Vol.2011-DBS-153 No.33
2011/11/3

経歴・オフセット法による XML 文書の一実装方式

天木圭太[†]　西野裕臣[†]　都司達夫[†]　樋口健[†]

我々は，経歴オフセット法と呼ぶ多次元データセットのエンコード方式に基づいて，XML 木の構造更新に対して再ラベル付けを行う必要がない XML 木の構造エンコーディングの方式を提案している．この方式では XML 木の構造に従ったエンコードが行われるが，ノード名の連接であるノード名パス式が扱われていない．したがって，XPATH などによる検索要求を受け付けることはできない．ここでは，XML 文書を入力として，XML 木の構造をエンコード/デコードするデータ構造とノード名パス式の構造をエンコード/デコードするデータ構造を構築する．２つのデータ構造によるエンコード結果を相互に参照することにより，パス式の検索と軸の指定による構造検索を効率よく行う方式を提案し，システム構築と性能評価を行う．

An Implementation Scheme of XML Documents Based on History-offset Encoding

Keita Amaki[†]　Hiroomi Nishino[†]　Tatsuo Tsuji[†]　and Ken Higuchi[†]

In our previous work, a new labeling scheme for dynamic XML trees has been proposed, in which no relabeling is necessary against the structural update of the trees. The labeling scheme is based on the encoding method for multidimensional datasets called history-offset encoding. In the scheme, each node in the XML tree can be encoded along the tree structure, but a path expression, being the concatenation of the node names on the path, has not been treated. In this research, two kinds of data structure are constructed based on the history-offset encoding. One is for encoding/decoding the structure of an XML tree and the other is for encoding/decoding path expressions from the root node. By cross-referencing to the encoded results stored in these two kinds of data structures, both of the structural retrieval using axes and the retrieval of path expressions can be performed very efficiently. Using the constructed system, the performance of our proposed scheme is evaluated.

1. はじめに

我々は，経歴・オフセット法と呼ぶ多次元データセットのエンコード方式に基づいて，XML[1]を木構造として表現した XML 木の動的な構造更新に対して再ラベル付けを行う必要のない XML 木の構造エンコーディング方式を提案している[2]．この方式は，XML 木の構造情報を多次元拡張可能配列[3][4]に埋め込むことにより XML 木のノードのエンコードを行うが，他方式[5]~[8]と比べ，ノードの追加場所や追加順序に関わらず，ラベルの記憶コストが一定であり，ラベル検索も高速に行うことができる．

一般にラベル付けの方式では，XML 木の構造情報に対するエンコードが行われるのみであり，XML 文書中のタグ名（要素名）や属性，およびテキストデータ等の内容情報は捨象される．したがって，XML 木もノード名の連接であるノード名パス式は扱われないので，XPath などによるユーザからの検索要求を受け付けることはできない．

ここでは，単にラベル付けによる構造エンコーディングだけではなく，ノード名パス式のエンコーディング法も併せて提案し，高速に Xpath の検索要求を処理できるシステム構成を提案する．本システムでは，XML 文書を入力として，文書の内容情報を取り扱うと共に，XML 木の構造を経歴・オフセット法によりエンコード/デコードするデータ構造と，ノード名パス式の構造をエンコード/デコードするデータ構造を構築する．二つのデータ構造によるエンコード結果を相互に参照することにより，軸の指定による構造検索とノード名パス式の検索を共に効率良く行う方式を提案し，構築したシステムにより，その性能評価を行う．比較評価にネイティブ XML-DB システムである eXist-db(Release-1.4.1)[9]を使用した，評価の結果，記憶性能では eXist-db にやや劣るものの検索性能では，どの検索条件においてもかなり高い性能を確認した．

2. 経歴・オフセット法と HOMD

経歴・オフセット法は，多次元拡張可能配列の概念[3]を基にした多次元データセットのエンコード方式である．通常の拡張可能配列は主記憶上に領域が確保されデータの格納を行う．しかし，多次元データセットを多次元配列上にマッピングする場合は，通常，使用される配列要素が少ない疎配列であり，記憶領域の無駄が多い．経歴・オフセット法では，配列要素のデータ実体は持たず（論理配列，以下，単に論理配列），多次元データタプルは拡張可能配列の対応要素の位置として表現する．拡張により動的に生成される部分配列にその拡張が何番目であるかを表す拡張経歴値を与え，論理配列の構造を表す次元毎に持つ経歴値テーブルと呼ぶ補助テーブルに格納する．論理配列の各配列要素は経歴値と，部分配列内の位置を表すオフセットによって一意に表

[†] 福井大学大学院工学研究科
Graduate school of Engineering, University of Fukui

ⓒ2011 Information Processing Society of Japan

すことができる．この二つの値の対＜経歴値，オフセット＞が配列要素のエンコードである．多次元データのタプルが高次元であっても，この二つのスカラー値の対で配列要素を表現することができる．部分配列の拡張時の各次元のサイズによってオフセットの計算に使用するアドレス関数の係数が異なるため，各係数を係数ベクトルとして記憶する．

ここで，経歴・オフセット法による論理配列の座標を経歴値とオフセットに変換するエンコードの方法と，経歴値とオフセットから論理配列の座標に変換するデコードの方法を例を示して説明する．図 1 の論理配列において座標(1, 2)を変換する場合を考える．一次元の添字 1 の経歴値 1，二次元の添字 2 の経歴値 4 が経歴値を保持するテーブルから取得できる．座標で示された配列要素が属する部分配列は，座標の各値（添字）と対応する拡張経歴値中で最大の経歴値によって識別される部分配列である．つまり，例では添字 2 と対応する経歴値 4 が最大であるため，座標(1, 2)の要素は部分配列 *S* に属する．さらに最大経歴値以外の経歴値に対応する添字と，*S* のサイズを持つ係数ベクトルから *S* のアドレス関数を計算し *S* における要素のオフセットを求めることができる．例ではオフセットは 1 であるので，経歴値とオフセットの対<4, 1>にエンコードできる．

次にデコード方法について，図 1 の経歴値，オフセット対<4, 1>を座標に変換する場合を考える．まず経歴値を保持するテーブルを参照することで，経歴値 4 から二次元の添字 2 が求まる．そして，経歴値 4 の部分配列の係数ベクトルから部分配列のサイズが取得でき，オフセットから添字 1 が求まり経歴値とオフセットを論理配列の座標(1, 2)に変換することができる．

n 次元拡張可能配列では配列に新たな次元を動的に追加して *n*+1 次元へ次元拡張することができる．次元拡張時には，既存の経歴値テーブルと係数ベクトルは更新する必要がなく，次元拡張前の配列座標の再エンコードも必要ない．追加する新たな次元方向に対して，経歴値テーブルと係数ベクトルテーブルを設け，拡張前の *n* 次元配列の追加次元での添字は 0 となる．以後の次元サイズの拡張は *n*+1 次元の拡張可能配列として同じ手順で拡張することができる．

上述の経歴・オフセット法に基づく多次元データセットの実装データ構造を，HOMD（History-Offset implementation scheme for Relational Table）[4]と呼ぶ（図 2）．HOMD では，多次元データの各属性，属性値，タプルをそれぞれ論理配列の次元，添字，配列要素の座標に対応させる．HOMD は配列の各次元について，上述の経歴値テーブル，係数ベクトルテーブルおよび CVT（key-subscript ConVersion B+Tree）を持ち，一つの RDT（Real Data Tree）を持つ．各次元の CVT は B+木であり，当該次元の属性値をキーとし，その属性値に対応する添字をデータとして持つ．また，RDT は＜経歴値，オフセット＞の対を格納する B+木であり，経歴値が上位ビットにオフセットが下位ビットに配置されることにより，そのシーケンスセットには＜経歴値，オフセット＞の対が経歴値の順にソートされている．

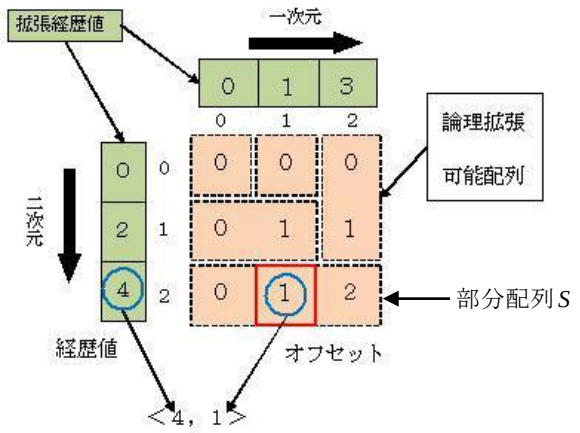

図 1　経歴・オフセット法の概念図

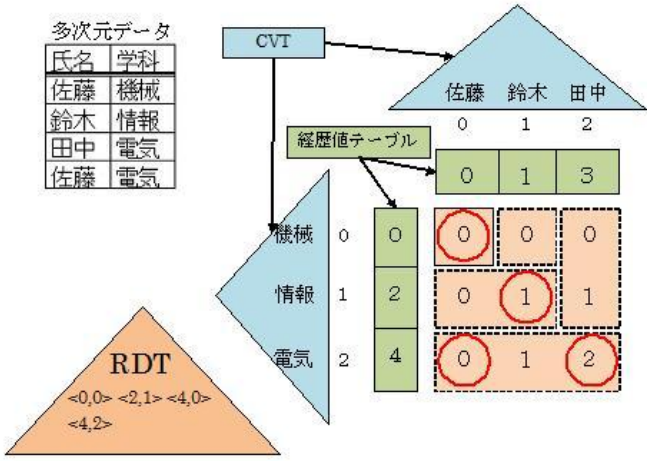

図 2　HOMD データ構造

ⓒ2011 Information Processing Society of Japan

Vol.2011-DBS-153 No.33
2011/11/3

3. XML 木ノードへのラベル付け方式

本研究で用いる XML 木ノードのラベル付けには前節で述べた経歴・オフセット法を用いる. XML 木のノードが存在する木の各高さレベルを論理配列の各次元に対応させる．各高さレベルに存在する兄弟ノードに対して左から順にその木の高さに対応する次元の添字を１，２，３，…と割り当てる．この時，各ノードはその次元における添字とその各祖先ノードの次元の添字とで論理配列上の座標と見ることができる(図 3)．この座標値は祖先の座標を含んでいるため，座標値を比較することでその座標値を持つノードと比較対象の座標値を持つノードとが祖先，子孫，兄弟の関係であるかの判別ができる．また，異なるノードの共通の祖先もそれぞれのノードの座標値で共通して持つ添字を調べることで簡単に求めることができる．

ノードの座標値は論理配列の一つの要素に対応する．座標値は前節で述べたように経歴・オフセット法により，経歴値とオフセットの対として表現できる．この対をラベルとして，図 2 の HOMD データ構造の RDT に格納する．

XML 文書の属性や PCDATA も要素ノードと同様に論理配列の一つの要素として扱う．属性はその属性を持つ要素ノードの子ノードとして扱う．格納したタグ，属性，PCDATA の各種類の識別のために，RDT に＜経歴値，オフセット＞のラベルを格納する際に，ラベルの上位 2 ビットを用いる．属性値，PCDATA のテキストの内容はそれぞれ，格納用にファイルを用意し，RDT 上で属性や PCDATA のラベル値に対応するデータ部に格納用ファイル内の当該データが格納されている位置へのオフセットを格納する．このように，ラベル値を知ることで，XML 文書でのタグ，属性，PCDATA の判別や，実データの参照が可能となる．

ノードの挿入によって XML 木の各木の高さでのノードの最大分岐数が変化した場合に，論理配列ではその木の高さに対応する次元方向への配列拡張が起こる．このように，分岐数の増大に対しては論理的な配列の拡張を行うことによって動的な更新に対応する．また，木の高さが増加して次元数が大きくなった場合には前節で述べた次元拡張の機能によって，再ラベル付けの必要なしに対応することができる．この場合でも，ラベル値は＜経歴値，オフセット＞の二つのスカラー値で表現でき，固定長で扱うことができる．これらの利点は各次元サイズが固定の通常の多次元配列では獲得できない．

この, 経歴･オフセット法を用いたラベル付けによる XML 木の格納構造を nHOMD, その RDT を nRDT，経歴値とオフセットの対のラベル値を nID と呼ぶ（n は node の n を表す）.

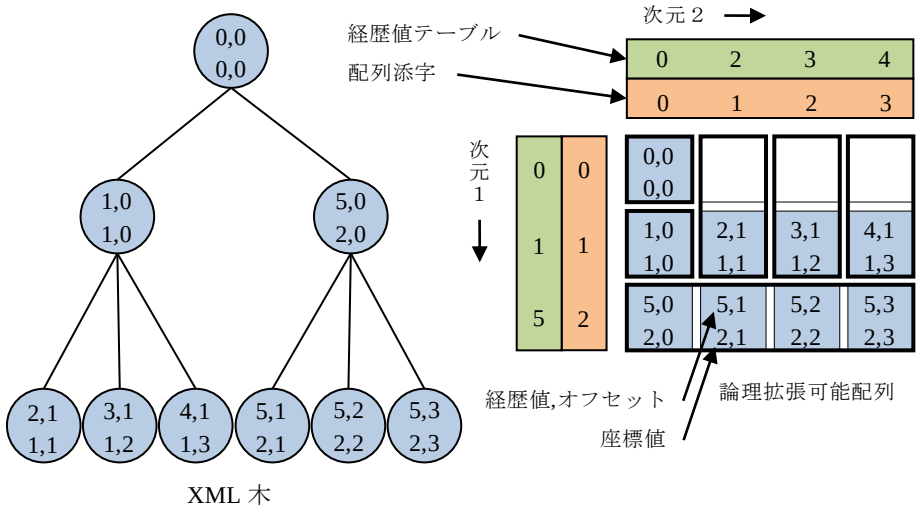

図 3　経歴・オフセット法によるラベル付け

4. 文書順の保持方法

XML 文書の重要な情報として，”文書順（document order）” が存在する．前節で述べたラベル付けの方法は文書順ではなく, 出現順にノードを論理配列に拡張するため, XML 木のノードを pre-order で格納した場合はノードに対応する配列添字がそのまま文書順を表現するが，新たなノードが追加されると添字が文書順を表さなくなる場合がある．そのため，ラベルだけで文書順を保った XML 文書を復元することができない．

そこで，文書順の情報を OS（Ordered Sequence）テーブル（図 4）と呼ぶ補助データ構造によって保持する．OS テーブルは一次元の配列で表現される．OS テーブルの添字は論理配列の添字を表す．要素には文書順でその要素の添字の次の兄弟ノードの添字を格納する．この OS テーブルを順に辿ることで，文書順を再現することができる．このとき，OS テーブルが表す兄弟ノードの先頭のノードも OS テーブルと共に記憶する．こうすることで，先頭の兄弟ノードから OS テーブルを辿る操作を行うことができる．これら OS テーブルの情報をその OS テーブルが文書順を保持する兄弟の親ノードの nID から参照できるように nRDT のデータ部に格納する．

ⓒ2011 Information Processing Society of Japan

新たなノードが挿入された場合には，その兄弟ノードの OS テーブルの内容を更新することで文書順を保つことができる．この更新は高々二つの要素の書き換えのみで行うことができる．

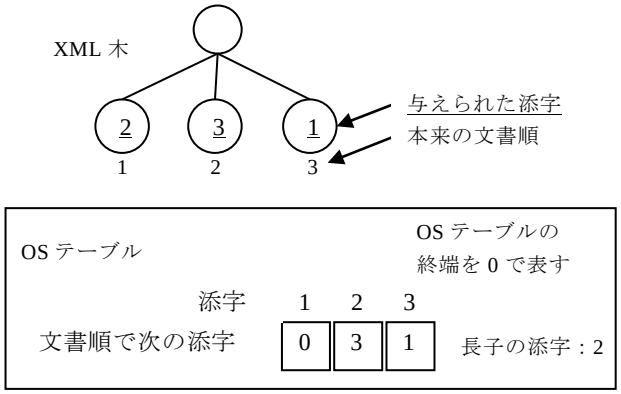

図 4　OS テーブルの構造

5. パス式の格納方法

前節で XML 木の木構造としての格納方式について述べた．しかし，ノード名（タグ名）やノード名のパス式の格納については扱っていない．本節では，XML 木のノード集合の構造を格納管理する nHOMD とは別に，タグ名のパス式を格納管理するために 3 節で述べた HOMD の構造を用いる．HOMD を用いることでルートノードからのパス式は対応する＜経歴値，オフセット＞としてエンコードできる．重複したパス式は同じ＜経歴値，オフセット＞にエンコードされる．

nHOMD の場合と同様に，XML 木の各レベルの高さを HOMD の論理配列の各次元に割り当てる．それぞれの次元の添字はその次元と対応する木の高さに属するノード名に対応させる．同じ名前のノードが同じ次元に複数存在する場合には，それらは同じ一つの添字に対応させる．属性についても同様に属性名をそれぞれの添字に対応させる. PCDATA については PCDATA であることを示す名前を与え, 添字に割り当てる．

ノードの名前から論理配列の添字への変換は CVT を用いる. こうすることで, XML 木の一つのノード名のパス式が HOMD の論理配列内の一要素に対応する座標値に変換される．論理配列の一要素は前述の通り，＜経歴値，オフセット＞にエンコードすることができ，これはキー値として HOMD の RDT に格納される．

複数存在する同一ノード名が一つの添字で表現されるため，同じパス式のノードはすべて，一つの同じキー値である＜経歴値，オフセット＞によって表される．図 に表すように，XML 木のパス式の情報はそれ以外の情報を排した木として表現でき，パス式情報をコンパクトに表現することができる．

この HOMD を用いてパス式情報を格納した構造を pHOMD，その RDT を pRDT，経歴値とオフセットの対を pID と呼ぶ（p は path の p）．

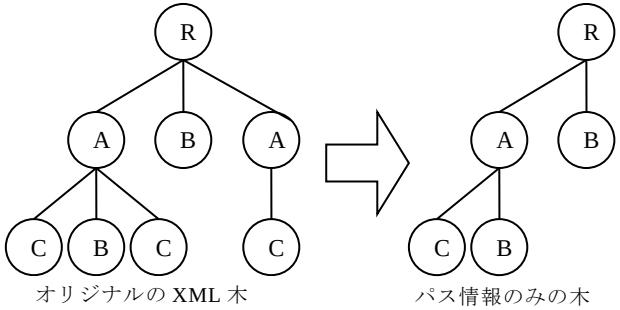

図 5　HOMD 上で表現されるパス情報のみの木

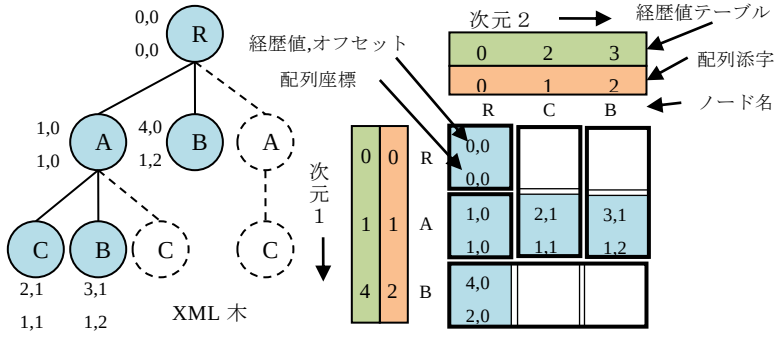

図 6　パス式を格納する HOMD（pHOMD）

6. nID とパス式の相互変換方法

nHOMD における nID を対応するノード名のパス式に変換するには，nID と pID を対応させる必要がある．そこで， nID に対応するノードのパス式を表す pID(唯一)を

ⓒ2011 Information Processing Society of Japan

その nID をキーとするデータとして nRDT に格納する．pID は pHOMD の係数ベクトルを用いて各次元の添字にデコードされる．pHOMD の論理配列の添字はノード名に対応しているので，ルートノードから対象のノードまでのパス式のノード名集合が求まる．こうして，nID からパス式への変換を行うことができる．すなわち，nID で指定される XML 木のノードを XML 文書のパス式にデコードすることができる．

逆に，パス式を nID に変換するには pID から nID を求める必要がある．これには pHOMD において pID と nID を関連付ける必要があるために，pRDT において，キーである pID のデータ部にその pID に対応する nID を格納する．pID は一つのノードパス式を表すが，pID 一つに複数のノードが対応し得るので，一般に nID が複数対応することになる．そこで，それぞれの pID に対して二次記憶中に一定サイズのディスクブロックをリスト要素としたリスト構造に nID を順次格納する．pRDT にはこのリスト構造の先頭ブロックのアドレスを格納する．こうして，ディスクのシーク回数を抑制して，対応する nID 集合を迅速に得る．

例として，図 の XML 木で/R/A/C というパス式が指定された場合を考える．まずそれぞれの高さにノード名を分けると，ルートは R，高さ 1 のノード名は A，高さ 2 のノード名は C である．この A と C をそれぞれの次元の CVT で添字に変換すると A が添字 1，C が添字 1 に変換される．２節で述べたように，この添字から，経歴値とオフセット値に変換することができる．結果，経歴値とオフセット値が＜２，１＞と求まるので，これを pRDT から探し，リストから nID を取得する．

6.1 具体例

図 5 の XML 文書を具体例として説明する．

6.1.1 nID からパス式への変換

図 7 の nHOMD において nID <5,1> は ”/本社/事業所/業務部”のパス式に一意的に対応している．nRDT から nID <5,1>を探す，そのデータ部には pHOMD を構築する時に格納した対応パス式の pID が格納されている．図 7 の場合は pID は<4,1>である．この pID は pHOMD 上で座標に変換できる．その座標値を対応するノード名に変換して対応パス式が得られる．このようにして，nID をパス式に変換することができる．

6.1.2 パス式から nID への変換

パス式から nID に変換する場合について説明する．まず，指定されたパス式（/本社/事業所/業務部）を木の高さ別のノード名に分解する．そして，分解したノード名を CVT により，添字に変換する．本社，事業所，業務部の添字がそれぞれ 0，1，3 と求まるので座標値は（1,3）となる．この座標値を経歴値とオフセットに変換すると，経歴値が 4，オフセットが 1 となるので，経歴値とオフセットを組み合わせて pID として pRDT から pID <4,1>を探す．pRDT のデータ部からそのパス式を持つノードの nID を格納しているリストを参照し，指定されたパス式に対応する二つの nID <4,1>と <5,1> を得る．このようにしてパス式を nID に変換することができる．

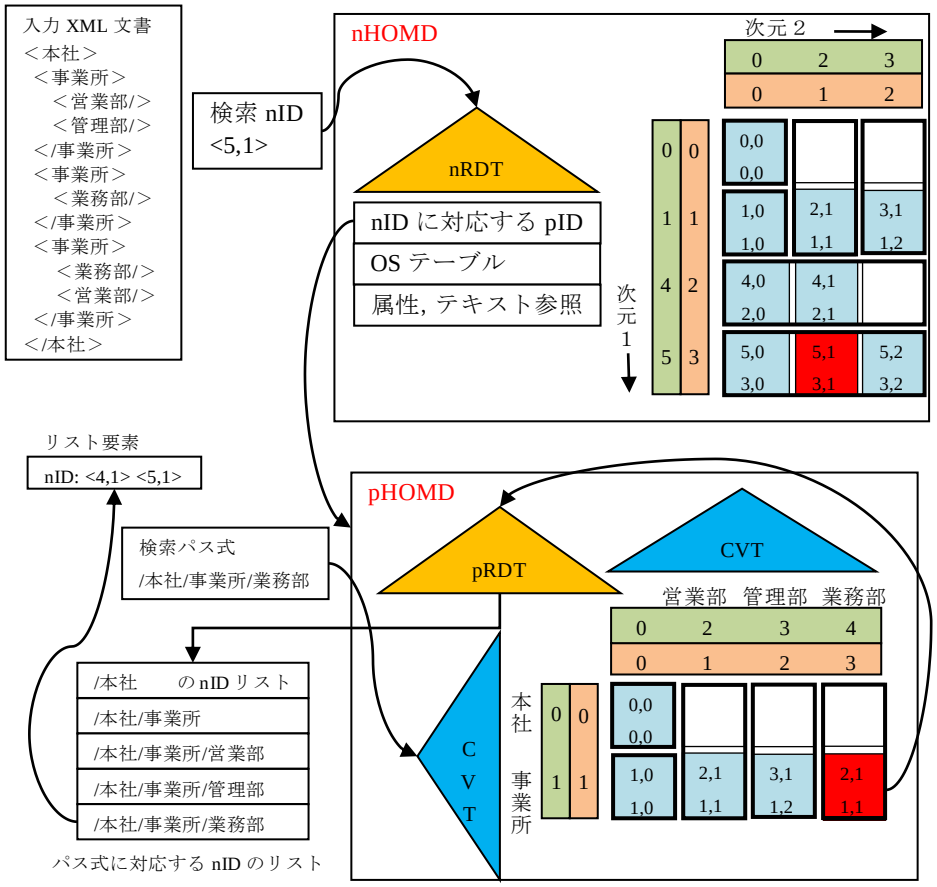

図 5 nID と pID の相互参照

7. nHOMD と pHOMD を用いた構造検索

前節で述べたように，pHOMD によって/R/A/B のようなパス式から該当するノードの nID 集合を求めることができる．しかし，カレントノードを起点とする軸に沿った構造検索には pHOMD だけでは対応できない．そこで，nHOMD を用いることにより

ⓒ2011 Information Processing Society of Japan

軸検索を行う．ここでは，XML 木のカレントノードの nID が与えられたとして，その子ノード，親ノード，兄弟ノードの nID を求める方法を説明する．

7.1 子ノードの検索

子ノードの検索は，まず，カレントノードの nID を nHOMD の論理配列の座標値に変換する. 次に, カレントノードの nID によって nRDT から OS テーブルを取得する．そして，OS テーブルを用いて長子から添字を順に辿り，カレントノードの座標と OS テーブルから得られた添字を組み合わせることで子ノードの座標値が長子から末子まで順番に求めることができる．子ノードの座標値を再び経歴値とオフセットにエンコードすることで子ノードの nID が得られる．得られた nID をキーとして，nRDT を探索し，対応する pID を求めることができる．さらに，pHOMD を使用して，6 節で述べた手順によって，pID に対応するノード名のパス式にデコードすることができる．

7.2 親ノードの検索

親ノードの検索は，まず，カレントノードの nID を同様に nHOMD の論理配列座標値に変換する．座標値の中で最大次元の添字を 0 にすれば親ノードの座標値となる．求まった親の座標値を再びエンコードして親ノードの nID を求めることができる．得られた nID からノード名のパス式へのデコードは同様に 6 節の手順による．

7.3 兄弟ノードの検索

兄弟ノードの検索は，まず，カレントノードの nID を nHOMD の論理配列座標値に変換する. 親ノードを求める方法でカレントノードの親ノードの nID を求め, その nID をキーとして nRDT から OS テーブルを取得する．その後，子ノードを求める方法と同様にカレントノードの親ノードの子ノードの nID を長子から末子まで求める．そのとき OS テーブルから得られる添字とカレントノードの座標値の最大次元の添字を比較して，一致するまでがカレントノードより文書順で先に存在する兄弟ノード，一致してから末子までがカレントノードより文書順で後に存在する兄弟ノードとなる．このように，nHOMD を用いて，XML 文書木の構造検索を処理することができる．得られた nID からノード名のパス式へのデコードは同様に 6 節の手順による．

7.4 具体例

図 6 の XML 文書を具体例として前述のノードの求め方を説明する．カレントノードを nID <2,0>，座標値(2,0)のノードとする

7.4.1 子ノードの検索例

子ノードを求めるには，カレントノードの nID <2,0> を nRDT から探し，その OS テーブルを取得する．そして，OS テーブルから文書順に添字を得る．図 8 の例では，OS テーブルから添字 1 と 2 が順に得られる．それぞれをカレントノードの座標値と組み合わせると，座標値(2,1), (2,2)が得られる．この得られた座標値を nID に変換すると<3,2>, <4,2>が得られる．この二つの nID がカレントノードの子ノードとなる．

7.4.2 親ノードの検索例

親ノードは，座標値の 0 でない最大次元の添字を 0 とおくことで求められる．カレントノードの座標値は(2,0)であるので，0 でない最大次元の添字は 2 である．この 2 を 0 とすると，座標値(0,0)が求まる．座標値(0,0)を nID に変換して<0,0>が得られ，親ノードの nID を求めることができる．

7.4.3 兄弟ノードの検索例

兄弟ノードを求めるには, まずカレントノードの親ノードを求める. 親ノードの nID は<0,0>である．nRDT から求めた親ノード<0,0>を探索し，OS テーブルを取得する．OS テーブルから文書順に添字を読み込む．例の場合は添字 1, 2, 3 が順に得られる．カレントノードの親ノードの座標値(0,0)と OS テーブルから読み出した添字を順に組み合わせ，nID に変換する．すると，最初に座標値(1,0)，nID<1,0>が求まる．カレントノードが現れるまではカレントノードよりも文書順で前に現れる兄弟ノードである．次に座標値(2,0)，nID<2,0>が求まり，これはカレントノードであるので，以降の兄弟ノードがカレントノードよりも文書順で後に現れるノードである. 最後に座標値(3,0), nID<5,0>が求まる．このようにして兄弟ノードを求めることができる．

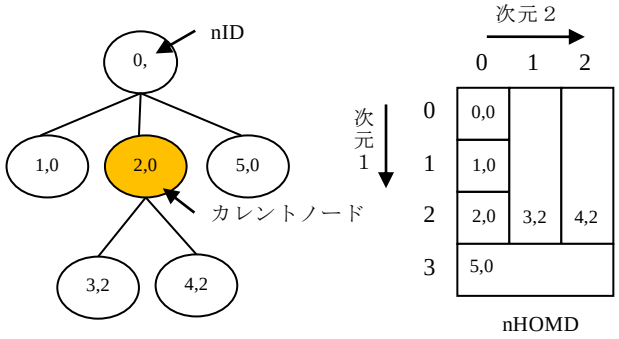

図 6　ノードの軸検索方法

8. 実験結果および考察

本研究における提案方式に従って構築したシステムに格納した XML 文書についてその記憶サイズと検索時間を計測する実験を行った. 検索は XPath[10]で表現できるいくつかの検索パス式を用いた．

実験に用いた計算機は以下の仕様である．

ⓒ2011 Information Processing Society of Japan

- CPU : Intel(R) Xeon(R) CPU X7560 @ 2.27GHz
- Memory : 65,992,668 [kByte]
- OS : Linux 2.6.18-164.9.1.el5

実験に用いた XML 文書ファイルは XMark[11]を用いて生成した以下のものである．

- ノード総数　370,311
  - 要素ノード　200,336
  - 属性ノード　53,512
  - テキストノード　116,463

実験には，提案方式のシステムとの比較対象としてネイティブ XML-DB システム eXist(version eXist-1.4.1-rev15155)[9]を用いた．

8.1 記憶コストの比較

提案方式の記憶コストは，メモリに格納される nHOMD と pHOMD，OS テーブルや文書内容を保持する外部データのファイルサイズを調べた．nRDT は B+木そのもののサイズではなく，ラベル値とデータの（pID，OS テーブル，属性値やテキストへの参照）対を格納したファイルのサイズとした．現在，データベースオープン時に nRDT の B+木を構築している．eXist の記憶コストについては，システムに XML ファイルを格納したのちデータベースの情報を格納するディレクトリ内のファイルサイズを調べた．これらの実験データを以下の表 1 に示す．

表 1　記憶コストの結果

提案方式		eXist	
データ名	記憶サイズ[MB]	ファイル名	記憶サイズ[MB]
nHOMD	10.1401	dom.dbx	16.4727
-nHOMD の内 nRDT	9.8884	elements.dbx	2.1406
pHOMD	0.02915	collections.dbx	0.01172
-pHOMD の内 pRDT	0.01542	values.dbx	0.007813
nID 格納用リスト	5.0938	ngram.dbx	0.007813
OS テーブル	0.03020	symbols.dbx	0.0008850
テキストデータ	5.8552		
属性データ	0.6368		
合計	21.2276	合計	18.6415

記憶コストは eXist が提案方式の 88%程度となっている．提案方式の記憶サイズは XML 木の構造を表現する nHOMD が大部分を占めており，nHOMD 内では特にラベルを記憶する nRDT のサイズが支配的である．

pHOMD は nHOMD に比べ非常に小さく主記憶上に配置できる．pHOMD はパス式に合致するノードを求める際に頻繁にアクセスされるため，pHOMD を主記憶に格納することで，検索時間を削減することができる．

8.2 検索コストの比較

それぞれのシステムにおいて，上記の XML 文書ファイルを格納し XPath を用いた問い合わせを行い，検索結果の件数が得られるまでの時間を測定した．各々の問い合わせを 10 回行い，その平均を求めた．実験結果を表 2 に示す．

検索コストは，どの問い合わせにおいても提案手法は eXist よりも高速に応答している．検索が nHOMD と pHOMD の構造内のみで完結するようなタグ名を単純に連ねたパス式の検索は高速に処理されている．逆に，属性値やテキストの内容を指定した検索は，ヒット数が多い場合には検索処理が比較的遅くなっている．これは，外部のファイルに格納された属性値やテキストのデータにアクセスする際，ファイル内の当該データの位置をシークするためであると考えられる．

表 2　検索コストの結果

検索パス式	提案方式 検索時間 [msec]	eXist 検索時間 [msec]	ヒット数
/site/regions/africa/item/description/parlist/listitem	4.8993	177.3	65
/site/closed_auctions/closed_auction/price	81.5875	265.8	1365
/site/people/person/name	121.3817	394.0	3570
/site/regions/*/item/description	213.6676	403.5	3045
/site/regions/africa/item/location[text()='United States']	5.4994	183.8	62
/site/regions/africa/item[@featured='yes']	4.2992	152.9	7
/site/regions/*/item	21.0967	367.0	3045
/site/regions/*/item/location[text()='United States']	262.960	614.7	2284
/site/regions/*/item[@featured='yes']	182.0725	254.4	303

pHOMD を用いて nID のリストからパス式に適合するノードを取得した時の検索時間を表 3 に示す．なお，この実験では，検索パス式の検索を 1000 回行った時間の合計を測定した．この検索は nHOMD の構造データを用いておらず，結果が文書順にソートされていない．また，属性値やテキストを指定した検索も nHOMD のデータを参照する必要があるため行っていない．

ⓒ2011 Information Processing Society of Japan

表 3　pHOMD を用いた場合の検索コストの結果

検索パス式	1000 回の検索時間[msec]
/site/regions/item/description/parlist/listitem	3.7993
/site/closed_auctions/closed_auction/price	22.0967
/site/people/person/name	52.5920
/site/regions/*/item/description	64.1902
/site/regions/*/item	63.5904

表 3 の結果は，nHOMD を用いた検索に比べ，殆ど無視できる時間コストである．nRDT などのディスク上のデータに何度もアクセスする必要が無く，nID 集合の取得もリスト要素単位でディスク上の連続領域に格納されているため，ディスクアクセスの回数が少なくすむ事が理由として挙げられる．

このように pHOMD では高速に検索を用いることで，nHOMD 上での検索範囲を狭めることで，より効率的に検索を行う事ができると考えられる．

9. おわりに

多次元データのエンコード方式である経歴・オフセット法により，XML 木の構造をエンコード/デコードするデータ構造とノード名のパス式の構造をエンコード/デコードするデータ構造を示した．さらに，nHOMD と pHOMD の 2 つの構造によって，パス式の検索と軸の指定による構造検索を効率良く行う方法を提案した．

さらに，他方式との性能比較を行い，記憶コストの面ではやや不利であるものの，検索コストに関してはややコストの高い，属性値やテキストを指定する検索を含め，全ての検索例で有利であることを示した．また，pHOMD を用いた検索が非常に高速であることについても示した．

今後の課題として，pHOMD を用いることで構造検索の高速化を図る手法の考案がある．また，経歴・オフセット法を用いたラベル値は固定長であり，予め想定したサイズ以上の XML が入力されると，ラベル値が溢れてしまう可能性がある．その問題に対応するための方法を実装することが必要である．

参考文献

1) Extensible Markup Language: http://www.w3.org/XML/
2) Li B.，Kawaguchi K.，Tsuji T.，Higuchi K.: A Labeling Scheme for Dynamic XML Trees Based on History-offset Encoding，情報処理学会論文誌: データベース, Vol.3, No.1, pp.1-17, 2010.
3) E.J.Otoo, T.H.Merrett: A Storage Scheme for Extendible Arrays, Computing, Vol.31, pp.1-9, 1983.
4) Tsuji T., Kuroda M., Tsuji T., Higuchi K.: History offset implementation scheme for large scale multidimensional data sets, Proc. of ACM Symposium on Applied Computing (SAC2008), pp.1021-1028, 2008.
5) O'Neil, P.E., O'Neil, E.J., Pal, S., Cseri, I., Schaller, G., Westbury, N.: ORDPATHs: Insert-friendly XML node labels. In: Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'04), pp. 903–908(2004)
6) Li, C., Ling, T.W.: QED: a novel quaternary en-coding to completely avoid re-labeling in XML updates. In: Proceedings of the 14th International. Conference on Informatio and KnowledgeManagement (CIKM'05), pp. 501–508 (2005)
7) B o h m e, T., E. R a h m. Supporting Efficient Streaming and Insertion of XML Data in RDBMS. – In: Proc. 3rd Int. Workshop Data Integration over the Web (DIWeb), 2004.
8) X. Wu, M. L. Lee, and W. Hsu, "A Prime Number Labeling Scheme for Dynamic Ordered XML Trees", Proc. of ICDE, pp.91-99, 2004.
9) XML Path Language (XPath), http://www.w3.org/TR/xpath/
10) XMark An XML Benchmark Project, http://www.xml-benchmark.org/
11) eXist-db Open Source Native XML Database, http://exist.sourceforge.net/

ⓒ2011 Information Processing Society of Japan