST/D をベースに応用層、インデックス層、トランザクション層のモジュール変更により行なった。実装言語には C を用い、さらに ASN.1 基本符号化規則 (BER) のコンパイラ [9] を使用し、ASN.1 内部表現用の構造体と符号化/復号関数を自動生成した。

5.2 評価

実現した MIB 用データベースの MO 作成、検索、更新性能をみるために、MO 格納件数をパラメータとして以下の評価を行なった。

使用する MIB

- 6 種類の MO のクラス（ネットワーク、WS、サーバ、モデム、端末、ポート）があり、各々4 から 5 件の属性を持つ。属性値の大きさは MO 当たり 100 バイト程度である。
- 包含木の構成を図 7に示す。

パラメータ

- 図 7の包含木の各段での分岐数（n で示されている）を変化させて、格納する MO の数を変更する。

測定項目

- M-CREATE 操作により MO を 1 件新規作成する操作応答時間
- M-SET 操作によりポートクラスからランダムに選択した MO の 1 つの属性値（インデックス有り）を変更する操作応答時間
- M-GET 操作によりポートクラスからランダムに選択した MO を読み出す操作応答時間
- M-GET 操作によりサーバクラスからランダムに選択した MO 以下の部分木に含まれる MO（分岐数を n とすると n+1 件）全てを読み出す操作応答時間

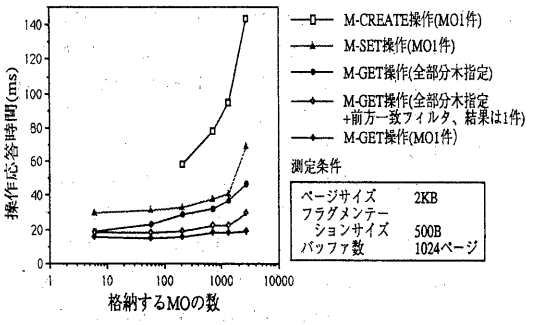

図 8: CMIS 操作の MO 格納件数に対する操作応答時間

- M-GET 操作によりサーバクラスからランダムに選択した MO 以下の部分木に含まれる MO（分岐数を n とすると n+1 件）に、さらに前方一致によるフィルタをかけ、結果の MO1 件を読み出す操作応答時間

図 8に操作応答時間の測定結果を示す。

6. 考察

6.1 性能評価結果について

(1) M-GET 操作による検索性能
から 1 件の MO を検索する M-GET 操作を約 20 ミリ秒弱で実行できた。この値は評価した格納 MO 件数の範囲では殆んど変化しないことが分かった。

(2) フィルタ処理性能
操作応答時間は概ねフィルタ検索の範囲ではなく、フィルタ結果として操作対象となった MO の件数に比例しているため、フィルタ範囲に依存しないフィルタ処理が実現できた。

(3) M-SET 操作による更新性能
更新対象とした属性にインデックスが付与されているため、10^3件 MO を格納した状態で更新に約 40 ミリ秒程度必要としている。格納 MO 件数が最大（約 2800 件）になった場合を除いて概ね格納件数の変化に対して応答速度は変化しなかった。格納件数が最大になった点で急激に操作応答時間が増加したのは、今回用いた M-SET 操作で更新した属性値の値域が狭く、同一の属性値が多数存在するため、属性インデックスで更新するページが増加したためと考えられる。

(4) M-CREATE 操作による MO 作成性能
MO1 件当たりの作成時間の増加は、包含木情報を格納する木構造アクセス手法における更新性能の低下と、上記の M-SET 操作での更新性能の低下の両方の原因によると考えられる。

(5) OSIディレクトリ用DBMS:ASSIST/Dとの比較
格納するエントリ件数が3000件程度以下の範囲では、ASSIST/Dでのエントリ読み出し操作、更新操作はそれぞれ7ミリ秒、15〜16ミリ秒程度であり、MIB用データベースの半分以下である。トランザクション層がASSIST/DはBIFではなく主記憶上のアルゴリズムを用いる点を除けば下位のデータ格納部分は同一であり、また評価したエントリとMOの1件当たりの大きさも殆んど同一であることから、検索性能の差は応用層で格納・通信に使用したASN.1の符号化規則（ASSIST/Dはライトウェイト符号化規則[10]）による所が大きいと考えられる。MIB用データベースではエージェントとの通信を考慮して一般的な符号化規則である基本符号化規則（BER）を用いたが、符号化処理の遅延が大きいことが分かった。

(6) 汎用DBMSによるMIB用データベースとの比較
汎用DBMSを用いたMIB用データベースの実現例としてRDBMSを用いた例[4]、OODBを用いた例[6]と比較すると、これらでのMIBの格納MO数は不明であるため正確な比較とはいえないが、RDBMSの場合と比較して平均10倍程度、OODBの場合とは2倍程度高速であった。

6.2 拡張可能DBMSによる効率的な実現について

ASSIST/Mの開発はベースとしたASSIST/Dの9層のうち、6層のモジュールをそのまま使用でき、残る3層のモジュールのみを新規に作成した。拡張可能データベース技法に基づきデータベースがモジュール化されていることで、多くのモジュールがそのまま再利用できたといえる。

プログラムサイズの面からはASSIST/M全体のプログラムサイズは32.2Kステップであり、内訳は応用層が17.6Kステップ、インデックス層以下の下位層が14.6Kステップであった。応用層は約3Kステップを従来のASSIST/Dのモジュールから再利用し、残りを新規に作成した。また下位層はインデックス層、トランザクション層のモジュール変更により5.0Kステップを新規に作成し、残りの9.6KステップはASSIST/Dのモジュールをそのまま使用した。従って、合計で12.6Kステップが再利用でき、特にアクセス手法といった実現の難しい部分を新規に作成する必要がなかったことで、効率的なデータベースの実現ができた。

7. むすび

本稿では、OSI管理情報ベース（MIB）に必要なデータ格納機能を提供するMIB用データベースASSIST/Mの設計と実装について報告した。本データベースは、拡張可能データベース技法に基づいてMIBのデータモデルとCMIS操作を直接提供する専用データベースとして実現した。

既に作成した拡張可能DBMS構築技法に基づくOSIディレクトリ用DBMSであるASSIST/Dをベースとして、9層のモジュールの内3層のモジュールを置換することによりCMIS操作を実現する基本部分を効率的に実装した。評価の結果、MOを10^3件程度格納した場合、1件のMOに対するM-GET操作の操作応答時間が20ミリ秒、M-SET操作の応答時間が40ミリ秒弱で実現でき、汎用DBMSを用いて実現した場合と比較して高速なMIB用データベースが実現できた。

今後は、今回実装しなかった自律動作機能を実装し評価を行なっていく予定である。最後に日頃御指導頂くKDD研究所浦野所長、眞家次長並びに御討論頂いたネットワークエンジニアリング支援グループ浅見グループリーダに感謝します。

参考文献

[1] CCITT勧告 X.700シリーズ/ISO 国際標準 9595,9596, 他, OSI管理, (1991).

[2] 西山 他: ASSIST/M: OSI管理情報ベース（MIB）用データベースの設計, 1993年電子情報通信学会秋季大会論文集 B-581, (1993).

[3] 桐葉 他: 管理情報ベース（MIB）の開発支援環境, 情報処理学会データベース研究会資料 DB86-9, (1991).

[4] 依田 他: 伝送網オペレーションにおける管理情報ベース（MIB）の構成法, 電子情報通信学論文誌 Vol75-B-I, No.8, pp.517-527, (1992).

[5] 桐葉 他: 管理情報ベース（MIB）のオブジェクト指向DBによる実現方式, 1993年電子情報通信学会秋季大会論文集 B-585, (1993).

[6] 清水 他: オブジェクト指向データベースによる管理情報ベース（MIB）の実装, 1993年電子情報通信学会秋季大会論文集 B-702, (1993).

[7] 西山 他: 拡張可能DBMS構築技法に基づく高速OSIディレクトリ専用DBMSの設計と評価, 情報処理学会論文誌 Vol.34, No.6, (1993).

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

[9] Hasegawa, T., et. al: Implementation and Evaluation of ASN.1 Compiler, Journal of Information Processing, Vol.15, No.2, (1992).

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