論文 / 著書情報
Article / Book Information

論題(和文)	DO-VLEIを用いた XML 格納におけるラベルサイズと問い合わせ性能
Title(English)	The Label Size and Query Performance of XML Storage using the DO-VLEI
著者(和文)	村上翔一, 小林大, 横田治夫
Authors(English)	Shoichi MURAKAMI, Dai KOBAYASHI, Haruo YOKOTA
掲載誌(和文)	DEWS2006 論文集
Citation(English)	Proc. of DEWS2006
Vol, no, pages	Vol. , No. , pp. 7B-o2
発行日 / Pub. date	2006, 3
URL	http://search.ieice.org/
権利情報 / Copyright	本著作物の著作権は電子情報通信学会に帰属します。 Copyright (c) 2006 Institute of Electronics, Information and Communication Engineers.

DO-VLEIを用いたXML格納におけるラベルサイズと問い合わせ性能

村上　翔一[†]　　小林　　大[††]　　横田　治夫[†††,††]

† 東京工業大学 工学部 情報工学科
†† 東京工業大学 大学院 情報理工学研究科 計算工学専攻
††† 東京工業大学 学術国際情報センター
E-mail: †sho@de.cs.titech.ac.jp, ††daik@de.cs.titech.ac.jp, †††yokota@cs.titech.ac.jp

あらまし　RDBにXML文書を格納することでRDBMSの様々な機能が容易に利用可能となるため，XML文書の各ノードへラベル付けを行い，RDBへ格納するための手法が注目されている．単純な数値を用いたラベリング手法では，更新処理に伴い大規模なラベル付け替えが発生し，更新コストが高くなる．我々は，更新コストを抑え無制限な挿入ができるVLEIコードと，一つのラベルで包含関係を表せるDeweyOrderとを組み合わせたDO-VLEIを提案し，更新処理と問い合わせ処理の向上を図ってきた．本稿では，DO-VLEIを用いて，ラベルサイズを縮小する手法と，問い合わせ処理性能を向上させる手法を提案する．そして提案手法を用いたDO-VLEIと，O'Neilらによって提案された同様の性質を持つORDPATHとの比較評価を行い，ラベルの格納に必要な容量と，問い合わせ処理時間における実験結果を報告する．

キーワード　XML, 問い合わせ処理, 性能評価, ラベリング

The Label Size and Query Performance of XML Storage using the DO-VLEI

Shoichi MURAKAMI[†], Dai KOBAYASHI[††], and Haruo YOKOTA[†††,††]

† Department of Computer Science, Faculty of Engineering, Tokyo Institute of Technology
†† Department of Computer Science, Graduate School of Information Science and Engineering, Tokyo Institute of Technology
††† Global Scientific Information & Computing Center, Tokyo Institute of Technology
E-mail: †sho@de.cs.titech.ac.jp, ††daik@de.cs.titech.ac.jp, †††yokota@cs.titech.ac.jp

Abstract　Techniques labeling each node in an XML document are paid to attention, because various functions in RDBMS can be used by storing the labeled XML document into an RDB. However, labels using simple numerical values make XML document update expensive by large-scale relabeling. We proposed the Variable Length Endless Insertable (VLEI) code to reduce the update cost, and the DO-VLEI method by combining the VLEI code with the Dewey Order method capable of expressing ancestor-descendant relationship by using a single label, to realize efficient XML query processing. In this paper, we propose compression and filtering methods for the DO-VLEI to reduce the size of labels and improve the query processing performance. We compare the label size and query processing performance of the proposed methods with the ORDPATH, a similar method proposed by O'Neil. The experimental results indicate that the DO-VLEI is superior to the ORDPATH for both the size and performance.

Key words　XML, query processing, performance evaluation, labeling

1. はじめに

大量のXML文書を効率よく管理し，高速に検索するために，XML文書をRDBに格納する方法が注目されている[1]．XML文書は，タグによって表現された要素間に包含関係があり，一組のタグをノードとして扱うことで木構造と見なすことができる[2]．図1にXML文書を木構造とみなしたXML木を示す．RDBに格納する時に，このXML文書の木構造を維持するための方法として，XML文書の各ノードにラベルを割り当て，ラベルとノードをタプルとして格納する方法がある．ラベルの規則性により，各ノード間の包含関係や，ノードのXML木における位置を判定することができる．

このようなラベル付け手法として，以前から前順後順法[3]やDewey Order[4]が提案されている．これらの手法をXML木

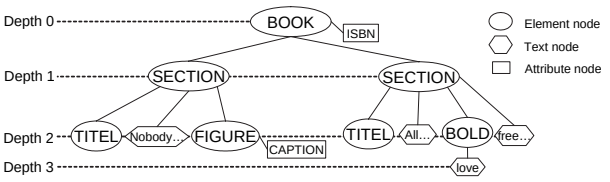

図 1　XML　木

のノードに適用することで，ラベルにより XML 文書の木構造を表現することが可能となる．しかし，それらのラベル付けに単純な連続する整数を用いると，更新に伴って大規模なラベルの付け替えが必要となる．このため，更新時のラベル付け替えのコストを抑える手法として，浮動小数点数法 [5] や，範囲ラベリング法 [6]，範囲ラベリング法を改良した更新に強い節点ラベル付け手法 [2]，などが提案されている．浮動小数点数法 [5] では理論上は無制限に浮動小数を作り出せるが，コンピュータ内部のレジスタで表現できる浮動小数には上限が存在するため，この上限を超えれば大規模なラベルの付け替えが発生する．範囲ラベリング法を改良した手法 [2] では，小規模なラベル付け替えを頻繁に行うことで大規模なラベルの付け替えを防ぐが，ラベル付け替えが発生することには変わりがない．

このようなラベル付け替えが発生する問題に対して，我々は他ノードのラベルを変更することなく無制限に挿入することができる VLEI コード (Variable Length Endless Insertable code) と，VLEI コードを用いたラベル付け手法を提案し，更新・検索処理に関する改良を行ってきた [7]〜[9]. [7], [8] では VLEI コードを前順後順法に適用した PP-VLEI と，Dewey Order に適用した DO-VLEI を提案し，簡単な検索と挿入の実験によって，PP-VLEI，DO-VLEI と，浮動小数点数法 [6]，範囲ラベリング法の改良手法 [2] との比較を行い，VLEI コードを用いたラベル付け手法の有効性を示した. [9] では，包含関係の判定において，DO-VLEI の子孫ノードのラベルに先祖ノードのラベルが含まれるために子孫ノードのラベルは必ず先祖ノードのラベルよりもラベルが長いという特徴を活用することで，検索処理の性能改善を行った．

このように，DO-VLEI を用いて更新・検索処理の向上を図ったが，ラベルサイズに関しては対策を行っていない．DO-VLEI の子孫ノードのラベルが先祖ノードのラベルを含むという特徴により，XML 木において，深いところにあるノードほどそのラベルサイズ (ラベルの長さ) は大きくなる傾向にある．その結果，ラベル格納に必要な記憶容量が増大し，ラベル処理の時間も増大するために大規模な XML 文書の処理性能が低下する．なお，ノードの深さは「XML 木のルートノードの深さを 0 とし，子ノードの深さは親ノードの深さより 1 大きい」と定義する．また, [9] で提案したラベルサイズによる絞り込み手法は，先祖子孫関係を示す包含関係に関する問い合わせを効率よく行うための手法である．しかし，絞り込みに用いるラベルサイズが変動しやすく，安定した絞り込みを行うことができないという問題がある．

本稿ではラベルサイズを縮小するために，DO-VLEI をデリミタも含めた内部的な bit 列により表現する圧縮 bit 列 DO-VLEI を提案する．また，より安定した絞り込み手法として，ラベルサイズではなくラベル付けされたノードの深さを絞り込みに用いる手法を提案する．

表 1　アルゴリズム： 挿入要素のコードの決定

アルゴリズム InsertVLEI(v_l, v_r)
入力: 挿入する要素の左側のコード v_l, 右側のコード v_r (但し $v_l < v_r$)
出力: 挿入された要素のコード v_i

if $length(v_l) \leqq length(v_r)$
$v_i = v_r \cdot 0$
else
$v_i = v_l \cdot 1$
endif
return v_i

本稿の構成は次の通りである．まず，2 節で従来の DO-VLEI について説明し，3 節では関連研究として DO-VLEI と同様な問題解決案として提案された手法について説明する．次に 4 節では我々が新しく提案する圧縮 bit 列 DO-VLEI について説明する．そして 5 節では圧縮 bit 列 DO-VLEI のラベルサイズに関する効果を実験によって示す．さらに 6 節ではノードの存在する深さを利用した絞り込み手法を提案し，7 節では提案手法を用いた絞り込みの効果を測定する．さらにラベルから深さを抽出する字句解析について言及し，実際に字句解析処理の時間を測定しその結果について報告する．最後に 8 節で総括を述べる．

2. DO-VLEI

我々がこれまで提案してきた VLEI コードと，VLEI コードに Dewey Order を組み合わせた DO-VLEI について説明する．

2.1 VLEI コードの定義

VLEI コードは 1 から始まる 0,1 の可変長 bit 列であり，特殊な大小関係を持つ．以下に VLEI コードの定義を示す．

定義 1. VLEI コード

v を 1 から始まる {0,1} からなる bit 列とし，以下の大小関係を満足する場合に VLEI コードと呼ぶ．

$$v \cdot 0 \cdot \{0|1\}^* < v < v \cdot 1 \cdot \{0|1\}^*$$

すなわち，あるラベル v に対して，右に 0 のつくものは v よりも小さく, 1 のつくものは v よりも大きい．

表 1 に示すアルゴリズムにより，任意の 2 つの VLEI コードの間に存在する新しい VLEI コードを，無制限に作り出すことができる．証明は論文 [7] [8] を参照されたい．

したがって，木構造において挿入されるノードのラベルを容易に求めることができ，また他ノードのラベルを付け替える必要がないため，更新コストの低いラベル付けが可能となる．

2.2 DO-VLEI

Dewey Order では，親ノードのラベルの末尾に，デリミタ (区切り文字) と兄弟ノード間の順序を表現するコード (兄弟コード) とを加えて，子ノードのラベルを表現する．DO-VLEI では兄弟コードに VLEI コードを用いる．

定義 2. DO-VLEI

（1） root ノードの DO-VLEI を 1 とする．

（2） 兄弟ノード間の順序を VLEI コードの大小関係で表し，これを C_{child} とする．このとき，親ノードの DO-VLEI を L_{parent} とすると，

$$\text{DO-VLEI} = L_{parent} \cdot C_{child}$$

となる．

図 2 に DO-VLEI によってラベル付けされた XML 木を示す．

DO-VLEI ラベルでは，特定ノードのラベルを接頭辞としたラベルを探すことでそのノードの子孫ノードを探すことができる．また，ラベルの右から n 個目のデリミタから，デリミタを含めた右側の文字列を除けば，n 番目の先祖ラベルが得られる．

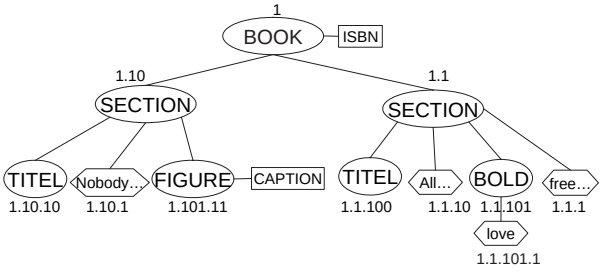

図 2　DO-VLEI ラベルによるラベル付け

3. 関 連 研 究

DO-VLEI 同様に，他ノードのラベルを付け替えることなく無制限にノードの挿入を行うことのできる，新しいラベル付け手法がいくつか提案されている．素数を用いた手法 [10] は素数の積によってラベル付けを行い，挿入時には未使用の素数と親ラベルの積をとることで新しいラベルを作る．そして素因数分解を行うことで，その先祖ラベルを抽出することができる．また，O'Niel らによって提案された ORDPATH [11] では，Dewey Order を基本に，初期のラベル付けには奇数を用いて，挿入には偶数を用いる．そして奇数と偶数の区別によって先祖ラベルを抽出する．本節では以下，ORDPATH について簡単に説明する．詳しくは [11] を参照されたい．

3.1 ORDPATH の概要

ORDPATH は，DO-VLEI ラベルと同様に，他のラベルを付け替えることなく挿入を無制限に行うことのできるラベル付けである．Dewey Order を基本とするため DO-VLEI 同様に，親ノードの ORDPATH に，デリミタと兄弟コードを付け加えることで，新しい ORDPATH を作ることができる．例えば，親ノードの ORDPATH が 1.3 であるとき，1.3.1 や 1.3.7 はその子ノードであり，兄弟コードは 1 と 7 である．兄弟コードを比較すると，1.3.1 の後に 1.3.7 があることが分かる．図 3 に ORDPATH によってラベル付けされた XML 木を示す．

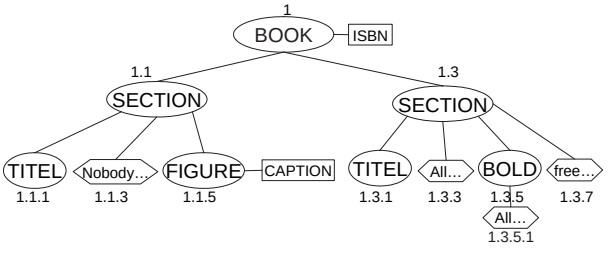

図 3　ORDPATH によるラベル付け

表 2　圧縮 bit 列 ORDPATH で用いる接頭辞スキーマ

L_i Bitstring	O_i length	O_i value range
000000001	20	[-1118485, -69910]
00000001	16	[-69909, -4374]
0000001	12	[-4373, -278]
000001	8	[-277, -22]
00001	4	[-21, -6]
0001	2	[-5, -2]
001	1	[-1, 0]
01	0	[1, 1]
10	1	[2, 3]
110	2	[4, 7]
1110	4	[8, 23]
11110	8	[24, 279]
111110	12	[280, 4375]
1111110	16	[4376, 69911]
11111110	20	[699912, 1118487]

3.2 ORDPATH におけるノードの挿入

ORDPATH は初期のラベル付けにおいて，正の奇数のみを用いている．これは，負の整数や偶数を，後で XML 木に挿入が行われたときに用いるためである．また兄弟順を決定する兄弟コードは必ず奇数で終了する．例えば，1.3.1 と 1.3.3 の間に新しく挿入されるノードの ORDPATH は 1.3.2 のあとに，1 を加えた，1.3.2.1 となる．

3.3 圧縮 bit 列 ORDPATH

ORDPATH は内部では 0,1 の bit 列表現により実装される．これを圧縮 bit 列 ORDPATH と呼ぶこととする．表 2 のような接頭辞スキーマを用いて，整数を bit 列に変換する．一つの整数を表すために，一組の bit 列コンポーネント L_i-O_i を用いる．i はデリミタで区切られた整数のラベル中における順番である．L_i は頭から逐次解析することで決定する bit 列であり，続く O_i の bit 数と整数の範囲を接頭辞スキーマにより決定する．そして O_i は L_i によって指定される範囲の数を 2 進数によって表す．例えば圧縮 bit 列 ORDPATH(0111001) は，表 2 の接頭辞スキーマを用いた逐次解析により，(01,110-01) に分割され，表 2 の対応する O_i value range を基に「1」と「5」に変換されて ORDPATH「1.5」が得られる．

圧縮 bit 列 ORDPATH は bit 列を接頭辞スキーマに従って区切ることで，デリミタを表現し，ラベルサイズの圧縮を行っている．図 4 に，圧縮 bit 列 ORDPATH によるラベル付けの例を示す．

4. 圧縮 bit 列 DO-VLEI

Dewey Order を基にしたラベル付け手法では，扱う XML 文書の規模が増すにつれ，ラベル表現のための格納容量が増加す

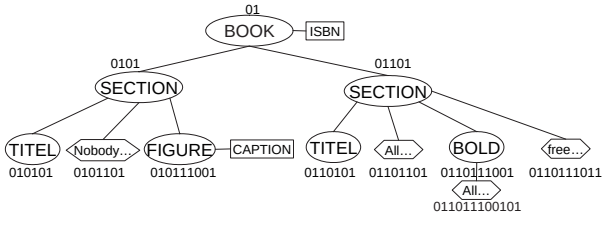

図 4　圧縮 bit 列 ORDPATH によるラベル付け

るため，ラベルサイズをできるだけ小さくすることが重要である．我々が新しく提案する圧縮 bit 列 DO-VLEI は，ラベルが内部的に bit 列で表されることを利用し，DO-VLEI をデリミタも含めて全て bit 列によって表現することで，ラベルサイズの縮小を行う．

4.1 DO-VLEI の構成要素

DO-VLEI は「.」と「1」と「0」の三つの記号から構成される文字列である．そして，三つの記号から構成される文字列が DO-VLEI であるためにはいくつかの条件が存在する．

（1）「.」は連続しない．

（2）「.」がラベルの末尾にあることはない．

（3） VLEI コードは「1」で始まる．

条件 (1)(2) から「.」の後には「1」か「0」が存在し，条件 (3) から「.」の後には必ず VLEI コードの先頭の文字「1」が存在する．ゆえに,「.」と「1」をセットとして扱い，DO-VLEI は「.1」と「1」と「0」の三つの記号から構成されると言い換えることができる．

4.2 圧縮 bit 列 DO-VLEI の定義

以下に圧縮 bit 列 DO-VLEI の定義を示す．

定義 3. 圧縮 bit 列 DO-VLEI

DO-VLEI において,「.1」を bit 列 (11) に,「1」を bit 列 (10) に,「0」を bit 列 (0) に変換したものを，圧縮 bit 列 DO-VLEI と呼ぶ．

三つの記号「.1」「1」「0」それぞれに，語頭符号 [12] となる bit 列 (0)，(10)，(11) を割り当て，bit 列に符号化する．語頭符号は互いが互いの接頭辞 (語頭) とはならないため，任意の符号語から成る列に対して先頭から 1bit づつ逐次的に解析していくことで個々の符号語に一意に分離できる．したがって，分離されたそれぞれの符号語を対応する記号に変換すれば一意の DO-VLEI へと復元ができる．

次に，三つの記号にどの符号語を割り当てるかが問題となる．特に符号語 (0) は，1bit で表現されるため，ラベルサイズをできる限り小さくするためにはもっとも出現頻度の高い記号に割り当てる必要がある．まず，挿入が行われると兄弟コードである VLEI コードの長さが増大し，デリミタの個数が増大することは少ないため，デリミタが含まれる記号「.1」の出現率は低いと考える．ゆえに (0) に割り当てる候補として「.1」は除外する．insertVLEI アルゴリズム (表 1) では，比較するラベルサイズが等しいときに「0」付け加えることになるため，挿入時には「0」の出現率が高くなると予想する．したがって「0」に (0) を割り当てることにする．

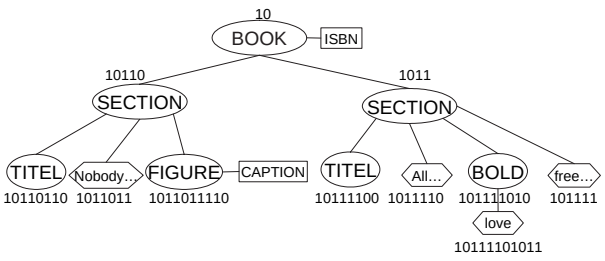

図 5　圧縮 bit 列 DO-VLEI によるラベル付け

(11) と (10) にラベルサイズの差異はない．しかし，連続する (1) の数を数えることで DO-VLEI へ変換することなくデリミタを抽出することができる．ゆえに「.1」に (11) を,「1」に (10) を割り当てる．図 5 に，圧縮 bit 列 DO-VLEI でラベル付けした XML 木を示す．

5. ラベルサイズ比較実験

圧縮 bit 列 DO-VLEI と圧縮 bit 列 ORDPATH を用いて実際に XML 文書にラベル付けを行い RDB に格納する．ここでは，それぞれの手法によって付けられるラベルの長さの総和を求め，比較する．

5.1 使用する XML 文書のデータ構造

XMark [13] は XML 文書における代表的なベンチマークとして知られる．XMark 用の XML 文書生成プログラムである xmlgen [14] から生成された XML 文書を実験に用いる．xmlgen では文書の規模を表す scale factor を設定することで，構造の定まった任意の規模の XML 文書を生成できる．図 6 に，この XML 文書の基本的な構造を示す．scale factor を大きくすると，図 6 にある ∗ の付いた項目の数が増える．

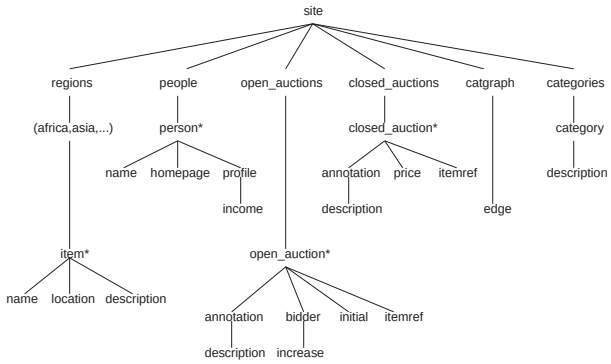

図 6　XMark で用いる auction.xml の構造

5.2 ラベル付け

XML 文書を DOM (Document Object Model) 木として取り込み，要素ノードテーブル，属性ノードテーブル，テキストノードテーブルの 3 種類のテーブルに分けて RDB に格納する．用いる 3 種類のテーブルのスキーマは表 3 のようになる．ラベル付けは XML 文書の要素ノードとテキストノードに対して行い，属性ノードへのラベル付けは行わない．一つの要素に付随する複数の属性間には順番が存在しないため，どの要素に付随しているかのみが重要である．

圧縮 bit 列 DO-VLEI と圧縮 bit 列 ORDPATH の 2 種類のラベル付け手法によりラベル付けを行う．これらのラベル付け手法では，ルートノードのラベルを除く全てのラベルが，その親ラベルを接頭辞として持っているため，全てのラベルはルートノードのラベルを接頭辞として持つ．ゆえに，ルートノードのラベルは省略する．実際に RDB に格納されるラベルの例として，図 1 の要素ノード BOLD につけるラベルを表 4 に挙げる．図 2〜5 を併せて参照されたい．

5.3 挿入操作

挿入操作によるラベルサイズへの影響は重要であると考える．挿入のパターンには次の 2 つのパターンを仮定する．(i) 兄弟

表 3　RDB の各テーブルスキーマ

要素ノードテーブル

id	INTEGER	自分の id
parent	INTEGER	親要素ノードの id
label	TEXT	ノード固有のラベル
length	INTEGER	label の長さ (ラベルサイズ)
depth	INTEGER	デリミタの数 (DO-VLEI), 奇数の数 (ORDPATH)
name	TEXT	要素名

テキストノードテーブル

id	INTEGER	自分の id
elementid	INTEGER	親要素ノードの id
label	TEXT	ノード固有のラベル
length	INTEGER	label の長さ (ラベルサイズ)
depth	INTEGER	デリミタの数 (DO-VLEI), 奇数の数 (ORDPATH)
content	TEXT	テキストの内容

属性ノードテーブル

elementid	INTEGER	親要素ノードの id
name	TEXT	属性名
value	TEXT	属性値

表 4　実験で用いるラベル付けの例

ラベル付け手法	格納されるラベル
圧縮 bit 列 DO-VLEI	1111010
(参考) DO-VLEI	.1.101
圧縮 bit 列 ORDPAHT	10111001
(参考) ORDPATH	.3.5

順序の末尾に挿入する場合 (あるいは先頭に挿入する場合). (ii) 兄弟間に均等に挿入する場合.

さらにそれぞれのパターンに 2 つのパターンを仮定する. (a) 一つの要素ずつ挿入する場合. (b) 部分木の塊で挿入する場合.

本稿では (ii) のように兄弟間に均等に，かつ，(b) のように部分木の塊で挿入する場合を考える．このような挿入パターンが起こり得る実例としては，広くインターネット上で活用される wiki [15] などがある．wiki は閲覧者が自由に情報を追加することのできるシステムであり，文書中に情報の追加が頻繁に行われる．文書としての構造を維持するため，XML 文書の同じ深さに偏りなく挿入が行われやすい．

図 6 のようなデータ構造において，データの更新時には * のついている要素を追加するのが自然である．実験ではこれら * のついた要素ノードをルートノードとした XML 文書を用意し，これを部分木と見立て，部分木ごと挿入する．挿入される部分木のルート要素名は，* のついたものと同じである．偏りの少ない挿入を実現するため，複数存在する * のついたノード間全てに挿入を行うことを，一回の挿入とみなす．例えば，item 要素が 5 個並んでいれば，それぞれの item 要素間とその前後の計 6 箇所それぞれに，item 要素をルートとする部分木を挿入する．

5.4　ラベルサイズ比較実験結果

初期状態から挿入を何回か行った状態までのラベルサイズ総和を測定した．図 7 は圧縮 bit 列 ORDPATH のラベルサイズを 100%としたときに，圧縮 bit 列 DO-VLEI のラベルサイズ比率が挿入に伴いどのように変化するか表したグラフである．

初期状態 (挿入回数 0) においては圧縮 bit 列 DO-VLEI は圧縮 bit 列 ORDPATH の 80%以下のラベルサイズであることが分かる．圧縮 bit 列 ORDPATH が初期状態では偶数と負の整数によるラベル付けを行わないために，特定のラベルサイズで表現できるラベルの数が少なくなり，全体のラベルサイズが増大するためと考える．例えば，5bit での表現可能ラベル数は表 5 のようになる．圧縮 bit 列 DO － VLEI では，5 個がラベルとして成立する．一方圧縮 bit 列 ORDPATH では 8 個がラベルとして成立するが，このうち偶数を含むラベルは初期状態のラベル付けでは使われないため，実際に初期のラベル付けで使われるラベルは 4 個である．ラベルの数が一定であれば，そのラベルサイズ比は 4:5 となると推察する．

表 5　5*bit* で表現可能なラベル

圧縮 bit 列 DO-VLEI	圧縮 bit 列 ORDPATH
(11000)=「.1000」	(01100)=「.1.2」**(01101)=「.1.3」**
(11010)=「.101」	(10001)=「.2.1」**(10101)=「.3.1」**
(11011)=「.10.1」	(11000)=「.4」　**(11001)=「.5」**
(11100)=「.110」	(11010)=「.6」　**(11011)=「.7」**
(11110)=「.1.10」	

挿入を行うと，徐々に圧縮 bit 列 DO-VLEI と圧縮 bit 列 ORDPATH の比率が増大する．すなわち，挿入に対して圧縮 bit 列 DO-VLEI の方がラベルサイズの増加量が大きい．圧縮 bit 列 DO-VLEI では初期状態において，bit 数に対して最適なラベルサイズを与えるようなラベル付けを施す．このとき，連続する兄弟ノードの一つおきに最長のラベルサイズを持つ兄弟ノードが存在する．全ての兄弟ノード間に挿入を行うと，表 1 の挿入アルゴリズムから必ず兄弟内の最長ラベルサイズよりも長いラベルが付されたノードを挿入することになる．ゆえに，圧縮 bit 列 DO-VLEI で挿入される部分木ルートノードのラベルは既存の兄弟ノードらよりも必ず 1〜2bit 増大する．一方，圧縮 bit 列 ORDPATH では，初期状態で未使用の偶数や負の整数を用いたラベルを挿入時に使うため，ラベルサイズの増加量は圧縮 bit 列 DO-VLEI と比較して小さい．

図 7 に記してある挿入回数以降の，ラベルサイズ比率の変化が挿入回数に対して比例であると仮定して, ラベルサイズ比率が 100%になるとき，すなわち，ラベルサイズが等しくなるときの挿入回数とデータ規模を試算した．結果を表 6 に示す．結果から，実験に使用した scale factor において最小でも 59 回の

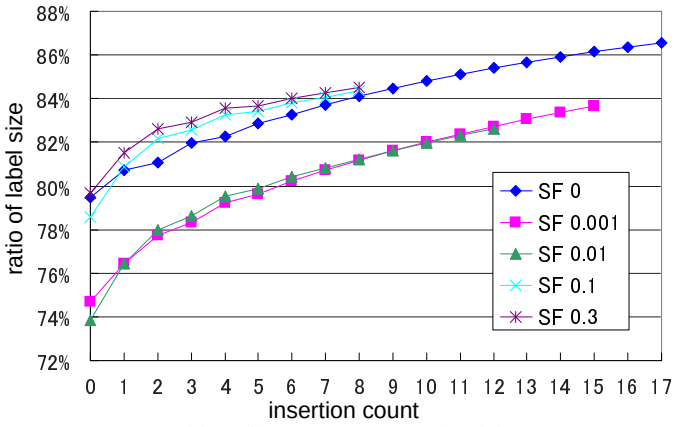

図 7　挿入に伴うラベルサイズ比率の変化

表 6　ラベルサイズが等しくなるときの挿入回数とデータ規模

scale factor	挿入回数	データ規模
0	86	2.97×10^{30} B (2.97×10^{21} GB)
0.001	73	1.46×10^{27} B (1.46×10^{18} GB)
0.01	59	3.21×10^{25} B (3.21×10^{16} GB)
0.1	63	1.27×10^{26} B (1.27×10^{17} GB)
0.3	67	1.53×10^{27} B (1.53×10^{18} GB)

挿入回数でデータ規模にして 32yotta byte にもなることが分かる．これは，現在のデータベースにおいて現実的なデータ規模ではないので，圧縮 bit 列 DO-VLEI は圧縮 bit 列 ORDPATH よりもラベルサイズが小さいと言える．

6. 深さによる絞り込み

ラベルサイズによる絞り込み手法 [9] は，先祖子孫関係を示す包含関係に関する問い合わせを効率よく行うための手法であるが，いくつかの欠点がある．それらの欠点を解決する手法として，XML 木の各ノードの存在する深さに着目した絞り込み手法を提案する．

6.1 ラベルサイズによる絞り込みの欠点

ラベルサイズによる絞り込み手法 [9] は，特定のノード A の子孫ノードを求める問い合わせ処理における絞り込み手法である．絞り込みを行わない場合，全てのノードに対して，特定のノード A の子孫であるかをラベル同士のマッチングにより判別する．ラベルサイズを用いた絞り込みでは，まずノード A のラベルサイズ l を求め，l 以下のラベルサイズをもつノードを調査の対象から除外することで，マッチング判定の回数を減らし，問い合わせ速度を向上させる．

しかしこの手法には特定のノード A においては効率的な絞り込みが行われないという欠点がある．XML 木において同じ深さに存在するノードであっても，付されるラベルサイズは異なる．そして，特定のノード A のラベルサイズ l が短いとき，そのノードの子孫ノードになりえない，同一の深さに存在するノードも，問い合わせ時の候補として挙げられる．また，挿入を行うと，挿入されたノードのラベルは既にあるノードのラベルよりも長い場合が多いため，同様の理由から挿入されたノードが候補として残ることが多くなる．さらに，特定のノードが A_1，A_2 のように複数ある場合に，これらのノードのラベルサイズ l_{A_1} と l_{A_2} が等しくなければ，l_{A_1} と l_{A_2} のそれぞれを用いて絞り込みの処理が行われるため，絞り込みの処理が 2 倍に増える．l_{A_1}，l_{A_2} のうち，短い方のみを絞り込みに用いると絞り込み処理の回数は減るが，これによりもっとも短い l を選択する処理が発生する．

6.2 深さによる絞り込み

このような問題を解決するため，XML 木中のノードの存在する深さを用いた手法を提案する．深さによる絞り込みでは，ルートノードの深さを 0 とし，特定のノード A の深さを d とすると，深さ 0 から深さ d を含めた範囲に存在するノードを，確実に子孫ノードの候補から除外することができる．この手法を用いることで，包含関係を求める問い合わせにおいて，子孫ノードの候補をより少なく絞り込むことができ，マッチング判定の回数を減らすことができる．

表 7　実 験 環 境

CPU	AMD Opteron(tm) Processor 248 × 2
HDD	SEAGATE ST336607LC 37GB
Memory	6.1GB
OS	RedHat Linux 3.3.2-1
DBMS	PostgreSQL 7.3.4-RH
java	jdk1.5.0_05

DO-VLEI では，深さはデリミタの個数に相当する．また，ORDPATH では奇数の個数が深さに相当する．

深さによる絞込みのアルゴリズムは以下の通りである．まず，特定のノード A のラベルから深さ d を算出する．次に，ノード A の子かどうかを判別したい全ての候補ノードの位置する深さを調べ，d 以下のノードを，候補から除外する．最後に，残った候補ノードのラベルと，ノード A のラベルとでマッチング判定を行い，ノード A の子孫ノードを決定する．

7. 問い合わせ処理時間の比較実験

深さによる絞り込み手法は，まずラベルから深さを抽出し，次に深さを用いて絞り込む，という二段階構成になっている．ここでは，最初に深さによる絞り込みの効果について，次にラベルから深さを抽出する処理について，それぞれ検証する．

7.1 絞り込み処理時間の比較実験

二つのラベル付け手法によってラベル付けされたデータに対して，三つの絞り込み手法により問い合わせ処理行い，その絞り込みの効果を測定する．

7.1.1 実 験 準 備

表 7 に実験環境を示す．使用するデータ，RDB のスキーマ，挿入操作は 5 節と同様である．この実験では深さの絞り込みがどの程度の効果を得るかを測定するため，ラベルから深さを抽出する処理を省き，表 3 に示す様に，深さを保持する RDB スキーマの属性をラベル付けを行う際に各タプルに持たせる．ラベルから深さを抽出する処理については 7.1.2 節を参照されたい．

6 節で述べたアルゴリズムを表現した SQL を以下に示す．

（1） 条件を満たす特定のノード A の深さを求める．

```
CREATAE TABLE d AS
(SELECT depth FROM nodeTable WHERE 条件);
```

（2） d より深さが大きいノードを，子孫ノード候補とする．

```
(深さが小さいものを候補から除外する)
CREATE TABLE candidate AS
(SELECT * FROM nodeTable WHERE depth > d);
```

（3） マッチング判定を行う．

```
(ここでは部分文字列を使ったマッチング判定を用いる)
SELECT candidate.* FROM candidate,
        nodeTabel as parentNode
WHERE parentNode.label || '.'
        = substring(candidate.label, 1, parentNode.legnth + 1);
```

問い合わせ処理には XMark [13] で使用する 20 個のクエリセットの中から，上記の SQL に適合できるクエリを選び，そのクエリを SQL に変換したものを用いる．

表 8　初期状態における問い合わせ処理時間の平均値

単位 : ms		scaling factor			
Query1		0.001	0.01	0.1	0.3
non-Select	C-DO-VLEI	176	521	2056	5866
	C-ORDPATH	198	511	2076	6003
by Length	C-DO-VLEI	220	534	2074	5999
	C-ORDPATH	219	533	2087	6207
by Depth	C-DO-VLEI	232	517	1510	4066
	C-ORDPATH	232	538	1532	4103
Query5					
non-Select	C-DO-VLEI	153	2180	192298(1)	1700755(1)
	C-ORDPATH	158	2188	195300(1)	1724641(1)
by Length	C-DO-VLEI	354	799	7561	38804
	C-ORDPATH	384	660	5714	33713
by Depth	C-DO-VLEI	391	625	4209	25932
	C-ORDPATH	388	614	4757	30144
Query6					
non-Select	C-DO-VLEI	180	353	954	2243
	C-ORDPATH	185	339	975	2280
by Length	C-DO-VLEI	198	345	956	2286
	C-ORDPATH	201	355	959	2288
by Depth	C-DO-VLEI	205	345	950	2269
	C-ORDPATH	207	335	969	2320

表 9　3 回挿入を行った状態における問い合わせの処理時間の平均値

単位 : ms		scaling factor			
Query1		0.001	0.01	0.1	0.3
non-Select	C-DO-VLEI	464	1807	16866	51671
	C-ORDPATH	463	1827	16950	51806
by Length	C-DO-VLEI	450	1845	16981	52556
	C-ORDPATH	421	1885	17368	53062
by Depth	C-DO-VLEI	410	1245	10907	32027
	C-ORDPATH	405	1248	11021	32400
Query5					
non-Select	C-DO-VLEI	2041	147552(1)	14572182(1)	未測定
	C-ORDPATH	2036	148901(1)	14856354(1)	未測定
by Length	C-DO-VLEI	635	6134	222119(10)	1721354(1)
	C-ORDPATH	695	6501	251350(10)	1953786(1)
by Depth	C-DO-VLEI	445	1522	27765	190695(1)
	C-ORDPATH	477	1587	32061	225197(1)
Query6					
non-Select	C-DO-VLEI	212	806	6515	19094
	C-ORDPATH	208	856	6592	19207
by Length	C-DO-VLEI	200	856	6527	19200
	C-ORDPATH	213	874	6570	19506
by Depth	C-DO-VLEI	230	828	6140	18089
	C-ORDPATH	254	836	6137	18287

7.1.2　絞り込み処理時間の実験結果

初期状態における実験結果を表 8 に，三回挿入を行った状態における実験結果を表 9 に示す．基本的には 100 回の測定の平均値を取ったが，複数回の測定において値のばらつきは少なかったため，時間のかかるものは測定回数を減らした．C-DO-VLEI が圧縮 bit 列 DO-VLEI，C-ORDPATH が圧縮 bit 列 ORDPATH，non-Select が絞り込みを行わない状態での問い合わせ，by Length がラベルサイズを用いた絞り込み手法，by Depth が深さを用いた絞り込み手法である．なお，表 9 の Query5 において，scaling factor が 0.3 の non-Select の問い合わせ処理は時間がかかりすぎるため，今回は測定を行わなかった．

表 8，表 9 の Query1，Query5 では，ラベルサイズによる絞り込みよりも深さによる絞り込みの方が処理が速い．深さによる絞り込みの方が値のばらつきが少ないため効果的に絞り込みを行うことができるためである．表 8 の Query6 では，絞り込み手法による有意差は得られなかった．元々の問い合わせに存在する子孫ノードの条件が，絞り込みによる候補削減よりも強い絞り込みを行っているために，ラベルサイズや深さの絞り込みの効果がなかったと考える．しかし，挿入が行われた状態である表 9 の Query6 では，ラベルサイズによる絞り込みに比べ平均 7%の処理時間が削減されており，その効果が確認できる．

圧縮 bit 列 DO-VLEI と圧縮 bit 列 ORDPATH を比較すると，表 8 の Query5 では，ラベルサイズによる絞り込みで圧縮 bit 列 ORDPATH の方が平均 15% 速い．圧縮 bit 列 ORDPATH は接頭辞スキーマ (表 2) にあるように定まった bit 長で値を表現する部位があるためラベルサイズにばらつきが少ない．したがって，絞り込みに用いる長さの個数が減り，結果として絞り込み処理の回数が減る．しかし，表 9 では逆に圧縮 bit 列 DO-VLEI の方が速い．これは挿入操作によって，圧縮 bit 列 ORDPATH のラベルサイズにおけるばらつきが増加したことが一因であると考える．

また，Query1 では深さによる絞り込みにおいて，圧縮 bit 列 DO-VLEI と圧縮 bit 列 ORDPATH の差はなかった．ノードの深さがどちらの手法を用いても同じであるため，絞り込みの効果が同程度であることが理由である．ところが Query5 では深さによる絞り込みにおいて，scale factor が増すにつれて圧縮 bit 列 DO-VLEI の方が圧縮 bit 列 ORDPATH より速くなった．scale factor が 0.3 の時では圧縮 bit 列 ORDPATH の約 15% 速い．これは大きなラベルサイズを扱う方が substring 関数などを扱う上で効率が悪いことが一因と考える．

7.2　ラベル字句解析処理

問い合わせ処理では深さによる絞りこみ手法において二つのラベル付け手法に有意な差は得られなかった．そこで，ラベルから深さを抽出するための字句解析処理について考える．

7.2.1　字句解析の要求

RDB に XML 文書を格納する際に，できるだけ格納に必要な容量を小さくしたいという要求があるため，必要最小限の RDB スキーマが求められる．例えば，表 3 では要素ノードテーブルが (id,parent,label,length,depth,name) という 5 つの属性を持っているが，label は一意の値をもつため ID の代わりになり，また parent や length，depth は label から抽出できる内容である．ゆえに最小限の RDB スキーマを考えると (label, name) の二つの属性があればよい．この状態で問い合わせを行うためには label を字句解析して depth などの値を抽出する必要がある．したがって，容量を極力抑えるような状況においては，字句解析にかかる時間の差異が問い合わせ処理時間に影響すると考える．

7.2.2　圧縮 bit 列 DO-VLEI の深さ抽出

圧縮 bit 列 DO-VLEI ではデリミタの数がそのノードの存在する深さに相当する．そして,「.1」を表す (11) を，label から探し出すことでデリミタの数は求めることができる．

7.2.3　圧縮 bit 列 ORDPATH の深さ抽出

圧縮 bit 列 ORDPATH では奇数の数が深さに相当する．以下

表 10　RDB に収められたラベルから深さを求める処理時間

初期状態　　単位：ms

scaling factor	0.001	0.01	0.1	0.3
C DO-VLEI (A)	1.68	19.35	202.06	549.05
C ORDPATH (B)	2.78	26.99	299.22	939.96
A / B × 100%	60.4%	71.7%	67.5%	63.2%

三回の挿入後　　単位：ms

C DO-VLEI (A)	20.02	193.02	2143.47	7031.31
C ORDPATH (B)	30.82	305.85	3282.54	10537.91
A / B × 100%	65.9%	63.1%	65.3%	66.7%

の手順によって label から奇数の数を求める．まず label の先頭から逐次的に解析を行い L_i を算出する．続いて接頭辞スキーマ (表 2) を用いて O_i の bit 長を決定する．O_i から一組の L_i-O_i が表す 10 進数の値を求める．最後に負の整数も考慮に入れて，2 乗してから 2 で割り，余りが 1 であれば奇数として数える．

7.2.4　深さを求める処理時間実験

先の実験と同様の実験環境で行った．圧縮 bit 列 DO-VLEI のデリミタの数と，圧縮 bit 列 ORDPATH の奇数の数をそれぞれの label 属性から抽出する処理時間を測定した．結果を表 10 に示す．結果から，深さを求める処理にかかる時間は，平均 35%，圧縮 bit 列 DO-VLEI が速いことが分かった．これは，圧縮 bit 列 ORDPATH をスキーマに照会する処理と，そこから得た 10 進数の偶奇性判定が圧縮 bit 列 DO-VLEI には存在しないため，処理時間に差が出たと考える．

以上のことから，格納量を重視し，問い合わせ時に字句解析を行うことを前提とすれば，深さを得るためのラベル字句解析の処理コストの差により，問い合わせ処理において圧縮 bit 列 DO-VLEI は圧縮 bit 列 ORDPATH より優れている．

8. 結　　論

挿入無制限なラベル付け手法である DO-VLEI を前提として，ラベルサイズの縮小を図った圧縮 bit 列 DO-VLEI を提案した．同様に無制限な挿入を可能とする圧縮 bit 列 ORDPATH との比較を行い，初期のラベル付けにおいて圧縮 bit 列 ORDPATH の 80%以下のラベルサイズを維持することを示した．さらに，wiki のような Web 上の文書形式を更新することを想定し，XML 文書全体に偏りなく挿入を行った状態においても比較を行った．その結果，現実的なデータ規模において圧縮 bit 列 ORDPATH よりもラベルサイズが小さいということを示した．

また問い合わせ処理の向上を図るため，深さによる絞り込み手法を提案した．圧縮 bit 列 DO-VLEI と圧縮 bit 列 ORDPATH の二手法に対して深さによる絞り込みとラベルサイズによる絞り込みを適用し，処理時間を比較した．これにより，深さによる絞り込みがラベルサイズによる絞り込みよりも絞り込みの効果が高いことを示したが，また深さによる絞り込みにおいて圧縮 bit 列 DO-VLEI と圧縮 bit 列 ORDPATH にほとんど差が出なかった．そこで圧縮 bit 列 DO-VLEI と圧縮 bit 列 ORDPATH のラベルからノードの深さを抽出する処理時間を測定・比較したところ，圧縮 bit 列 DO-VLEI の方が圧縮 bit 列 ORDPATH の平均 35% 処理にかかる時間が少なかった．ゆえに，問い合わせ処理を行うときに深さを抽出するラベル字句解析を必要とする場合は，圧縮 bit 列 DO-VLEI の方が優れていることを示した．

今後の課題として，素数を利用したラベリング手法 [10] などの挿入無制限なラベリング手法が他にも提案されているため，これらの手法とも比較を行う必要がある．また，今回挿入パターンは偏りなく行ったが，より広い事例に対応するために，現実的な使用環境で想定される挿入パターンにおいても検討を行う必要がある．また，今回は DOM によって XML 文書を取り込んでいるため，scale factor 0.3 までの文書しかメモリ上に乗らなかった．そこで，DOM ではなく SAX による実装を行い，より大規模な文書に対して挙動を確認する必要がある．

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 (16016232)，独立行政法人科学技術振興機構 CREST，および東京工業大学 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」の助成により行われた．

文　　献

[1] Igor Tatarinov, Stratis Viglasand Kevin S. Beyer, Jayavel Shanmugasundaram, Eugene J. Shekita, and Chun Zhang. Storing and querying ordered xml using a relational database system. In *Proc. of SIGMOD Conf.*, pp. 204–215, 2002.

[2] 江田毅晴 , 天笠俊之 , 吉川正俊 , 植村俊亮. Xml 木のための更新に強い節点ラベル付け手法. In *DBSJ Letters,No.1 in Vol.1,2002*, 2002.

[3] Paul F. Dietz. Maintaining order in a linked list. In *Proceedings of the fourteenth annual ACM symposium on Theory of computing*, pp. 122–127, 1982.

[4] Online Computer Library Center. Introduction to the dewey decimal classification. http://www.oclc.org/oclc/fp/about/about_the_ddc.htm.

[5] Toshiyuki Amagasa, Masatoshi Yoshikawa, and Shunsuke Uemura. Qrs: A robust numbering scheme for xml documents. In *19th International Conference on Data Engineering (ICDE 2003)*, pp. 705–707, 2002.

[6] Edith Cohen, Haim Kaplan, and Tova Milo. Labeling dynamic xml trees. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART*, pp. 271–281, 2002.

[7] Kazuhito Kobayashi, Wenxin Liang, Dai Kobayashi, Akitsugu Watanabe, and Haruo Yokota. Vlei code: An efficient labeling method for handling xml documents in an rdb. In *ICDE*, 2005.

[8] 小林一仁 , 横田治夫. 挿入制限のないコード vlei を用いた xml ラベリング手法の検討とその評価. 第 15 回電子情報通信学会データ工学ワークショップ (DEWS2004) 論文集, 2004.

[9] 長良香子 , 小林一仁 , 小林大 , 横田治夫. Xml データベースのラベル付け手法 vlei の評価. 第 16 回電子情報通信学会データ工学ワークショップ (DEWS2005) 論文集, 2005.

[10] Wynne Hsu Xiaodong Wu and, Mong Li Lee and. A prime number labeling schema for dynamic ordered xml trees. In *ICDE*, pp. 66–78, 2004.

[11] Patrick O'Neil, Elizabeth O'Neil, Shankar Pal, Istvan Cseri, and Gideon Schaller. Ordpaths:insert-friendly xml node labels. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Paris, France, June 13-18, 2004*, pp. 903–908, 2004.

[12] 韓 太舜 , 小林欣吾 , 金谷文夫 , 山本博資 , 横尾英俊 , 鈴木譲 , 森田啓義 , 佐藤創 , 伊藤秀一. 情報源符号化 無歪みデータ圧縮. 情報理論とその応用シリーズ 1-1. 培風館, 1998.

[13] Albrecht Schmidt, Florian Waas, Martin Kersten, and Daniela Florescu. The xml benchmark project. Technical report, Technical Report INS-R0103, http://monetdb.cwi.nl/xml/index.html, April 2001.

[14] xmlgen. http://monetdb.cwi.nl/xml/downloads.html.

[15] B.Leuf and W.Cunningham. *The Wiki Way:Quick Collaboration on the Web*. Addison-Wesley, 2001.