# 構造符号化による軽量 XML データベースの実装

## Structure Coding based Light-Weight XML Database －Design and Implementation－

服部 雅一♥ 宮澤 隆幸♥
芦川 将之♥ 神長 昭子♥
野々村 克彦♥

Masakazu HATTORI Takayuki MIYAZAWA
Masayuki ASHIKAWA Akiko KAMINAGA
Katsuhiko NONOMURA

**XQuery 処理の構造パターン照合について、ストック型やストリーム型など様々な手法が提案されてきた. ただ、ディスク I/O 処理や CPU 処理に対するインパクトが大きく、GB オーダの大規模 XML データに対して十分な応答性を期待することはできなかった.**

**本論文では、XQuery を対象にした軽量な構造パターン照合の手法を提案する. XML データの構造に対して構造ガイドを使ってコード化し、圧縮した上で構造ストリーム化する. そして、XQuery をペトリネットに類似した走査プログラムにコンパイルして、1 回の構造ストリームに対する走査で処理を完了する.**

**さらに、ベンチマークテストを使って、本手法に基づく XML データベースが、従来 XML データベースを凌駕するパフォーマンスを持つことを確認した.**

In regard to structure pattern matching of XQuery processing, various approaches such as stock based approach and stream based approach were suggested. But the load of disk I/O and CPU processing was too high to maintain acceptable response time for large-scale XML data of the GB order.

This paper proposes light-weight structure pattern matching method to speed up XQuery processing. Our method introduces a structure data guide, encodes structure elements of XML data, translates them into a structure stream after compressing it, compiles XQuery expression into a structure scan program that resembled Petri-net, and carries it out with one-pass structure stream scan.

Furthermore, results of XMLDB benchmark shows our method achieves more than 10 times performance gain compared to conventional native XML database management systems.

♥ 株式会社 東芝 研究開発センター
知識メディアラボラトリー
{masakazu.hattori, taka.miyazawa, masayuki.ashikawa, akiko.kaminaga, katsuhiko.nonomura}@toshiba.co.jp

## 1. はじめに

XML(Extensible Markup Language)は、インターネット上の交換・蓄積のフォーマットとして標準化された新たなデータ記述言語として期待されている. 近年、XMLの柔軟性や拡張性を生かすため、スキーマレスでXMLデータを格納するネイティブXMLデータベースに注目が集まっている. このXMLデータベースに対する問い合わせ言語として、2007年1月にW3Cで正式勧告となったXQuery(An XML Query Language)がある. XQueryは性的型付け機能を持つ関数型言語であり、RDBMSで提供されている宣言型の問い合わせ言語SQL(Structured Query Language)と比較していくつかの相違点がある. その一つにXMLに対する構造パターン照合がある. 構造パターン照合とは、親子や兄弟などの階層条件やデータ比較条件を指定して、条件を満足するXMLの部分構造を抽出するデータ参照機能である.

従来のネイティブXMLデータベースは、レコード型に近いXMLデータの検索や複雑なXQuery検索において、システム構築時に人手で十分にチューニングされないと、性能が出せないケースがあった。その中で大きくクローズアップされたのが、構造パターン照合における性能劣化である. 多くのネイティブXMLデータベースでは、XMLデータの階層構造をノードとリンクで表現しているが、構造パターン照合はリンクを順々に辿る処理で実現されるため、ディスクI/O処理やCPU処理に対するインパクトが大きく、大規模なXMLデータに対して応答性を確保することが困難であった.

この問題を解決するため、我々はアーキテクチャが従来のものと異なるXMLデータベースの研究開発を進めてきた. これは従来のコンパクトな数値配列だけで、XMLデータの階層構造を表現し、数値配列に対する1回の走査だけで構造パターン照合が行える新たな手法である. それによって、XQuery中に複雑な構造パターン照合があっても、ディスクI/O処理やCPU処理に対するインパクトを極小化できる. 本論文では、アーキテクチャのコアとなるXML構造符号化格納技術とXQueryコンパイル技術を中心に説明する.

## 2. XML データベースのアプローチ

### 2.1. 従来のアプローチ

XQueryを処理するアプローチとして、ストック型処理とストリーム型処理に大別される. ストック型とは、XMLデータを一旦記憶領域(ディスクやメモリ)に蓄積し、それから記憶領域にアクセスしてXQueryを処理するものである. 一方、ストリーム型とは、XMLデータを記憶領域に蓄積すること無しにXQueryを処理するものである. 前者は、データベースなど大規模データ対応を目的にしているが、後者は簡易プロセッサなど少メモリや逐次応答性を目的にしており、両者は相補的な関係にある.

ストック型として、XMLを写像してRDBMSに格納するものと、XMLをネイティブに格納するもの(ネイティブXMLデータベース)がある. 前者には構造写像アプローチやモデル写像アプローチがある. 後者にはCLOB的に格納するものがあるが、その多くはXMLの持つツリー構造をディスクにそのまま格納するというものである[1]. XMLの持つツリー構造をディスクにそのまま格納する場合、複数の構造パターン照合を含む複雑なXQueryでは、構造パターン照合の回数分だけ同一のXMLデータの部分構造に対するアクセスが発生してしまう. 特に、メモリに乗り切れない数GB以上の大規模XMLデータを格納す

る場合、主な記憶領域はディスクになるが、同一ブロックへのディスクI/Oが大量に発生して、性能劣化が激しくなる．また、ツリー構造をDOMのようなノードとリンクで表現してXMLデータを記憶領域に展開する場合、構造に限定して原文の4～10倍の記憶領域サイズが必要になる．

### 2.2. 我々のアプローチ

我々は10GBのXMLデータを蓄積したネイティブXMLデータベースに対して高速に構造パターン照合が行えるアプローチについて検討してきた．それは、ストック型とストリーム型を融合したものである．

#### XML 構造符号化格納技術

まず、XMLデータ＝構造データ＋テキストデータ、と考えて、構造とテキストを分離する．登録するXMLデータから、XMLデータ集合の全体に渡る構造データの縮約である構造ガイドと呼ばれるデータ構造を抽出する．これはDataGuide[2]と同じであるが、構造ガイドの各要素にはユニークなGID (Guide Element ID)と呼ばれるシリアルナンバーが付与されている．さらに、XMLデータの構造要素を構造ガイドの対応するGIDを使って、数値配列に変換する．これを構造ストリームと呼ぶ．構造要素は、SID(Structure Element ID)と呼ばれる物理的なIDで取り出すことができる．

#### XQuery コンパイル技術

XPathやXQueryなどの問い合わせ文が与えられると、構造パターン照合の部分に着目して検索構造木を抽出し、走査プログラムと呼ばれる中間データを生成する．さらに、構造ストリームを走査して、走査プログラムにGIDとSIDを次々と入力することで、問い合わせ文に含まれる複数の構造パターン照合の条件を満足するSID、またはその他データの組の集合を出力する．テキストデータは、テキストストリームに格納されるが、テキストデータを取り出すには、構造ストリームを経由してテキストストリームを走査する必要がある．走査プログラムのデータ構造は、並行動作を表現するために考案された理論的計算モデルのペトリネット(Petri-Net)[3]に類似している．ペトリネットは、プレース(Place)やトランジション(Transition)を使った二部有向グラフであるが、走査プログラムも同様に、プレース(Place)とトランス(Trans)によって表現される．構造ストリームの走査回数は、XPathやXQueryによらずに全て1回である．そのため、走査プログラムの実行もバックトラックを使わないCPU処理へのインパクトが小さいものになっている．

## 3. ネイティブ XML データベースの構成

上述した手法を実装したネイティブ XML データベース管理システムのシステム構成を図 1 に示す．クライアント・サーバ型のデータベース管理システムで API を通してデータベースにアクセスする．障害回復機能は、ログ(Log)と WAL(Write Ahead Log Protocol)[4]を独自にカスタマイズしたプロトコルで実現されている．

また、各データ間の論理的な実体関連を図 2 に示す．ストリームデータとして、構造ストリーム、テキストストリーム、索引ストリームの 3 種類があるが、索引ストリームは未実装である．テキスト集合は親要素でクラスタリングされて複数(デフォルト 10 個)のテキストストリームに分散配置される。各テキストは先頭の固定長部分とそれ以外の可変長部分に分割して 2 種類のテキストストリームに格納される．

以下、XML 構造符号化格納技術と XQuery コンパイル技術に関連のある、XML 登録処理と XML 検索処理を中心に説明する．

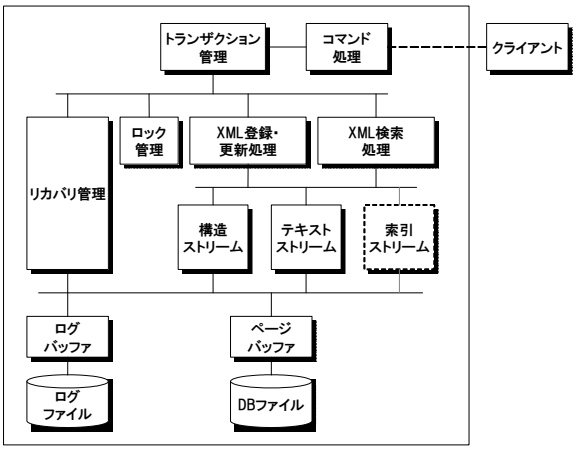


図 1　システム構成
Fig.1　System Configuration of XML Database

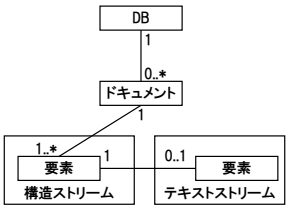


図 2　データ間の論理的な実体関連
Fig.2　Entity Relationship Diagram of XML Database

### 3.1. XML 登録処理の概要

XML データの登録処理のフローチャートを図 3 に示す．フローチャートでは、XML データから文書要素を取り出して必要な処理を行うというループ構造になっているが、実装上はイベントベースの XML パーサ SAX(Simple API for XML)を使っている．構造要素のタイプ毎に GID を計算して、構造ストリームに GID を追加する．これは 1 次元の数値配列になるが、数値はランダムな値をとらずに 1 つずつ連続的に増加といった規則性があるので、部分数値配列を定期的に圧縮する．圧縮率は、XML の構造化の度合いによって変化する．実験では、XMark[5]のように DTD は存在するものの複雑な構造を持つ XML の場合の圧縮効果は 17 倍であったが、50 個のカラムがフラットに並ぶレコードの場合だと圧縮効果は 387 倍になった．また、テキスト要素の場合は、構造ストリームに簡易なリンク情報を付けることで、構造要素からテキスト要素をたどれるようにしている．

登録する XML データのサンプルを図 4 (a)に示す．以降、XML データベースのルートノード直下にサンプルデータを登録するケースを考える．サンプルデータが追加された状態での構造ガイドのイメージを図 4 (b)に示す．XML データの構造要素には対応する構造ガイドの構造要素 GID が割り当てられる．構造要素<BOOK>には GID2、“階層構造”というテキストデータを子要素に持つ構造要素<KEY>には GID15、というように GID が割り当てられて、それらが数値配列になったものが図 5 の構造ストリームである．図は圧縮前と圧縮後を示している．

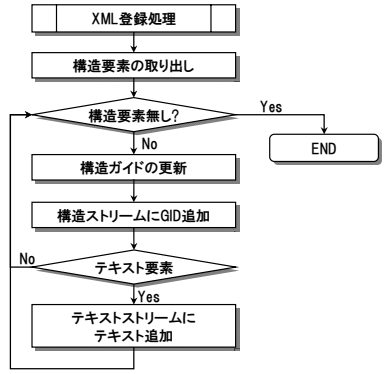


図 3 XML データの登録処理
Fig.3 Registration Processing for XML Data

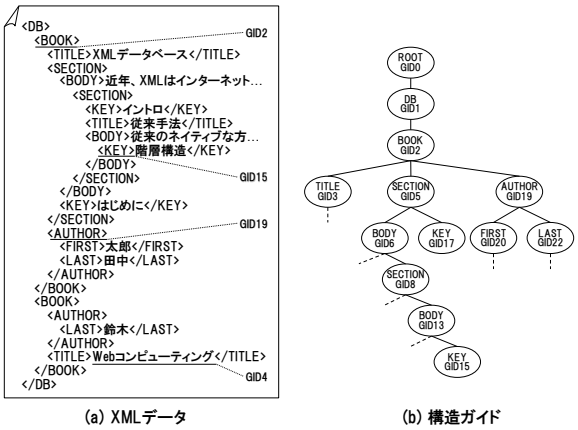


図 4 XML データと構造ガイド
Fig.4 XML Data and DataGuide

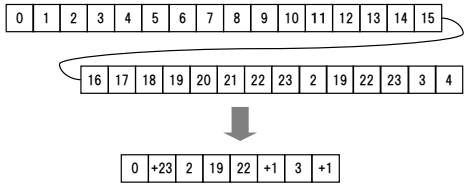

図 5 構造ストリーム
Fig.5 Structure Stream

### 3.2. XML 検索処理の概要

XML データの検索処理のフローチャートを図 6 に示す．まず、XPath や XQuery の問い合わせ文が与えられると、構造パターン照合の部分に着目して検索構造木を抽出するが、問い合わせ文によっては、複数の検索構造木が抽出される．そして、検索構造木から走査プログラムを作成するが、複数の検索構造木が抽出された場合でも走査プログラムは1つである．その走査プログラムを実行することで、結果テーブルが生成される．検索構造木が複数であれば、結果テーブルは同じ数だけ生成されるが、これら結果テーブルが構造パターン照合を満足した SID、またはその他データの組の集合である．それ以外にも、データ比較条件があれば、SID の代わりにテキストストリームから取得されたテキストデータがセットされる．

普通の問い合わせ文では、構造パターン照合以外に条件が入り込んでいる．例えば、データ比較やグループ化などであるが、この場合はリレーショナルデータベースと同様のリレーショナル演算を実行することで最終的な XML データの集合を生成し、結果出力する．

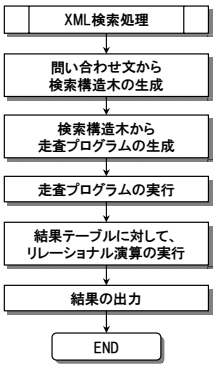


図 6 XML データの検索処理
Fig.6 Query Processing for XML Data

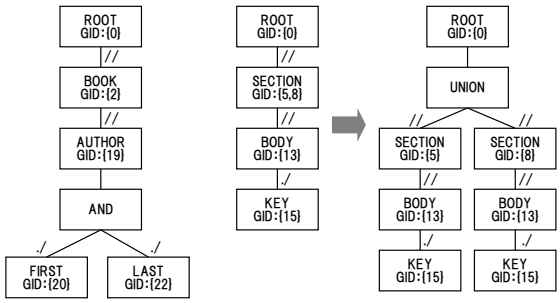


図 7 検索構造木
Fig.7 Query Skelton Tree

以降、以下の Q1、Q2 の 2 つの簡単な問い合わせ文を使う．

```
for $x in //BOOK
for $y in //AUTHOR[./LAST and ./FIRST]
return <res>{$x, $y}</res>                    ...(Q1)

for $x in //SECTION
for $y in $x//BODY[./KEY]
return <res>{$x}</res>                        ...(Q2)
```

検索構造木は、XPath 1.0 におけるロケーションパス(軸、要素名/型テスト、述部)の関係を要約したツリーである．Q1、Q2 に対応した検索構造木を図 7 (a)(b)に示す．ノードは構造要素の変数をアークは変数間の制約関係を表しているが、XPath だけではなく XQuery の FLWR 式からも変数同士の制約関係を纏め上げて、1 つの検索構造木を生成している．ノードに付加されているのは、割当可能な構造ガイドの GID 集合である．

Q1 の検索構造木の BOOK ノードに対しては、図 4 (b)の構造ガイドから GID2 を持つ構造要素のみが具体化可能であること、Q2 の検索構造木の SECTION ノードに対しては、5 か 8 のいずれかの GID を持つ構造要素のみが具体化可能であること、を表している．これは、ノード間の軸であれば descendant-or-self, descendant, following-sibling などの 2 項制約があると考えて、事前に全ノードに全ての GID 集

合を割り当てて、不要な候補を制約関係で絞り落としていく制約伝播アルゴリズムにより実現されている．また、検索構造木を不要な中間ノードを削除するなど木の簡略化を行う．さらに、ノードに割り当てられた GID 同士に先祖・子孫関係を持つものが発生する場合、走査プログラムの実行結果に結果データの欠落が発生するので、UNION 制御ノードを挿入して上位のノードから分割する．図 7 (b)はその例である．SECTION の GID5 と GID8 は構造ガイドでは先祖・子孫関係にあるので、その部分を基点に UNION 制御ノードを挿入して分割する．

### 3.2.1. 走査プログラムの生成

検索構造木から走査プログラムを生成する部分について説明する．まず、Q1、Q2 に対応した走査プログラムを図 8 (a)(b)に示す．エントリテーブルという全 GID に対応した配列があり、これがデータの入り口になる．構造ストリームを走査して、走査プログラムに GID と SID を次々と入力する．エントリテーブルの配列要素から複数プレースに PList(Push List)と CList(Clear List)という 2 種類のリンクが張られている．PList はプレースにデータをプッシュする操作、CList はプレースのデータをクリアする操作を意味する．

プレースはデータ列がキューとして記憶されるノードであり、SID、またはその他データ(ブール値、テキストデータ、SID セット)の組がキューにプッシュされる．また、トランスは、プレース集合を入出力として基本処理を行うノードであり、入力のプレース集合からデータ列を受け取り、マージや論理式評価などの処理を行い、その結果データを出力のプレース集合にプッシュする．最終的な結果データは、最終プレースに記憶されている．これが問い合わせ文に含まれる複数の構造パターン照合の条件を満足する SID、またはその他データの組の集合となる．

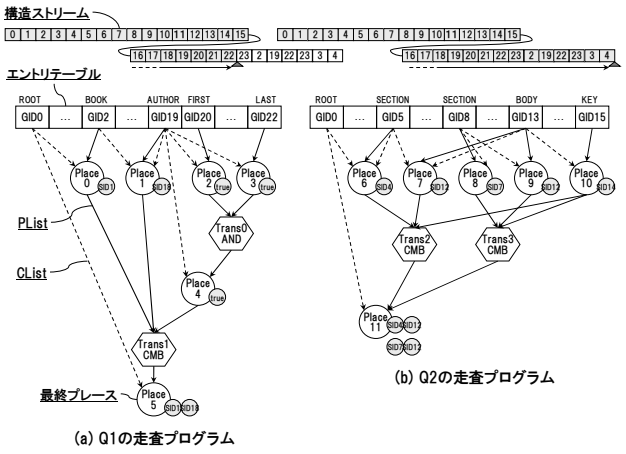


図 8　走査プログラム
Fig.8　Structure-Scan Program

走査プログラムの生成のフローチャートを図 9 に示す．検索構造木のノードを入力、プレースを出力とする再帰アルゴリズムになっている．初期入力はルートノードである．検索構造木のノードを AND や OR などの制御ノードかタグノードかを判定して、処理を 2 手に分けている．タグノードであれば、ノード間の軸 descendant-or-self, descendant, following-sibling などの 2 項制約があるので、2 つの入力プレースをマージトランス(CMB)で結合し、出力プレースを返す．これが走査プログラムにおける構造パターン照合の基本形である．それ以外であれば、AND など対応したトランスで入力プレース集合を結合し、出力プレースを返す．このように生成された走査プログラムには冗長なトランスが発生するので、このような冗長トランスを除去する必要がある．図 8 は冗長トランスを除去した結果になっている．

走査プログラムの生成の概要は以上の通りだが、より広範囲の XQuery を処理するため、いくつかの拡張が行われている．これまでトランスの種類として、マージトランスや AND トランスなどがあることを述べたが、他にも SID からテキストストリームを参照して定数値との比較を行うデータ比較トランスや XPath コンテキストポジションを計算して定数値との比較を行うポジション比較トランスなどのトランスも存在する．

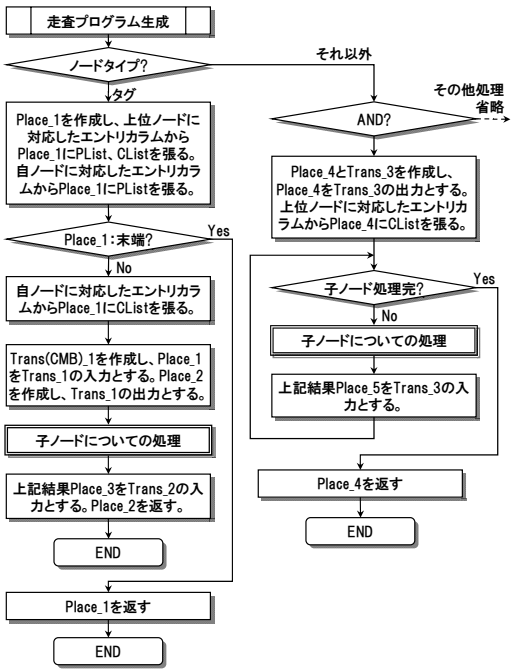


図 9　走査プログラムの生成
Fig.9　Generation Processing of Structure-Scan Program

### 3.2.2. 実行

走査プログラムの実行のフローチャートを図 10 に示す．構造ストリームを走査して、走査プログラムのエントリテーブルの該当カラムへのアクセスを起点に連結されたプレースやトランスへ変更伝播するアルゴリズムとなっている．図 8(a)には、最初の BOOK 要素に対応する GID22 までを走査した段階の走査プログラムの状態も表されている．以下にそのトレースを示す．

(t1) 構造ストリームの最初の GID、つまり GID0,SID0 の走査時点で、Place0 と Place5 をクリアする．
(t2) GID2,SID1 の走査時点で、Place1 をクリアした後に Place0 に SID1 をプッシュする．
(t3) GID19,SID18 の走査時点で、Place1～4 をクリアした後に Place1 に SID18 をプッシュする．
(t4) GID20,SID19 の走査時点で、ブール値として処理する必要があるので Place2 に true をプッシュする．
(t5) GID22,SID20 の走査時点で、ブール値として処理する必

要があるのでPlace3にtrueをプッシュする.それが起点となりTrans0はAND処理を行い、Place4にtrueをプッシュする.さらに､Trans1はPlace0,Place1,Place4のデータを組み合わせて､Place0,Place1のデータの組SID1,SID18を最終プレースPlace5にプッシュする.

これはQ1における$x と$yの値に他ならない. つまり、最初の<BOOK/>と<AUTHOR/>を意味している. 同様に、図8 (b)は､DB要素全てを走査した段階の走査プログラムの状態を表している. 最終プレース Place11 には、SID4,SID12 とSID7,SID12 という2つのSIDの組がプッシュされている.

応用形ではあるが､走査プログラム同士を単純にマージすることで、複数の問い合わせ文の一括処理が可能である. 例えば、Q1とQ2の走査プログラムをマージすれば、構造ストリームに対する1回の走査だけで、上記 Place5 と Place11 のデータ列を同時生成することができる.

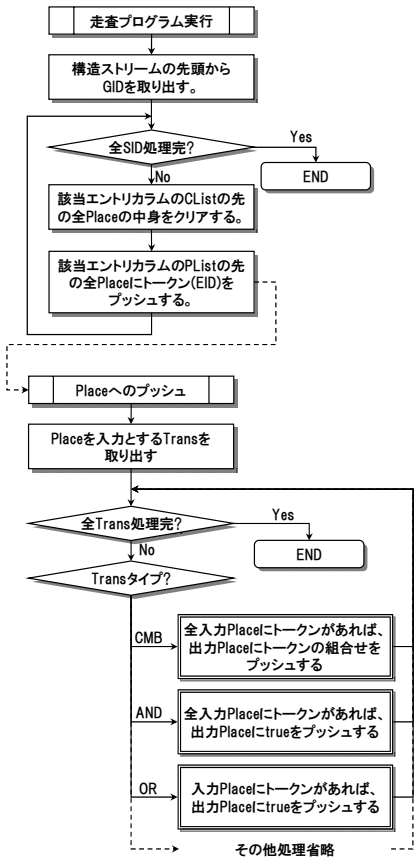


図10 走査プログラムの実行

Fig.10 Structure-Scan Program Processing

## 4. 評価実験

図1に示したネイティブXMLデータベースを構築し､XMLDBのベンチマークを使って実験を行った. ベンチマークとして、TPC-W をベースにしたトランザクションデータの XBench[6](Data-Centric Multiple Document 型､Large(1GB))を用いた. XBench は、address,author, country,customer,item,orderという6種類のデータから成り、全部で約26万件のデータ集合になる. このベンチマークを用いた理由は、レコードデータに近いデータセットを選ぶことで、なるべく対等な条件でRDBMSと比較したかったからである. 計測にはCPU Intel Xeon 3.00GHz×2、メモリ 3.5GB、HDD 500GB SCSI、OS Windows2003 Serverのスペックを持つマシンを用いた.

ベンチマーク対象となるDBMSとして、商用RDBMS-XML対応、RDBMS(SQLite3.3.15)、商用ネイティブXML-DBMSを選んだ. RDBMSに格納する場合、XBenchはXMLDBのベンチマークであるため前処理が必要になる. 具体的には、itemとorderのデータについては構造写像アプローチに基づいてそれぞれ2リレーションに分解してから格納した. いずれも索引設定をしていない. さらに、各DBMSのサポート状況を見て、XBenchで示されている複数のXQueryからタイプの異なる7つの問い合わせ文を選択した. ちなみに､各問い合わせ文は､ID指定(Q1, Q12, Q16)、大小比較と並び替え(Q10, Q11)、ID指定とジョイン(Q19)、構造パターン照合(Q12, Q6)というように分類される. また、RDBMSではSQLを使うので、XQueryに対応するSQL文を用意した.

### 4.1. 登録性能

各DBMSの登録時間を表1に示す. 約26万件のデータを登録するのに、全件登録した後に1回コミットをかけるか、1件登録ごとに毎回コミットをかけるかで登録時間が変わってくる. 1_COMMITは前者､N_COMMITは後者を意味している. データはファイルから読み込んでいるが、この読み込み時間はDBMSの処理とは無関係なので別途時間を測定して全体の応答時間から減算している.

表1からわかるように、本XML-DBMSは他のRDBMSやXML-DBMSと比較して、最も優れた性能を示している. 他のRDBMSやXML-DBMSは､コミットのかけ方によって登録時間が4～6倍程度変化するが、本XML-DBMSは3倍弱の小ささである. 1件コミットでの構造ストリームサイズとテキストストリームサイズはそれぞれ38MBと1.0GBである.

表1 登録時間

Table1 Execution Time for Registration

| | 本XML-DBMS | RDBMS-XML | RDBMS(SQLite) | 従来XML-DBMS |
|---|---|---|---|---|
| 1_COMMIT | 279 | ー | 498 | 1,701 |
| N_COMMIT | 825 | 2,236 | 2,222 | 9,158 |

ー：エラー 単位:s

### 4.2. 検索性能

表2 Cold状態の検索時間

Table2 Cold-Cache Execution Time for Query

| | 本XML-DBMS | RDBMS-XML | RDBMS(SQLite) | 従来XML-DBMS |
|---|---|---|---|---|
| Q1 | 11,044 | 101,037 | 15,695 | 173,312 |
| Q12 | 11,017 | 95,641 | 16,611 | 172,922 |
| Q16 | 11,367 | 97,872 | 14,645 | 172,718 |
| Q10 | 14,386 | 157,271 | 16,309 | 219,187 |
| Q11 | 10,621 | 162,117 | 16,346 | 217,094 |
| Q19 | 14,308 | 144,127 | 36,913 | 261,922 |
| Q6 | 11,814 | 231,336 | 18,571 | 397,094 |

単位:ms

表3 Warm状態の検索時間

Table3 Warm-Cache Execution Time for Query

| | 本XML-DBMS | RDBMS-XML | RDBMS(SQLite) | 従来XML-DBMS |
|---|---|---|---|---|
| Q1 | 709 | 18,804 | 459 | 171,484 |
| Q12 | 703 | 18,773 | 992 | 171,938 |
| Q16 | 678 | 18,757 | 498 | 172,234 |
| Q10 | 1,194 | 66,124 | 618 | 212,610 |
| Q11 | 1,553 | 66,527 | 612 | 212,469 |
| Q19 | 1,302 | 49,085 | 1,443 | 259,781 |
| Q6 | 1,055 | 149,726 | 2,842 | 396,110 |

単位:ms

Cold時の各DBMSの検索の応答時間を表2に示し､Warm時の応答時間を表3に示す. キャッシュにデータが存在しない場合はCold状態、同一の問い合わせ文を再実行することで

多くのデータがキャッシュに存在する場合は Warm 状態とした．ちなみに、この応答時間には最終結果の取得および表示を含んでいない．

本 XML-DBMS を RDBMS と比較すると、遜色の無いか、むしろ全体的に優れた応答性を示している．特に構造パターン照合については、ジョインが発生する RDBMS よりも本 XML-DBMS の方が 2～4 倍高速というケースもある．本 XML-DBMS を XML 対応も含めた他 XML-DBMS と比較すると、COLD 状態で 10～20 倍、Warm 状態で 25～400 倍程度の大きな性能差がついていることがわかる．1GB の XML データを内部展開するとメモリに乗り切れない．そのため、他 XML-DBMS はディスク I/O が多く発生して、Cold 状態と Warm 状態の性能差がわずかなものになる傾向がある．本 XML-DBMS の Cold 状態での応答時間がほぼ一定であり、構造ストリームとテキストストリーム、つまりディスクを 1 回だけしか走査しない手法の効果が現れている．さらに、検索における本 XML-DBMS でのメモリ使用量は 10MB 強であり、メモリ的にも軽量である．

## 5. 関連研究

同一の XML データの部分構造に対する繰り返しアクセスを抑える手法として、ストリーム型の構造パターン照合のアプローチがある．XML データに対するストリーム処理として、XML パーサである SAX が知られている．ストリーム型の構造パターン照合は、SAX のようなストリーム処理を XQuery や XPath に適用しようというものであり、XFilter[7]、YFilter[8]、XTrie[9]、TurboXPath[10]などの研究が挙げられる．これらの論文では、状態遷移を使った最適化、同時要求に対する応答改善、などが目的であり、多数の亜種とともに研究が盛んである．しかし、構造パターン照合を実現するのに、バックトラックをベースとした非決定的なアルゴリズムであるため、CPU 処理の負荷が大きいという問題がある．つまり、メモリサイズを越える GB オーダのデータを蓄積した XML データベースに対して、XQuery を高速に処理するのは困難であると言える．

また、従来のストリーム処理では子軸や子孫軸のみ、パス途中の述部が末端のみ、など XPath を限定している研究も多いが、本 XML-DBMS では例で示した AND トランスやマージトランス以外にポジション比較トランスや OR トランスやテキスト比較トランスなどのバリエーションにより、そのような制限は無く、ほとんどの XPath 表現が実装可能であると考えている．

## 6. おわりに

我々はアーキテクチャが従来のものと異なる XML データベースの研究開発を進めてきた．それが、XML 構造符号化格納技術と XQuery コンパイル技術である．これは、コンパクトな数値配列だけで、XML データの階層構造を表現し、数値配列を 1 回の走査だけで構造パターン照合が行える．それによって、XPath や XQuery の問い合わせ文中に複雑な構造パターン照合があっても、ディスク I/O 処理や CPU 処理に対するインパクトを極小化できる．結果として、検索の高速化を達成しただけでなく、少ない H/W リソースでも動作可能、少ないチューニングコスト、などの多くのメリットも獲得することになった．

今後の課題として、索引機能や XQuery サポート率向上などは課題として残されている．特にストリームデータと索引機能の無理のない統合方式については緒に就いたばかりであり、今後の重要テーマになる．

## [文献]


[1] 江田毅晴，清水敏之，XML データベース技術に関する研究動向 -研究の流れと XML 索引付け-，電子情報通信学会第 17 回データ工学ワークショップ第 4 回日本データベース学会年次大会（DEWS2006）ミニサーベイ，March 2006.

[2] R. Goldman and J. Widom, DataGuides: Query Formulation and Optimization in Semistructured Databases, In Proc. 18th ICDE, 2002.

[3] T. Murata, Petri Nets: Properties, Analysis and Applications, In Proc. IEEE, VOL.77, pages.541-580, 1989.

[4] C. Mohan. Repeating History Beyond ARIES. In Proc. 25th VLDB, pages. 1-17, 1999.

[5] A. Schmidt, F. Waas, M. Kersten, M. Carey, I. Manolescu, and R. Busse, XMark: A Benchmark for XML Data Management, In Proc. 28th VLDB, pages 974-985, 2002.

[6] B. B. Yao, M. T. Özsu, and N. Khandelwal, "XBench Benchmark and Performance Testing of XML DBMSs", In Proc. 20th ICDE, 2004.

[7] M. Altinel and M. J. Franklin, Efficient filtering of XML documents for selective dissemination of information, In Proc. 26th VLDB, pages 53-64, 2000.

[8] Y. Diao, P. M. Fischer, M. J. Franklin, and R. To, YFilter: Efficient and scalable filtering of XML documents, In Proc. 18th ICDE, pages 341-342, 2002.

[9] C. Y. Chan, P. Felber, M. N. Garofalakis, and R. Rastogi, Efficient filtering of XML documents with XPath expressions, In Proc. 18th ICDE, pages 235-244, 2002.

[10] V. Josifovski, M. Fontoura, and A. Barta, Querying XML streams, The VLDB Journal, 2004.



**服部 雅一 Masakazu HATTORI**

1990 年 慶応技術大学工学研究科修士課程修了．同年，(株)東芝入社．現在，同社研究開発センターに勤務．XML データベースの研究開発に従事．情報処理学会会員．

**宮澤 隆幸 Takayuki MIYAZAWA**

1995 年 名古屋大学大学院工学研究科修士課程修了．同年，(株)東芝入社．現在，同社研究開発センターに勤務．XML データベースの研究開発に従事．情報処理学会，IEEE Computer Society 各会員．

**芦川 将之 Masayuki ASHIKAWA**

2001 年 早稲田大学大学院理工学研究科情報学専攻修士課程終了．同年，(株)東芝入社．現在，同社研究開発センターに勤務．XML データベースの研究開発に従事．

**神長 昭子 Akiko KAMINAGA**

2003 年 早稲田大学大学院理工学研究科電気工学専攻修士課程終了．同年，(株)東芝入社．現在，同社研究開発センターに勤務．XML データベースの研究開発に従事．

**野々村 克彦 Katsuhiko NONOMURA**

1993 年 東京工業大学理工学研究科修士課程修了．同年，(株)東芝入社．現在，同社研究開発センターに勤務．XML データベースの研究開発に従事．