リージョンディレクトリを用いた関係データベースによる大規模XMLデータ処理

Processing Large-Scale XML Data by RDBMS using Region Directory

天笠 俊之♡ 植村 俊亮♡

Toshiyuki AMAGASA Shunsuke UEMURA

本論文では，関係データベースを基盤としたXMLデータベースにおいて，大規模XMLデータの効率的な処理を実現するための手法を提案する．大規模XMLデータでは，その性質からサイズ以上にノード数が増加するため，関係データベース上での問合せ処理が効率的に行えないという問題がある．しかしながら，多くの場合，問合せ処理にXMLデータ全体は必ずしも必要ではなく，問合せに応じたデータの一部分さえあれば十分であることが多い．本稿では，このデータアクセスの局所性をうまく利用するために，ディスク上に置かれたXMLデータから，問合せ処理に必要な部分のみを関係表へ動的にマッピングする手法を考案した．これはStrong DataGuideに基づいたデータ構造であるリージョンディレクトリによって実現される．すなわち単純経路式から対応するXML部分データを知ることができ，その部分を関係表への動的にマッピング（アンマッピング）することによって，問合せ処理に不要な部分を関係表の上におく必要がなくなる．関係表のタプル数が減少するため，全体を一様にマップする従来手法より効率的な検索処理が期待できる．

This paper proposes a scheme for processing large-scale XML data efficiently by RDBMS-based XML databases. From the nature of XML data, their data size gives serious impact on the performance of query performance on RDBMS, due to the fact that the number of nodes grows quickly as the data size increases. However, in many cases, the entire XML data is not always necessary for processing XML queries, that is, only small portion is enough for answering the queries. This paper, therefore, attempts to develop a scheme for dynamically map (unmap) partial XML data to (from) relational tables that are necessary for given queries. To this end, a data structure called "region directory" is introduced. In fact, region directory is based on a well-known data structure, Strong DataGuides, that is used to compute summary of graph-structured data. A region directory enables us to know the regions of XML data related to a path expression. We can thus dynamically map (unmap) partial XML data to (from) relational tables. As a consequence, relational tables only contains relatively small numbers of tuples enough for processing particular XML queries, and this leads more efficient query processing than existing schemes.

♡ 正会員 奈良先端科学技術大学院大学情報科学研究科
{amagasa,uemura}@is.naist.jp

1. はじめに

XML [11] の応用範囲が多様化するとともに，そのデータサイズも拡大している．従来は，数KBから数MBの比較的小規模なデータを対象に用いられることが多かったが，最近では数十MBから数GBにもなる規模の大きなデータをXMLで記述する事例が出始めている．例えば，The Open Directory Project (ODP) [1] では，フリーのWebディレクトリデータをRDF/XMLフォーマットで配布しているが，そのサイズはディレクトリ構造に関する部分だけでも503 MB．コンテンツを含めると1.8 GBにもなる．このような大規模なXMLデータを効果的に運用するにはDOMやSAX等のAPIでは役者不足であり，専用のシステムをゼロから構築することを除けば，XMLデータベースを利用するのは妥当な選択の一つであると言える．

XMLデータベースを構築する方法には多くの提案がなされているが，関係データベースを用いる方法は，問合せ最適化やトランザクション管理等の技術の蓄積があることや，システムが広く普及し既存の情報資源が豊富であることなどから有力であるとされている．しかしながら，既存の手法をそのまま大規模XMLデータに対して適用することは現実的ではない．これは主にXMLの特性，すなわち木構造（ハイパーリンクを考慮するとグラフ構造）を有するため，サイズの増加に伴うノード数の増加が顕著であることに起因する．一般に，XMLデータの関係表への写像はノードを基本単位として行なわれるため，関係表に写像されるタプル数もこれに伴い増加することとなる．これに対し，構造結合等に代表されるXML問合せ処理手法 [2] は，処理に要する計算量が大きいため，結果としてデータサイズが処理性能に深刻な影響を及ぼすことになる．

この問題を解決するための手がかりの一つが，アクセスの偏り（局所性）である．一般に，ある問合せ（群）を処理するには，関連する部分データだけがあれば十分であり，必ずしもXMLデータ全体が必要なわけではない．この事実に着目した研究はすでにいくつかなされている．Marian等は，XQueryを対象に問合せを分析し，XMLデータから必要な部分だけを射影する手法を提案している [8]．また，中島等はDOMを対象として，処理に必要な部分だけを部分的に主記憶に保持するSPlitDOMを提案している [14]．横山等は，SAXパーサの高速化のために局所性を利用することを検討している [13].

そこで本論文では，XMLデータの局所性を利用した大規模XMLデータベース実現手法を提案する．提案手法では，XML部分データのマッピングを可能にするためにリージョンディレクトリを利用する．これはStrong DataGuideに基づいたデータ構造であり，任意の単純経路式からXMLデータの該当する部分（リージョン）を取得することができる．さらに，問合せエンジンとXMLマッパが連携することにより，システムに発行された問合せ処理に応じて必要な部分だけをXMLファイルから関係表にマッピングすることを可能にする．

2. システムの概要

図1にシステムの概要を示す．XMLデータは初期状態ではファイルシステム上に格納されている．XMLマッパは，前処理として対象となるXMLデータを走査し，構造概要とファイル中における各ノードの位置を抽出してメタデータを構築する．抽出されたデータはメタデータとしてデータベースに格納される．

利用者（アプリケーション）が発行した問合せは，問合せエンジンによって処理される．問合せエンジンはまず問合せを解析し，それが処理に必要なXML部分データが関係表に存在するかどうかを調べる．無い場合は，マップ処理をXMLマッパに要求する．XMLマッパは要求に従い [12, 6] 等の手法に基づき部分データを関係表にマップする．マップ処理が終了すると，問合せエンジンは [12, 6] に従って問合せ処理を行う．

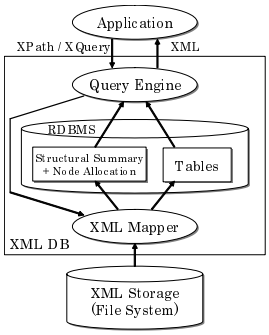

図 1: システム構成
Fig. 1: System overview.

3. XML 部分データの動的マッピング

3.1 リージョンディレクトリ

XML 部分データの関係表への動的なマッピングを可能にするためには，システムが XML ノードのファイル中での位置を常に把握している必要がある．このために，本研究では，XML データの構造概要 (structural summary) を利用し，XML ノードとその位置情報との関連付けを行う．このデータ構造を「リージョンディレクトリ」と呼ぶ．これは，UNIX や Windows (DOS) のファイルシステムにおけるディレクトリ構造に類似しているが，XML は木構造を有しており，ノード間に厳密な包含関係が成り立っているところが異なる．

3.1.1 Strong DataGuide

XML を含む半構造データを対象とした構造概要は，これまでに多くの提案がなされているが [7, 10, 4]，ここでは最も単純な Strong DataGuide [7] を用いることにする．

情報源 s の DataGuide d とは，1) s の全てのラベル経路が d において一つのデータ経路インスタンスを持ち，2) d の全てのラベル経路は s のラベル経路になっているものを言う．ここで，ノード o のラベル経路とは，o から辿ることのできるエッジ $(e_1, e_2, \ldots, e_n)$ のラベルの系列 $l_1, l_2, \ldots, l_n$ である．また，ノード o のデータ経路とは，o から辿ることのできるノードとラベルの系列 $l_1, o_1, l_2, o_2, \ldots, l_n, o_n$ である．d においては s 全ての経路が含まれているものの，s の複数のノードが d の一つのノードに集約されることから，d は s を要約していると言うことができる．

一般に，あるグラフから導出可能な DataGuide は複数存在する．紙数に限りがあるため詳細は述べないが，Strong DataGuide はその中の一つのクラスであって，情報源における全ての経路式を任意の経路式のターゲット集合という視点から，s と d が区別できないものをいう．対象とするデータが木構造の場合，Strong DataGuide の構築はグラフのサイズに対して線形の計算量で行うことができる．なお，対象を木構造に限定した場合，1-index [10], D(1)-index [4] と Strong DataGuide は等価であることが知られている．

3.1.2 リージョンディレクトリ

Strong DataGuide によって，ある経路式によって到達可能なノード集合をクラスタリングすることができる．本研究では，これを XML ノードのディレクトリとして利用する．具体的には，Strong DataGuide の各ノードから，そのノードのターゲット集合に含まれるノードのリージョンを索引付けする．本研究では，このデータ構造をリージョンディレクトリと呼ぶ．

リージョン [12] とは，そのノード (v) のデータ中での開始位置と終了位置の組 $(start(v), end(v))$ である．本論文では，これに根からの深さを加えた $(start(v), end(v), depth(v))$ を用いる．これは，ノードのファイル中での位置を特定するだけではなく，2 ノード間の先祖，子孫関係を判定する目的にも使うことができる．すなわち，

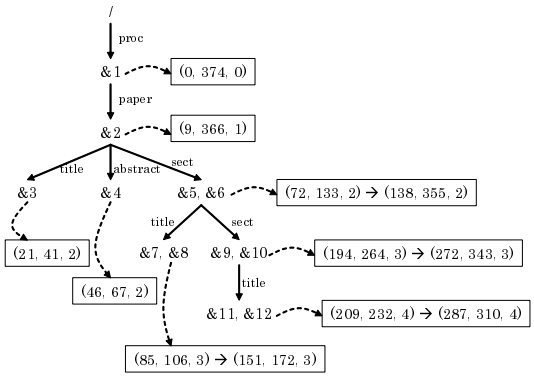

図 2: リージョンディレクトリ
Fig. 2: A region directory.

- もしノード v が v' の先祖である場合, $(start(v) < start(v')) \wedge (end(v') < end(v))$ が成り立つ
- もしノード v が v' の親である場合, $(start(v) < start(v')) \wedge (end(v') < end(v)) \wedge (depth(v) = depth(v') - 1)$ が成り立つ
- もしノード v が v' の前にある場合，$end(v) < start(v')$ が成り立つ

これは良く知られた前置順，後置順を利用したラベリングや範囲ラベリングなどの手法 [5] と同様の性質である．

リージョンディレクトリの各ノードからは，そのノードのラベル経路式に対応するリージョン索引へのポインタを配置する（図 2）．例えば，元データにおいて経路式 “/proc/paper/sect/title” にはノード &7 と&8 が対応する．このため，図 2 では, “&7, &8” のノードからポインタを辿るとこれらのノードのリージョンを得ることができる．他のノードに関しても同様である．

大規模な XML データを扱う場合，各ノードの索引部分，特に葉に近い部分には大量のリージョンが格納されるので，リンクリスト，ハッシュ，B+-tree 等を利用して索引付けを行う．また，テキストノードはリージョンディレクトリには含まれず，必要に応じてリージョンの包含関係を利用して直接ファイルから読み取られることになる．

3.2 XML 部分データの動的マッピング

3.2.1 単純経路質問の場合

利用者（アプリケーション）から発せられた問合せは，問合せエンジンによって処理される．ここでは問合せとして XPath 式を考える．問合せエンジンは，内部状態として既に関係表にマップされた XML 部分データの一覧を単純経路式として保持している．その初期値は空である．また，単純経路式とは, “/” で始まり，軸として “/” または “//” だけを含む XPath 式のサブクラスである．

問合せエンジンは受け取った質問を構文解析し，単純経路式に分解する．例えば，

$$q = //title$$

なる問合せが発行されたとする．この例では質問自身が単純経路式となっているので，単純経路式への分解は行われない．次にこの経路式がすでに関係表内に存在するかどうかを，経路式同士の包含関係で調べる．一般に XPath 式の包含関係を判定するのは困難であるが，ここでは対象を単純経路式に限定 ($XP^{//}$) しているため，多項式時間で判定を行うことができる [3, 9]．この例は初期状態であるので，問合せエンジンはただちにこの単純経路式に該当するリージョンをリージョンディレクトリ (RD) から取得する．

$$RD(//title) = \{(/proc/paper/title, \{(21, 41, 2)\}),$$

(/*proc*/*paper*/*sect*/*title*,
{(85, 106, 3), (151, 172, 3)}),
(/*proc*/*paper*/*sect*/*sect*/*title*,
{(209, 232, 4), (287, 310, 4))}

この結果は各ノードの根からの経路式とともに返却され，経路式の値ごとにグルーピングされている．これは XML マッパが XML データ関係表にマップする際，現在処理中の部分データの根からの絶対経路式を知るために必要である．

XML マッパは該当する部分文書を XML ファイルから読み込み，関係表へとマップする．このとき，このリージョンに含まれる全ての子孫ノードに関しても同様に関係表へのマッピングを行う．この例の場合は，各 title 要素に含まれる全てのテキストノードもその対象となる．この処理によって，問合せ q を処理するのに必要な XML データは全て関係表へマップされるので，XML データ全体が関係表の上に存在しなくても q に関しては処理を行うことができる[1]．

XML マッパは，内部状態としてマップ済みの領域を図 3 の形で保持している．各列は左から順にマップされた時刻を示すタイムスタンプ，単純経路式，マッピングの種類（後述），マップされたノード数，マップされた領域の長さの合計，参照頻度，最後に参照された時刻である．

3.2.2 単純経路質問でない場合

問合せが述語を含むなど単純経路質問でない場合は，より複雑な処理が必要である．例えば，

$$q' = //paper[title = \text{“}title\text{”}]//\text{sect/title}$$

なる質問を考える．問合せエンジンはこれを構文解析して，問合せ木 [12] を得る．ここで，*title* ノードは最終的に解として出力されるノードなので，出力ノードと呼ばれる．この問合せ木を処理するのに必要なのは以下のノードである．

- 出力ノード: //paper//sect/title
  この例では葉になっているが，一般に出力ノードは葉にあるとは限らない．
- 出力ノード以外の全ての葉ノード: //paper/title
- 分岐ノード: //paper
  出次数が 2 以上の中間ノード．すなわち分岐点になっているノードである．

これらは以下の二つの場合に分けて考える．

出力ノードおよび葉ノードのマッピング 出力ノードおよび葉ノードに関しては，述語の判定や結果の出力などにそのノードと子孫ノードの内容の両方を使う可能性が高いことから，3.2.1 節で行ったのと同様に，リージョンに含まれる全てのノードを関係表へマッピングする．リージョン全域を再帰的にマップするのでこれを「深いマッピング (deep mapping)」と呼ぶ．

この例では，//paper//sect/title と //paper/title が深いマッピングの対象となる．問合せエンジンはそれぞれの単純経路式のマッピングの要求を XML マッパに対して行う．XML マッパはマップ済みの領域とこれらを比較し，もしマップされていない領域であればマッピングの処理を行う．この場合，両方とも //title に含まれているので，実際のマッピング処理は行われず，参照頻度と最終参照時刻の更新だけが行われる．

分岐ノードのマッピング 分岐点になっている中間ノードに関しては，問合せ処理にその内容は使われず，出力ノードと葉ノードが共通の祖先を持っているかどうか，換言すれば実際のデータ上で両者に関連があるかどうかをチェックするためだけに必要となる．このため，全域を関係表にマップするのは必ずしも効率的であるとは言えない．この例の場合，paper 要素が分岐点になっているが，paper 要素をマップすることはすなわちデータのほぼ全域を関係表にマップすることになってしまい，データの局所性を利用することにつながらないからである．

このため，分岐ノードに関してはそのノードだけを関係表にマップする．これを「浅いマッピング (shallow mapping)」と呼ぶ．この処理が終わった時点で，XML マッパの内部状態は図 3 (II) のようになる．

3.3 部分データのアンマップ

アクセスの局所性は時間に関しても考慮する必要がある．すなわち，関係表にマップされた部分データの内，アクセス頻度の少ないものは積極的に関係表から落としてやることによって，システム全体のパフォーマンスの向上が期待できる．

このために XML マッパの内部状態表の参照頻度，参照時刻等を利用する．具体的には，

- 参照頻度がある閾値よりも低い部分データに関しては関係表から別の一時表へ待避させる
- 一定時間アクセスのなかった部分データに関しては関係表から別の一時表へと待避させる

などの方針が考えられる．これはシステムに入力される問合せの傾向などから総合的に判断することになるが，細かいパラメータの設定や最適化の方法などは今後の課題である．

4. 予備実験

提案手法が有効かどうかを検証するために予備実験を行った．実験は，Sun Microsystems Sun Blade 150 (CPU: UltraSPARC-IIe 550MHz, memory: 1GB, disk: 40GB E-IDE (7,200rpm)) 上で行い，関係データベースには PostgreSQL-7.4 を用いた．データセットには，XMark (http://monetdb.cwi.nl/xml/index.html) で生成した合成 XML データを用いた，具体的には，scale factor を 0.001, 0.01, 0.1 とした．実ファイルサイズはそれぞれ 100KB, 1MB, 10MB である．

まず，XML データの部分的なマッピングがどの程度有効であるかを調べた（図 4）．横軸は scale factor, 縦軸は関係表にマップされるノードの数およびデータサイズである．XML データ全体を関係表にマップする場合，ファイルサイズに応じてノード数も増加している (図中 full)．これに対して，問合せ (`/site/people/person[@id="person10*"]/name`) の処理に必要な部分にマッピングの対象を絞ると処理に必要なノード数は激減することが分かる (図中 proj)．

このデータに対して，実際に問合せ処理を行った結果が図 5 である．部分マッピングを行ったことにより，関係表のサイズがコンパクトになり，同じ問合せであってもより高速に処理を行うことができることがわかる．実際には，問合せの解析およびマッピングの処理によるオーバーヘッドがかかるため，これほどの差は出ないが，問合せの分布に偏りがある場合には提案手法は有効であると予想される．

5. まとめ

本論文では，関係データベースを基盤とした大規模 XML データベース構築の一手法として，データアクセスの局所性に着目した XML 部分データのマッピング手法を提案した．これを可能にするために Strong DataGuide に基づいたリージョンディレクトリを導入した．これにより任意の単純経路式から XML データの該当する部分（リージョン）を取得することができる．さらに，問合せエンジンと XML マッパが連携することにより，システムに発行された問合せ処理に応じて必要な部分だけを XML ファイルから関係表にマッピングすることが可能となった．また，提案手

[1] XML データの関係表へのマッピング手法そのものについては，この論文の議論の範囲を超えるのでここでは述べない．

マップ状態表 (I)

No.	path exp.	mode	# of nodes	total length	freq	last referenced time
1	//title	deep	5	113	1	2004/07/01 23:34:45

マップ状態表 (II)

No.	path exp.	mode	# of nodes	total length	freq	last referenced time
1	//title	deep	5	113	3	2004/07/01 23:40:00
2	//paper	shallow	1	855	1	2004/07/01 23:40:01

図 3: マップ状態表
Fig. 3: Mapping-status tables.

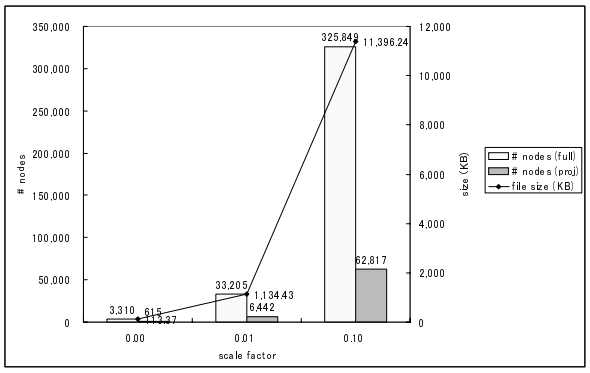

図 4: データサイズ
Fig. 4: data size.

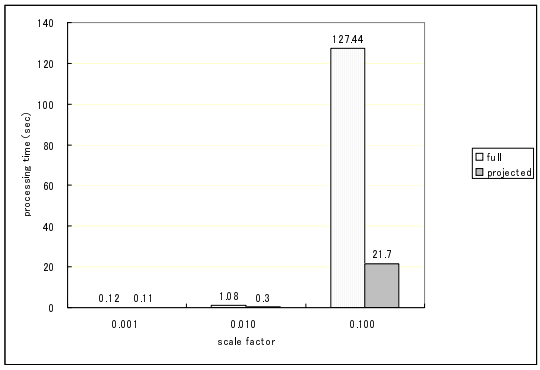

図 5: 処理時間
Fig. 5: processing time.

法の有効性を予備実験によって検証した.

今後はシステムの実装と性能評価，更新のサポート，より複雑な XPath 問合せや XQuery への対応などに関して研究を進める予定である

[謝辞]

本研究の一部は，文部科学省科学研究費補助金（課題番号 15017243），日本学術振興会科学研究費補助金（課題番号 15700097）の支援によるものである．ここに記して謝意を表す.

[文献]

[1] The Open Directory Project (ODP). http://dmoz.org/.

[2] Shurug Al-Khalifa, H. V. Jagadish, Jignesh M. Patel, Yuqing Wu, Nick Koudas, and Divesh Srivastava. Structural joins: A primitive for efficient xml query pattern matching. In *Proc. ICDE 2002*, pp. 141–152, 2002.

[3] Sihem Amer-Yahia, SungRan Cho, Laks V. S. Lakshmanan, and Divesh Srivastava. Minimization of tree pattern queries. In *Proc. SIGMOD 2001*, 2001.

[4] Qun Chen, Andrew Lim, and Kian Win Ong. D(k)-index: an adaptive structural summary for graph-structured data. In *Proc. SIGMOD 2003*, pp. 134–144, 2003.

[5] Edith Cohen, Haim Kaplan, and Tova Milo. Labeling dynamic XML trees. In *Proc. PODS 2002*, pp. 271–281, 2002.

[6] Daniela Florescu and Donald Kossmann. Storing and querying XML data using an RDMBS. *IEEE Data Engineering Bulletin*, Vol. 22, No. 3, pp. 27–34, 1999.

[7] Roy Goldman and Jennifer Widom. DataGuides: Enabling query formulation and optimization in semistructured databases. In *Proc. VLDB 1997*, pp. 436–445, 1997.

[8] Amélie Marian and Jérôme Siméon. Projecting XML documents. In *Proc. VLDB 2003*, pp. 213–224, 2003.

[9] Gerome Miklau and Dan Suciu. Containment and equivalence for an XPath fragment. In *Proc. PODS 2002*, pp. 65–76, 2002.

[10] Tova Milo and Dan Suciu. Index structures for path expressions. In *Proc. ICDT 1999*, pp. 277–295, 1999.

[11] World Wide Web Consortium. Extensible Markup Language (XML) 1.0 (Third Edition). http://www.w3.org/TR/REC-xml. W3C Recommendation 04 February 2004.

[12] Masatoshi Yoshikawa, Toshiyuki Amagasa, Takeyuki Shimura, and Shunsuke Uemura. XRel: A path-based approach to storage and retrieval of XML documents using relational databases. *ACM Transactions on Internet Technology (TOIT)*, Vol. 1, No. 1, pp. 110–141, June 2001.

[13] 横山昌平, 太田学, 片山薫, 石川博. SAX-GTR: 高速 XML ストリーム読み込み手法. 電子情報通信学会技術研究報告 DE2004-10〜47, pp. 169–174, July 2004.

[14] 中島哲, 小田切淳一, 井谷宣子, 吉田茂. XML 高速処理技術 SPlitDOM の機能拡張と web アプリケーションへの適用評価. 第 15 回データ工学ワークショップ (DEWS2004), March 2004.

天笠 俊之　Toshiyuki AMAGASA
奈良先端科学技術大学院大学情報科学研究科助手．データベースシステムの研究に従事．情報処理学会正会員．電子情報通信学会正会員．日本データベース学会正会員.

植村 俊亮　Shunsuke UEMURA
奈良先端科学技術大学院大学情報科学研究科教授．データベースシステムの研究に従事．情報処理学会フェロー．電子情報通信学会フェロー．IEEE Fellow. 日本データベース学会正会員．著書に「データベースシステムの基礎」（オーム社）など.