DEWS2006 6B-i4

索引分散管理システムを用いたXML文書検索

高橋　貴之† 　樋口　健† 　都司　達夫†

† 福井大学工学研究科　〒910–8507 福井県福井市文京 3–9–1
E-mail: †{takahasi,higuchi,tsuji}@pear.fuis.fukui-u.ac.jp

あらまし　本研究は索引分散管理システムを XML 文書検索に応用することを目的とする．本 XML 文書検索システムでは XML 文書中の要素の親子関係を親方向と子方向の 2 種類の索引として作成し，この索引を分割し，その索引を非共有メモリ型並列計算機上で分散管理することで検索処理の効率化を目指す．本システムでは XML 文書の親子関係を索引化しているため，XML 文書が変更された場合の索引への影響を小さくすることができ，XML 文書の要素にグローバルな番号付けを用いる方式の欠点である XML 文書の結合や分割時の再編成処理を行う必要がない．
キーワード　XML 文書検索，索引分散管理システム，並列処理

XML Document Search Method using Distributed Index System

Takayuki TAKAHASHI†, Ken HIGUCHI†, and Tatsuo TSUJI†

† Graduate School of Engineering, University of Fukui　Bunkyo 3–9–1, Fukui-city, Fukui, 910–8507 Japan
E-mail: †{takahasi,higuchi,tsuji}@pear.fuis.fukui-u.ac.jp

Abstract　This research proposes a new XML document search method using distributed index system. In our XML document search method, two kinds of indexes（parent-child index and child-parent index）are made to hold the information of parent-child relationships in an XML document tree. By dividing these indexes and managing on shared-nothing parallel computer, good performance for retrieve and modification will be expected. Since our system only uses the index for parent-child relationships in an XML document tree, the influence of changing XML document can be reduced, and large restructure which is needed in the systems using document numbering methods is not need.
Key words　XML document serch, distributed index system, parallel processing

1. はじめに

近年，インターネットやイントラネットなど，ネットワークにコンピュータが接続されることが普通になってきた．それにともない，様々なデータ形式による情報の転送や受け渡しを行なう場面が多くなってきているが，使用する計算機によっては処理できないデータ形式も多い．そこで，情報をタグという概念を用いることでテキスト形式で表すことができる XML [1] に注目が集まってきている．XPath [2] などの経路情報を検索条件に含むような XML 文書に対する問い合わせにおいては，索引を使用することで高速に検索する手法が取られることが多い [3]〜[9].

しかし，索引として経路情報まで含む場合は，XML 文書が更新された場合，索引を大規模に更新する必要がある．これは，オブジェクト指向データベースにおけるパスインデックス，入れ子インデックスが検索には高速であるが，更新には処理に時間がかかるのと同じ理由である．

一方，オブジェクトの直接的参照関係を 1 つの索引要素とするマルチインデックスでは，更新は高速であるが，検索に時間がかかるという欠点がある．これは，XML 文書を木構造ととらえ，この XML 木を検索条件に沿って探索するのと同種の処理ととらえることができる．したがって，XML 木の親子関係を基本とした索引においてもマルチインデックスと同種の欠点が存在する．

これらに対処する方法として XML 文書のエレメントを名前や子節点情報などによりクラスタリングすることにより索引要素数を減少し，経路探索に関しては高速化することが可能となる [3]〜[5], [7], [8]．ただし，XML ドキュメントの更新時には索引の更新部分が大きくなってしまう．

また，他の問題としてドキュメント順の問題がある．ドキュメント順とは，文書の形にした（つまり文字の連続として表現した）ときの，現れる順序のことである．つまり，各ノードについてそのノードを表したものが XML 文書において文字として現れる順序を調べて，その順に並べたものである．XPath を検索条件とする検索においては，その結果はドキュメント順に従わなければならない．XML 木においては深さ優先探索を行

うことでドキュメント順で結果が得られるため，出力の順に関しては問題はない．しかし，索引を用いた場合，ドキュメント順に問題がある．経路情報まで含む索引では，経路情報を用いてドキュメント順を再構成することは可能である．ただし，検索結果が大量のノードであった場合は非常に高コストな処理となる．

また，クラスタリングする方法 [3]〜[5], [7], [8] では，クラスタリングすることによりドキュメント順に関する情報がなくなり，ドキュメント順のための情報をあらかじめ付加するか，XML ドキュメント本体にアクセスする必要が出てくる．XML 文書本体にアクセスする方法は非常に高コストであることは明らかである．また，ドキュメント順のための情報をあらかじめ付加する場合は XML 文書の更新時には再構成が必要となる．

これらの問題点を考えると，更新が頻繁であるような XML 文書に対しては，マルチインデックスと同様の索引を用い，ドキュメント順の情報を保持することが得策と考えられる．

そこで，XML 文書への索引付けにわれわれが開発中の索引分散管理システムを用いることで検索効率の向上を目指す．

索引分散管理システムはオブジェクト指向データベース用に提案されているもので，オブジェクトの参照関係に対してマルチインデックスにより索引付けし，非共有メモリ型並列計算機上に分散配置し，並列処理することにより検索スループットの向上を目指している．また，索引分散管理システムでは任意の順番に要求を処理することが可能であり，データ更新時でも処理順の指定さえ可能であれば，データに対するロックと索引に対するロックを分離することができる．これにより処理の並列性を高めることができる．

本研究では検索条件としては XPath の軸を用い，また，検索結果がドキュメント順になることを保証する．本研究で XPath の軸を検索条件とたのは，一般的なオブジェクト指向データベースにおける検索パス式においては，XPath の軸である following や preceding に相当する直接的表現は存在せず，索引分散管理システムでは直接指定することは不可能であり，索引分散管理システムに XML 文書検索に適応する場合には，簡単に指定可能で，処理可能とする必要があると考えたためである．

2. XPath

XPath（XML Path Language）[2] は，XML データを表す木構造をたどることで，要素や属性といったデータを抽出するための記述方法を規定した仕様である．

以下に，XPath の軸について説明する．

2.1 axis

axis は，コンテキストノードとどのような関係かを指示する．以下は axis で指定できる関係を記す．

（1） child：コンテキストノードの子

（2） descendant：コンテキストノードの子孫

（3） parent：コンテキストノードの親

（4） ancestor：コンテキストノードの先祖

（5） preceding-sibling： コンテキストノードの前方兄弟（兄と弟でいえば兄）

（6） following-sibling：コンテキストノードの後方兄弟（兄と弟といえば弟）

（7） preceding：コンテキストノードの前方にあるノード

（8） following：コンテキストノードの後方にあるノード

（9） attribute：コンテキストノードのアトリビュート

（10） namespace： コンテキストノードのネームスペースノード

（11） self：コンテキストノード自身のみ

（12） descendant-or-self： コンテキストノードとそのコンテキストノードの子孫ノード

（13） ancestor-or-self：コンテキストノードとそのコンテキストノードの祖先ノード

本システムでは，検索条件として（1）〜（8）の 8 つを用いる．

3. 親子索引を使った軸の検索

3.1 索　　引

XML 文書中の要素の親子関係を親要素と子要素の 2 種類の索引として作成する．

この索引のキー値は XML 文書中の要素を表す EID（Element ID）であり，データ値はその要素の親要素もしくは子要素の EID である．EID は，IID（Instance ID），TID（Tag ID），NO（兄弟要素の何番目かを表す番号）からなる．

また，索引は二種類の B+tree に格納する．その二種類の B+tree は，以下の通りである．

- child B+tree

キーとなる要素の子要素をデータ値として返す．

- parent B+tree

キーとなる要素の親要素をデータ値として返す．

3.2 検 索 条 件

検索条件として XPath の軸を用いる．

また，検索条件には固有の検索条件番号がつけられており，以下に各軸における検索処理の概要を述べる．

（1） child

child B+tree を使い，指定要素の子要素を求める．

（2） parent

parent B+tree を使い，指定要素の子要素を求める．

（3） descendant

指定要素の child を再帰的に繰り返すことにより子孫要素を求める．

（4） ancestor

指定要素の parent を再帰的に繰り返すことにより子孫要素を求める

（5） fsibling（following-sibling）

parent で指定要素の親要素を求め，その求めた親要素の子要素を child で求め，求めた子要素の NO を指定要素の NO と比較し，指定要素の NO より大きい要素を求める．

（6） psibling（preceding-sibling）

parent で指定要素の親要素を求め，その求めた親要素の子要素を child で求め，求めた子要素の NO を指定要素の NO と比

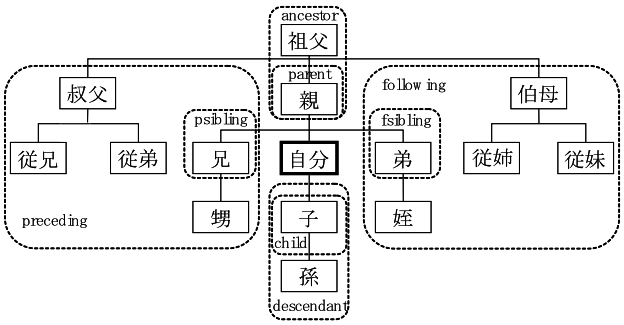

図 1　検 索 結 果

較し，指定要素の NO より小さい要素を求める．

（7） preceding

psibling で指定要素の前の兄弟要素を求め，その子孫要素を descendant で求める．次に親要素に起点を移し，同じように前の兄弟要素とその子孫要素を求めていく．このように最初の子孫要素を求める以外は再帰的な処理で求める．この再帰的処理の際に，検索要求要素の ancestor は結果に含まない．

（8） following

指定要素の子孫要素を descendant で求める．次に fsibling で指定要素の後ろの兄弟要素を求め，その子孫要素を descendant で求める．次は親要素に起点を移し，同じように後ろの兄弟要素とその子孫要素を求めていく．このように preceding と同様の再帰的な処理で求める．この再帰的処理の際，検索要求要素の ancestor は結果に含まない．

図 1 に,「自分」を検索の起点とした場合の各検索結果の例を記す．

4. 索引分散管理システムへの適応

索引分散管理システムの概要について述べる．索引分散管理システムを構築する並列環境としてメッセージ通信非共有メモリ型並列計算機を仮定する．また，各 Processor Element（以下，PE）での処理は 1 プロセスの逐次処理とする．

PE は HOST，検索 PE，DETECTOR の 3 種類に分類される．以下にそれぞれの PE の処理の概要を記す．また，図 2 は検索時の実際のシステムにおける物理的な処理を表す．ここで，図中の三角形は索引の一部を表している．また，図 2 における交差した実線の矢印は HOST，検索 PE 間での相互の通信を簡略化して表現しており，実際には，検索 PE への通信は検索要求であり，HOST への通信は検索結果の送信である．

4.1 HOST

検索要求，更新要求などの外部からの処理要求を検索 PE, DETECTOR に送信し，その処理結果を集計し，最終的な処理の終了を判定するための PE である．このとき，外部からの処理要求に対して要求識別子（RID）を付加する．RID には自然数を用い，小さい順に処理要求に付加する．また，処理要求として送信したメッセージ総数の情報を DETECTOR に送信する．

4.2 検 索 PE

実際に索引を格納し，検索，更新処理などを処理するための

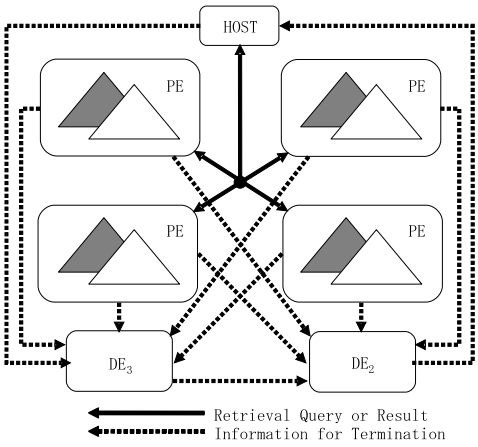

図 2　物理的な検索の流れ

PE である．

各検索 PE は受信した検索要求を処理し，その検索結果を次の検索要求として対応する検索 PE へ送る．1 つの検索 PE での 1 回の処理を 1step とする．各検索 PE 間の通信メッセージは，以下からなる．

- RID（Request ID）

RID は HOST から発行された検索要求に対する識別子のことである．

- step 数

送信側での step 数である．

- 検索命令

検索命令とは，送った検索要求を次の検索 PE で具体的にどのような処理をすべきか，という情報である．これは次の step の検索 PE に送る前に検索条件と前の検索命令から求める．検索命令は，次の 3 種類の命令に分けられる．

1. 1 つの EID に対して child B+tree のみを使う命令
2. 1 つの EID に対して parent B+tree のみを使う命令
3. 1 つの EID に対して両方の B+tree を使う命令

それぞれの命令には命令番号がつけられており，実際にはその命令番号を検索命令として送受信している．そして，各検索 PE は受け取った検索命令に従って分岐して処理する．

- 検索条件

検索条件は，先の 2.1 で説明した 8 つの条件を表した番号のことである．

- FEID（検索命令 1step 前の EID）

FEID（1step 前の EID）は兄弟要素を調べるときに用いられる．例えば,「自分」の fsibling，もしくは psibling を求める際に,「自分」の EID の NO と比較しなければならず,「自分」の親要素の child の結果でどれが「自分」であるかは判断できないため，1step 前の EID，すなわち,「自分」の EID を保持する必要がある．

- EID

EID は次の step でのキー値となる．

検索 PE がそれぞれの検索命令を受け取った時の処理の流れを説明する．

（1） 検索命令が 1 のとき

– child B+tree を EID で検索する．

– 検索条件が fsibling，psibling なら，検索結果の EID の NO を FEID（1step 前の EID）の NO と比較して，それより大きいものを fsibling の検索結果，小さければ psibling の検索結果とする．

– 検索条件が child，descendant，preceding，following なら，検索結果の EID の NO 全てを検索結果として HOST に送信する．

– 検索条件が descendant，preceding，following なら次の検索命令を 1 として，検索結果の EID を新たな検索要求として次の step の検索 PE に送る．

（2） 検索命令が 2 のとき

– parent B+tree を EID で検索する．

– 検索条件が parent，ancestor，preceding なら，検索結果の EID を検索結果として HOST に送信する．

– 検索条件が ancestor なら，次の検索命令を 2，fsibling，psibling なら，次の検索命令を 1，preceding なら次の検索命令を 3 として，検索結果の EID を新たな検索要求として次の step の検索 PE に送る．

（3） 検索命令が 3 のとき

– child B+tree を EID で検索する．

– 検索条件が following，preceding なら，検索結果の EID の NO を FEID（1step 前の EID）の NO と比較して，それより大きいものを following の検索結果，小さければ preceding の検索結果とする．

– 次の検索命令を 1 として，検索結果の EID を新たな検索要求として次の step の検索 PE に送る．

– 次に parent B+tree を EID で検索する．

– 検索条件が preceding なら，検索結果の EID を検索結果として HOST に送信する．

– 次の検索命令を 3 として，検索結果の EID を新たな検索要求として次の step の検索 PE に送る．

4.3 DETECTOR

DETECTOR（以下，DE）は HOST から発行された要求に対する処理が終了したか否かの判断を行うための PE である．1 つの検索要求に対する処理を step ごとに分けて考えると，以下の全ての条件が満たされたとき，その step（n step）の処理が終了したものと判断できる．

1. n-1 step 目の処理が終了
2. n step 目の処理結果としての検索要求が全て到着し，それに対する処理が全て終了

したがって，n-1 step 目の処理終了時の n step 目への検索要求メッセージ数が n step の受信して処理したメッセージ数と等しくなったときに n step 目の処理が終了したものとみなせる．

この終了判定を実現するために DE には各 RID の各 step に対して以下の情報を保持し，HOST，検索 PE からの終了判定情報をもとに更新し，終了判定を行う．

- 受信予定数
- 受信数（=処理数）
- 検索 PE への送信数
- HOST への送信数

これらの情報は検索 PE からの終了判定情報をもとに更新を行う．検索 PE からの終了判定情報としては 1 つの検索要求を受信，処理終了後に以下の情報を DE に送信させる．

- RID
- step
- 受信数
- 検索 PE への送信数
- HOST への送信数

これらの終了判定情報をもとに，対応する RID，step の受信数，検索 PE への送信数，HOST への送信数を更新していく．更新の結果，ある step で終了が判定された場合，その送信数を次の step の受信予定数として記録する．ただし，1 step 目の受信予定数は HOST からの検索要求メッセージ数を記録しておく．ここで，ある step の処理は終了したが検索 PE への送信数が 0 であった場合，それ以降の検索 PE での処理は行われないことになり，その RID に対する検索 PE での処理は終了したことになる．この場合は，各 step の HOST への送信数の合計を HOST の受信予定メッセージ数として HOST に送信し，HOST はこの情報をもとに終了判定を行う．これらの操作は同一 RID 内での処理であり，RID が違うものに関しては別々に行うものである．

5. ドキュメント順の復元

XPath における XML 文書検索では，検索結果はドキュメント順であることが必要である．しかし，索引分散管理システムにより XML 文書の検索を行った場合，索引を分割し，並列に処理を行うため，検索結果は単なる集合でしかない．そこで，検索結果をドキュメント順にソートする必要がある．一般的にはラベル付けを行う方法が多い [6]．しかし，ラベル付けは XML 文書の更新等の際に，ラベルを付け直さなければならず，XML 文書更新時には高コストな処理が必要となる．そのため，本研究では検索途中の NO の情報を利用することでドキュメント順にソートする方法を取り入れる．

ソートを可能とするために各検索 PE において以下の処理を追加する．

- 各検索結果の各要素には処理時の NO 情報を順に付加する．
- ただし，Parent B+tree を使用する場合は 0 を付加する．

上記の処理を各処理段階で行い，HOST においてドキュメント順にソートを行う．また，このように付加された NO 情報を NO 履歴と呼ぶこととする．

検索が終了し，全ての検索結果とその NO 履歴が到着したとき，NO 履歴を用いてソートを行う．NO 履歴の長さは各要素によって異なることが考えられる．このような場合は，HOST において，最長の NO 履歴の長さにあわせて後端に N（NULL）を入れて，各 NO 履歴の長さを統一しておくこととする．各検索結果の要素の NO 履歴を以下の規則に基づき順序付けを行うことでドキュメント順を復元することができる．

- 基本的には，0 < N < 自然数であると考え，0 の方が前

に現れる要素である．

- following だけは，N < 自然数 < 0 である．
- 自然数同士の場合は小さいものを前とする．

以上の規則でソートした結果はドキュメント順となる．これは本手法が XML 木上での親または子節点への探索の連続によって行われ，その情報が NO 履歴によって保持され，NO 経歴を用いることで探索経路を木構造として復元することが可能となり，この XML 木の部分木を上記規則を用いることで深さ優先探索，つまりドキュメント順で辿ることが可能となるためである．

図 1 の preceding と following を例として考えると，検索結果は

兄	(0,	1,	N,	N)	弟	(0,	3,	N,	N)
甥	(0,	1,	1,	N)	叔母	(0,	0,	3,	N)
叔父	(0,	0,	1,	N)	姪	(0,	3,	1,	N)
従兄	(0,	0,	1,	1)	従妹	(0,	0,	3,	1)
従弟	(0,	0,	1,	2)	従妹	(0,	0,	3,	2)
	preceding					following			

となり，上記の規則で復元した結果は以下の通りドキュメント順となる．

叔父	(0,	0,	1,	N)	弟	(0,	3,	N,	N)
従兄	(0,	0,	1,	1)	姪	(0,	3,	1,	N)
従弟	(0,	0,	1,	2)	叔母	(0,	0,	3,	N)
兄	(0,	1,	N,	N)	従妹	(0,	0,	3,	1)
甥	(0,	1,	1,	N)	従妹	(0,	0,	3,	2)
	preceding					following			

6. 評価実験

提案した NO 履歴を用いることで検索結果をドキュメント順に復元する手法を実装し，評価を行った．提案手法との比較対象は検索結果をソートしない従来の検索方法とする．

6.1 実験条件

実験は以下の条件で行った．

6.1.1 対象索引

以下のような XML 文書を想定し、索引付けを行った．

- 要素数は約 22000 個，木構造で 8 段．
- ルート要素を EID（IID，TID，NO）＝（1，1，1）とする．
- ルートから順に TID を付け，IID は段ごとに付ける．
- ルートから各要素は子要素を 4 つ持っている．

6.1.2 索引分散管理システム

実験に用いる索引分散管理システムは以下の条件からなる．

- HOST を 1 個，検索 PE は 8 個，DE は 5 個用意する．
- 通信には MPI ライブラリ、索引要素の格納に 2 次記憶上の B+tree を使用する．
- 索引の分割方法は，XML 文書を段ごとに分割する方法を用いる．

	検索結果 [個]	復元なし [s]	復元あり [s]
child	4	0.002113	0.002157
parent	1	0.001860	0.001821
ancestor	7	0.003357	0.003421
descendant	約 1000	0.300352	0.453626
fsibling	3	0.002636	0.002711
psibling	1	0.001806	0.001847
preceding	約 500	0.150786	0.177333
following	約 500	0.156549	0.194352

表 1 実験結果

6.1.3 その他の条件

実験における条件を以下に挙げる．

（1） 検索の基点となる要素と軸の組をランダムに指定する．

（2） 検索要求は連続には行わず（すなわち，RID = 1），各検索条件による検索を行う．

（3） これらの検索を 10 回繰り返して平均時間を測定し，比較・評価を行う．

以上の条件で Sun Fire E4900（コア数：24，メモリ：48GB）上に実装し，評価を行う．また，HOST が検索要求を発行した時刻から，HOST がその要求の終了を判定した時刻までを検索時間とする．

6.2 実験結果

表 1 に実験結果を示す．各検索要素におけるドキュメント順の復元をする方法と従来の復元をしない方法での検索時間を表したものである．

検索結果が少ない場合は従来の方法（ドキュメント順に復元しない方法）と比べても大きな差はないが，検索結果が大規模になった場合，比較回数が増え，処理時間が増加する．

7. おわりに

本研究では，XML 文書の各エレメント間の親子関係を索引化し，その索引を分散管理システムに管理し，その検索結果をドキュメント順にソートする手法を提案，評価した．本手法によるドキュメント順にソートする手法は検索結果が大規模になった場合，比較回数が増え，処理時間が増加する可能性は否めない．しかし，検索結果は各 PE でもソート可能であることから，マージソートを使用することが可能であり，比較回数の減少が見込まれる．今後の課題としては，他のドキュメント順を復元する方法との比較・評価が課題の 1 つである．また，各 PE でのソートによる処理時間の短縮や，軸が連接する場合のドキュメント順の復元方法があげられる．

文献

[1] Extensible Markup Language, URL:http://www.w3.org/XML/

[2] XML Path Language, URL:http://www.w3.org/TR/xpath/

[3] Ke, Y., Hao, He., loana, S. and Jun, Y., Incremental Maintenance of XML Structural Indexes, Proc.SIGMOD 2004, pp.491-502（2004）

[4] Raghav, K., Phlip, B., Jeffrey, F.K. and Pradeep, S., Updates for Structure Indexes, Proc.28th VLDB, pp.239-250（2002）

[5] Braian, F.C., Neal, S., Michael, J.R., Gisli, R.H. and Moshe, S., A Fast Index for Semistructured Data, Proc.27th VLDB,

pp.341-350（2001）
[6] Quanzhong, L. and Bongki, M., Indexing and Querying XML Data for Regular Path Expressions, Proc.27th VLDB, pp.361-370（2001）
[7] Roy, G. and Jennifer, W., DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases, Proc.23rd VLDB, pp.436-445（1997）
[8] Raghav, K., Philip, B., Jeffrey, F.N. and Henry, F.K., Covering Indexes for Branching Path Queries, Proc.SIGMOD 2002, pp133-144（2002）
[9] Michal, K. Jaroslav, P. and Vaclav, S., Implementation of XPath Axes in the Multi-dimensional Approach to Indexing XML Data, EDBT 2004 Workshops, pp219-229（2004）