DEWS2005 5A-i5

アクセス頻度を考慮した XML 文書分割方式の提案

中尾　伸章† 　天笠　俊之† 　的野　晃整† 　植村　俊亮†

† 奈良先端科学技術大学院大学情報科学研究科　〒 630–0192 奈良県生駒市高山町 8916–5
E-mail: †{nobua-n,amagasa,akiyo-ma,uemura}@is.naist.jp

あらまし　本論文では，XML 文書への問合せ処理を効率化するための XML 文書分割方式を提案する．近年，XML フォーマット利用の拡大と共に様々な XML 文書が生成されている．サイズが数百 MB から数 GB にもなる大規模な XML 文書もその一例である．大規模な XML 文書はサイズの大きさから処理コストが高くなる．しかし，一般に XML 文書内の要素や属性毎の使用頻度にはばらつきがあるため，XML 文書を分割し，必要部分のみを使用することによって処理の効率化が期待できる．本論文では，アクセスの偏りを元に XML 文書を分割し，問合せ処理を効率化する手法を提案する．アクセスの偏りとして，問合せの評価に必要な部分の局所性と，問合せの発行頻度を利用する．本手法では，アクセスの偏りを導く情報として，XML 文書への典型的な問合せセットとそれらの発行頻度が既知であることを仮定する．問合せ毎にアクセスする部分を断片の候補にしておき，それらをコスト関数に基づいて併合することで，XML 文書を任意の数へと分割する．コスト関数の値を元に，問合せを効率化する併合の組合せを欲張り法により決定する．また，DataGuide に基づく構造概要を生成し，分割方針の決定および問合せ処理に利用する．
キーワード　XML, 問合せ処理，半構造データ，XPath，DataGuide

A Scheme for Partitioning XML Documents Based on Access Frequency

Nobuaki NAKAO†, Toshiyuki AMAGASA†, Akiyoshi MATONO†, and Shunsuke UEMURA†

† Graduate School of Information Science, Nara Institute of Science and Technology (NAIST)
8916–5 Takayama-cho, Ikoma-shi, Nara 630–0192, Japan
E-mail: †{nobua-n,amagasa,akiyo-ma,uemura}@is.naist.jp

Abstract　In this paper, we propose a scheme for partitionig XML documents by taking locality of data access into account. By the spread of XML format, large-scale data described with XML are increasing. There are such XML documents whose sizes are several GB. In large-scale XML documents, the processing cost grows rapidly depending on data size. However, in general, there exists locality in XML documetns access, that is, we can find some portions that are accessed more frequently than elsewhere. In our scheme, for XML being processed, we assume that typical queries and their frequencies are known in advance. Based on the query set, we first derive the base partitions, each of which corresponds to a query in the set. In order to obtain the final partitioning, we reduce the number of partitions by integrating the partitions using a greedy method, whose cost function is defined in terms of data size and query frequency. The effectiveness of the proposed method is demonstrated by experiments.
Key words　XML, query processing, semi-structured data, XPath, DataGuide

1　はじめに

XML (Extensible Markup Language) [11] は，データ記述フォーマットとして急速に普及し，その関連技術の研究及び実装は現在も盛んに行われている．XML 文書はタグの入れ子関係から任意の木構造を表現することが可能であり，タグ内の要素，属性に名前を持たせることで自己記述的な表現が可能である．このような柔軟な表現力や仕様の簡明さ，インターネット標準であること等が要因となり，XML 文書は WWW (World Wide Web) 上をはじめ多く用途で増加している．数百 MB から数 GB のデータサイズを持つ大規模な XML 文書もその一例である．観測記録，サーバのアクセスログ，Web ディレクトリ，オントロジなどの記述フォーマットがその例として挙げられる．たとえば，Web ディレクトリの情報を XML で記述し

ている Open Directory Project (OPD, dmoz) [4] のデータはおよそ 1.8 GB(注1) の XML 文書である．このことから，規模の大きな XML 文書の高速な処理が望まれている．

XML 文書の処理では，XML 専用の API である DOM (Document Object Model) [10] や SAX (Simple API for XML) [9] が主として使用される．DOM は XML 文書を主記憶上にオブジェクトとして展開した上で処理を行う．SAX は，XML 文書を先頭から最後までパースしながら要素の開始や終了などのイベントを発生させ，アプリケーションがイベントを受取った時に処理を行う．よって，XML 文書のデータサイズは，DOM や SAX における処理コストに多大な影響をおよぼす．XML データベースを用いた処理においてもデータサイズは大きな要因となる．これは，ノード単位で管理を行う従来の手法では木構造の復元に用いる結合処理の計算量が大きいため，データサイズの増加に伴うノード数の増加により処理コストが増加するためである．

この問題の解決手段として，アクセスの偏りが利用できると考えられる．一般に，XML 文書の問合せ処理に必要な部分は必ずしも文書全体ではなく，局所的であることが多い．また，問合せが発行される頻度も異なることから，XML 文書内のアクセスはより偏っていると考えられる．そこで本論文では，XML 文書内のアクセスの偏りを考慮し，問合せ処理が効率化されるように XML 文書を分割する手法を提案する．具体的には，XML 文書への典型的な問合せセットとそれらの発行頻度が既知であることを仮定し，それらを元にアクセスの偏りと処理コストを考慮し，XML 文書を任意の数へと分割する．まず各問合せに必要な部分を断片の候補とし，それらを任意の数にまで併合することで，分割後の断片を規定する．併合の判断にはコスト関数を用いる．コスト関数は，問合せの処理コストと，問合せの発行頻度に基づいて算出する．その上で，欲張り法によって，コスト関数が低くなるよう併合を行う．また，Strong DataGuide [5] に基づく構造の概要を生成し，断片を規定する問合せ式の決定や，分割後の問合せ処理に利用する．問合せ処理時には，構造概要を元に問合せを解析，変換し，必要なノードが存在する断片のみに問合せを発行することで処理を効率化できる．

2 関連研究

まず，XML 文書の分割に関する研究として Bremer 等の研究 [3] について述べる．[3] は，XML 文書を分散処理するための研究であり，Repository Guide と呼ばれる索引付けされた構造概要を利用している．まず XML 文書の水平・垂直分割を XPath 式によって定義し，分割された文書を Repository Guide によって管理する．Repository Guide は三種類の索引に関連付けられ，分散環境での問合せ処理を効率化している．また，[2] の手法に基づき，ノードのラベルを符号化した Dewey Order [7] に基づいて生成し，索引部分を軽量化している．しかし，分散環境下での処理に注目しており，分割方法や問合せの変換，アクセスの偏りを考慮していない点が本研究と異なる．また，Bohannon 等は，用途に合せて処理を効率化できるように，処理コストを考慮して XML 文書を複数の関係表へと自動的にマッピングする手法を提案している [8]．この研究は，処理コストを考慮して複数の格納（記述）先へと分けるという点で我々の研究に近いが，関係データベースへの格納を前提にしている点や，いくつに分割するかを任意に決定できないという点で本研究とは異なっている．

次に，問合せに必要な部分の局所性に注目した研究について述べる．中島等は，処理に必要な部分のみを主記憶上に展開して処理できる SPlitDOM [17] を提案している．Marian 等は，XQuery [12] 問合せを静的に解析し，XML 文書内の必要な部分のみを射影して処理する手法を提案している [6]．横山等は，発生イベントとそれに付随するデータの保存先をアクセス頻度に応じて切り分けることで SAX における処理時間を短縮する手法を提案している [15]．また，我々は関係データベースに格納された大規模 XML データの処理において，問合せ処理に必要な部分のみを動的にマッピングして使用することで高速化する手法を提案している [14]．

我々は，これまでアクセスの偏りに基づいた XML 文書の部分圧縮とそれによる問合せ処理の効率化手法について研究してきた [16]．本論文は分割によるアプローチであり，分割によって処理するデータサイズを限定することで効率的な問合せ処理を実現する手法である．

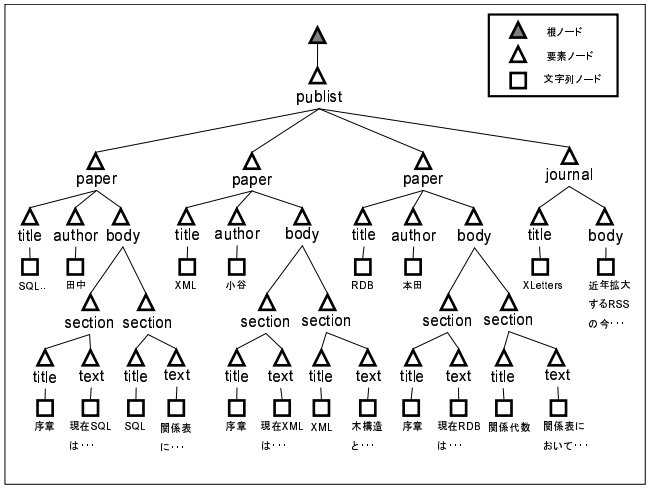

図 1　XML 文書例

3 提案手法

3.1 XML データモデル

まず，本論文で想定する XML 文書のデータモデルを示す．

【定義 (1)】 XML 文書

XML 文書 D は三つ組 $D = (V, E, r)$ で定義される．ただし，V はノード集合，E はエッジ集合，r は根ノードであり $r \in V$ である．

次に，本手法で用いる XML 文書の構造概要を示す．XML を含む半構造のための構造概要はこれまで多く提案されているが，ここでは最も単純な Strong DataGuide [5] を用いる．

(注1)：2004 年 12 月現在

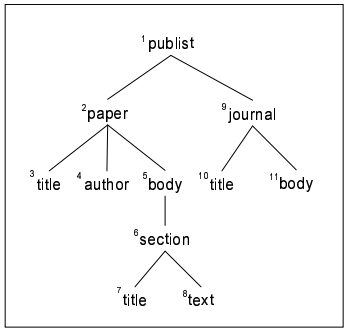

図 2　図 1 の XML 文書に対する構造概要

【定義 (2)】 Strong DataGuide
情報源 s の DataGuide d とは，1) 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いう．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 $(e_1, e_2, ..., e_n)$ のラベル系列 $(l_1, l_2, ..., l_n)$ 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 $l_1, o_1, l_2, o_2, ..., l_n, o_n$ である．Strong DataGuide sd とは，このような d のうち，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．

図 1 で示される XML 文書に対して定義 (2) に基づいて構造概要を生成した場合図 2 のようになる．ただし，構造概要上の各ノードの左上に付加してある数字は，構造概要上での各ノードを識別する値であり，後に利用する．

次に，問合せ式に基づく XML 文書の垂直分割 (*vertical partitioning*) を定義する [3]．ただし，本論文で使用する問合せ式は XPath のサブセットで表現し，“/” 及び “//” 軸とノードテストからなる．ただし，述語は含まないものとする．

【定義 (3)】 XML 文書の垂直分割
垂直分割で規定される断片 f は $f = sf - \{ef_1, ..., ef_m\}$ の式で定義される．sf は選択される断片を指定する問合せ式で表現される．また，ef は除かれる断片を，sf が指す部分木の根ノードからの相対パスで指定した問合せ式によって表現される．ただし，$\{ef_1, ..., ef_n\}$ は無くてもよい．

[3] では XML 文書の水平分割 (*horizontal partitioning*) も定義しており，それは垂直分割で規定した断片 f において，sf, ef を表現する問合せ式が述語を含んでいる時，f を水平分割としている．本論文では問合せ式に述部を含まないため，水平分割は考えない．

3.2 分割処理

本論文では，XML 文書に対する問合せセットとそれらの発行頻度が既知であることを仮定する．発行頻度は問合せの発行履歴の統計から算出した確率を元に，上位 n 件の問合せを典型的な問合せセットとし，その確率を用いる．図 1 に示す XML 文書に対する典型的な問合せセットとそれらの発行頻度の例を表 1 に示す．

問合せセットが既知である場合，あらかじめそれぞれの問合せ式を元に定義 (3) による分割を行い，断片を規定しておけば，問合せ処理時にはそれを返すだけで済むので効率的である．しかし，問合せセットは一般に多数あるので，そのまま素直に分割すると断片の数が余りにも多くなってしまう．また，後で述べる重複部分の問題も生じるので，一般にはうまく行かない．そこで本論文では 任意の n 個の断片へと分割することを考える．ここで n は，想定する環境によって自由に決めることができる．例えば，分散環境ではプロセッサの数に基づいて決めれば良いし，分散ディスクの場合はスピンドルの数などである．

表 1　典型的な問合せセットとそれらの発行頻度例

q	問合せ式	発行頻度
q_1	//title	0.5
q_2	//section	0.3
q_3	//author	0.1

以上より，問題は XML 文書を m 個の問合せ式に基づいて n 個の断片に分ける問題となる．ここで一般に $m > n$ である．提案手法では，まず与えられた m 個の問合せ式で規定される断片を候補の断片としておき，その m 個の断片のいくつかを併合することで n 個にまで減らすことを考える．併合の決定は，問合せ処理のコストに問合せの発行頻度を考慮したコスト関数を用いる．これにより，問合せ処理を効率化する分割を行うことができる．

3.2.1 重複したノードの取り扱い

一般に問合せ式に対応するノード集合は独立ではなく互いに重複部分を持つことがあるので，考慮が必要である．例として，図 1 において問合せ $q_1 =$ //title と $q_2 =$ //section を考える．この場合，title 要素のいくつかが section 要素に含まれている．q_1, q_2 に基づく断片 f_1, f_2 を求める場合，次のような方針が考えられる．

1) 重複部分を複製しない
2) 重複部分を複製して f_1 と f_2 両方にもたせる

2) は問合せの処理速度の面から有利であるが，余分な記憶容量を必要とすることや更新時の処理が複雑になる．以上の理由から本論文では，重複部分の複製は考えないこととする．

1) の方針はさらに下の二つのケースに分けることができる．

1a) 重複部分を新たな部分文書とし，元の問合せ式に対応する断片からはその部分文書を除く
1b) 重複部分をどの問合せ式に対応する断片に持たせるかを，頻度の高さや処理コストなどから決定する

本論文では，1b) の方針を取り，重複部分は発行頻度が最も高い問合せ式に基づく断片が持つこととする．このことから，他の断片に完全に包含されてしまうような断片において，その断片を規定する問合せの発行頻度がより低ければその断片は除外されることとなる．重複部分を持つ二つの問合せによる断片の規定について以下に示す．問合せ式 q を評価するために必要なノード集合を $v(q)$ で示し q の発行頻度を $freq(q)$ で示すとき，q_1, q_2 $(freq(q_1) > freq(q_2))$ なる問合せから二つの断片 f_1, f_2 を規定する場合を考える．$v(q_1) \cap v(q_2) = \phi$ であれば重複部分が無いので，各断片は，$f_1 = q_1, f_2 = q_2$ で規定すれば

表 2　各問合せから規定した断片とその情報

断片	断片を規定する式	ノード総数	処理対象とする問合せ
f_1	//title	10	q_1, q_2
f_2	//section − {./title}	12	q_2
f_3	//author	3	q_3
f_{remain}	/publist − {./paper/title, ./journal/title, ./paper/author, ./paper/body/section}	9	-

よい．$v(q_1) \cap v(q_2) \neq \phi$ のとき，重複部分 $v(q_1) \cap v(q_2)$ は発行頻度がより高い q_1 で規定された f_1 にのみに含める．重複部分を示す問合せ式は q_1, q_2 と元の XML データの構造概要から作成できる．その問合せ式を q_{2-1} とすると，各断片は，$f_1 = q_1, f_2 = \{q_2\} - \{q_{2-1}\}$ で与えられる．たとえば，表 1 の問合せ式 //title と//section の場合，構造概要（図 2）から，重複部分が存在することがわかる．さらに，構造概要から重複部分を示す問合せ式を section 要素からの相対パスで ./title のように表現できる．問合せの発行頻度から，重複部分は //title の規定する断片が持つため，重複部分が除かれた，//section を元に規定される断片の式は //section − {./title} となる．

以上により，問合せ式の数だけ候補となる断片を規定することができる．表 1 の問合せセットを用いて 図 1 の XML 文書を分割する場合，候補となる断片は表 2 のようになる．ただし，f_{remain} は，全ての問合せ式から指定されないノード集合からなる断片を示すものとする．表 2 のように，各断片を規定する式に加え，ノード数や処理対象とされる問合せ等，問合せの処理コストに関わる情報を収集しておき，後で利用する．

3.2.2　断片の併合

断片の候補を併合することによって n 個の断片を決定する．断片の併合を考慮した XML 文書の垂直分割を，定義 (3) の拡張によって次のように定義する．

【定義 (3)'】 併合を考慮した XML 文書の垂直分割
断片の併合によって規定される断片 f は $f = \{sf_1, ..., sf_n\} - \{ef_1, ..., ef_m\}$ の式で定義される．sf は選択される断片を指定する問合せ式，ef は除かれる断片を指定する問合せ式によって表現される． ただし，$\{ef_1, ..., ef_n\}$ は無くてもよい．なお，併合を考慮した断片の規定では，sf を表す問合せ式が複数存在するため，定義 (3) とは異なり ef は sf の問合せ式のいずれかを接頭辞とする絶対パスで指定した問合せ式によって表現する．

本手法では，併合後に問合せセットで問合せ処理を行った場合の処理コストと，各問合せの発行頻度からコスト関数を定め，その値から併合の組合せを決定する．コスト関数が最小となる組合せで併合を行うことで，問合せ処理を最も効率化するように併合することができる．問合せセット全体を処理するコスト関数 C_{total} は次式で与えられる．

$$C_{total} = \sum_{k=1}^{m} freq(q_k) \cdot C(q_k)$$

ここで $freq(q_k)$ は q_k の発行頻度，$C(q_k)$ は q_k の評価にかかる処理時間コストである．

$C(q_k)$ は問合せに必要な全ての断片に問合せを発行し，評価するコスト C_{eval} と，必要に応じて結果を統合するためのコスト $C_{integration}$ の和で次のように与えられる．

$$C(q_k) = C_{eval}(q_k) + C_{integration}(q_k)$$

$C_{eval}(q_k)$ は 必要なノードが含まれる断片全てに対して q_k を評価するためのコストである．たとえば，断片 f_a 内に q_k の評価に必要な全てのノードがあれば，評価する断片は f_a のみでよい．しかし，そうでない場合，f_a だけでなく重複部分を持つ他の断片も評価する必要がある．f_a 以外で q_k にどの断片が必要になるかは，問合せセットから規定される断片の情報（表 2）から判断することができる．また，$C_{integration}(q_k)$ は，複数の断片から得た結果を結合やソートによって統合するためのコストである．

C_{eval} は問合せの処理方法によって異なる．処理方法が既知の場合はそれに関する情報を収集しておき，推測値として算出する．ただし，処理方法が未知の場合や，分割後のデータへの処理方法が複数ある場合は，ノード数やデータサイズ等から推測する．また，$C_{integration}$ についても処理方法の他，結合処理の方法によって異なる [1]．たとえば，Stack-Tree Join アリゴリズムの一手法である Stack-Tree-Desc に基づいて結合処理を行う場合，結合の候補となる先祖のノード集合のサイズ，子孫のノード集合のサイズ，結合結果のノード集合のサイズの合計に計算量が依存するため，表 2 で示したような断片の候補に関する情報と構造概要から該当するノード数を割り出して計算する．

断片の合計が n となるような断片の候補の組合せ毎に C_{total} を計算する．併合の組合せの数だけ算出できる C_{total} の中で最小のものが問合せ処理を最も効率化する組合せとなる．しかし，m 個の断片を n 個に併合するとき，その全てのコスト関数を算出する計算量は $O(n^m)$ となってしまう．よって，本手法では，組合せ最適化問題の近似解法である欲張り法 (Greedy Method) を用いることで，近似解となる組合せを決定する．

欲張り法とは，目的関数値の良さを示す局所的評価（部分問題の解）に基づいて，可能解を直接構成していく方法である．部分問題毎に最適解を求め，それを繰り返すことで最適解の近似解を得る．本手法では，欲張り法に基づいて，最適な併合の組合せの近似解となる組合せを求める．具体的には，二つの断片の候補のみを併合した場合の C_{total} を部分問題とし，C_{total} が最も低くなる断片の候補同士を併合する．次に，$m-1$ 個となった断片の候補に対し，同様に C_{total} を求め，併合を行う．このような一組ずつの併合を部分問題とすることで，併合を $m-n$ 回行うことで最終的な併合の組合せを決定する．これにより，求める組合せ数が減少し，コスト関数を算出する計算量

表 3　併合後の断片を規定する式

断片	問合せ式での表現
f_A	{/publist/paper/title, /publist/paper/body/section, /publist/journal/title}
f_B	{/publist} − {/publist/paper/title, /publist/journal/title, /publist/paper/body/section}

```
アルゴリズム combineFragments (F：断片の集合, Q：問合せセット)
begin
  if m > n
    combineTwoFragments(F, Q)
  endif
  return F
end

アルゴリズム combineTwoFragments (F：断片の集合, Q：問合せセット)
begin
  foreach (f_a, f_b) ∈ F から選んだ 2 つの断片
    addCombineFlag (f_a, f_b)
    C_total = calculateCost (Q, F)
    if C_total < oldC_total or oldC_total == null
      oldC_total = C_total
      f_A = f_a, f_B = f_b
    endif
    omitCombineFlag (f_a, f_b)
  end
  addCombineFlag (f_A, f_B)
  m = m − 1
  return m, F
end

アルゴリズム calculateCost (Q：問合せセット, F：断片の集合)
begin
  C_total = 0
  foreach q in Q
    C_total = C_total + freq(q) · C(q)
  end
  return C_total
end
```

図 3　欲張り法による併合アルゴリズム

は $O\,(m^3)$ となる．

欲張り法に基づいて断片の数が n となるよう併合を行うアルゴリズムを図 3 に示す．ただし，*addCombineFlag* (f_a, f_b) は，f_a と f_b を併合するようフラグ付けし，F に戻す関数とし，*omitCombineFlag* (f_a, f_b) はそのフラグを解除する関数とする．

例として，表 2 の四つの断片の候補を二つの断片に分割するために併合する場合のコスト関数を考える．ただし，簡単のため，C_{eval} は問合せに必要な断片のノード総数のみに，$C_{integration}$ は問合せ結果のノード総数のみに依存するものとして計算する．たとえば，f_3 と f_{remain} を併合する場合の C_{total} を考える．この場合，問合せ毎の $C(q_k)$ はそれぞれ，$C(q_1) = C_{eval}(q_1) + C_{integration}(q_1) = 10 + 0 = 10, C(q_2) = (10 + 12) + 16 = 38, C(q_3) = 12 + 0 = 12$ となり，各々の問合せの発行頻度との積から $C_{total} = 17.6$ となる．また，同様に，f_1 と f_2 を選んだ場合は $C(q_1) = 24 + 0 = 24, C(q_2) = 24 + 0 = 24, C(q_3) = 3 + 0 = 3$ となり，$C_{total} = 19.5$ となる．f_1 と f_3，f_2 と f_3 など，他の組合せについても同様に C_{total} を求める．このようにして最初の併合での C_{total} が最小となるもの見つけ，それを起点として併合を繰り返していく．この場合，f_3 と f_{remain} の併合が起点となり，併合後の三つの断片で再び併合を行っていく．最終結果として，f_1 と f_2，f_3 と f_{remain} を併合してできる二つの断片が C_{total} が低くなる結果として選ばれるため，分割後の二つの断片 f_A, f_B は表 3 のように規定できる．

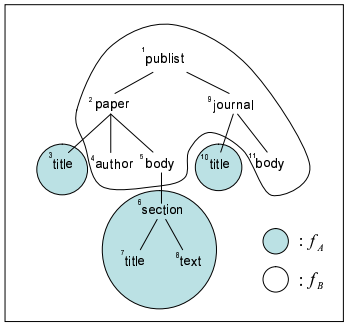

図 4　構造概要上で表現した記述先

併合後，断片を規定する式に基づいて XML 文書を n 個の断片へと分割する．ただし，断片は問合せ式から規定されるため，そのままでは最上位の要素が複数存在する場合がある．XML では最上位の要素を一つに定めているため，各断片の根要素となる要素を生成しておく必要がある．

3.2.3　問合せ処理のための情報付加

分割時に，分割によって破断されるエッジの両端のノードと構造概要にそれぞれ情報を付加しておく．破断されるエッジの両端のノードには，二つの情報を付加する．一つは，結合時に利用する，エッジの識別子である．破断されるエッジの両端のノードは，必要に応じて再び結合によって構造を復元する必要があるため，互いが結合先のノードを特定できるような識別子を付加する．本手法では，元の XML 文書における文書順を識別子として用いる．これにより，結合だけでなく，文書順でのソートが必要な場合でも利用することができる．二つ目は，構造概要上でのノード識別子（図 2 でのノード左上の値）である．破断後の子となるノードでは，根から親までの経路に関する情報を失うため，根からの経路が異なる同名の要素との区別がつかない可能性がある．たとえば，図 1 の XML 文書を表 3 に示す式で分割した場合，f_A に含まれる title 要素は，そのままでは paper 要素の子要素なのか，journal 要素の子要素なのかを判別できなくなる．このため，構造概要上でのノード識別子により，同名のノードが識別不可能になることを防ぐ．

構造概要には，ノード毎に記述先となる断片の情報を付加する．図 4 は，表 3 のように分割した場合に，構造概要上の各ノードに記述先を付加したイメージ図である．実際には各ノードに格納先を示す値を持たせるが，図 4 では格納先が同じノードを同じ色からなる範囲で示してある．これにより，問合せ時に問合せの解析を行うことができる．

3.3　問合せ処理

分割後の各格納先への問合せ処理は図 5 のような流れとなり，問合せの解析，変換，結果の統合から実現される．

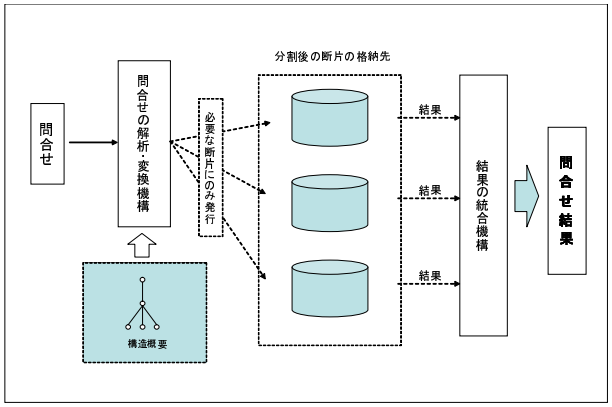

図 5　問合せ処理の流れ

3.3.1　問合せの解析と変換

分割後の問合せ処理では，構造概要を用いて問合せの解析と変換を行う．分割後，構造概要上の各ノードには 3.2.3 節の図 4 のように，各ノードの記述先である断片を示す値が付加されている．この情報を元に，構造情報を用いて問合せの解析，変換を行うことができる．具体的には，問合せ式の経路と一致するノードを元に構造概要をたどり，記述先の断片と，そこに発行すべき問合せを生成する．構造概要を用いて問合せを解析，変換するためのアルゴリズムを図 6 に示す．ただし，*MatchQuery(sd, q)* は構造概要 sd から問合せ式 q の経路式と一致するノードを取り出す関数とし，*addStep(v, q_v)* は，ノード v の名前からなるノードテストを先頭に加えて問合せ式 q_v を再構成する関数とする．また，*getExistFragment(v, sd)* は構造概要 sd からノード v の記述先を取り出す関数，*getParent(v)* は，ノード v の親ノードを取り出す関数とする．また，返される Q_{trans} は変換した問合せ（発行先の情報を含む）の集合である．たとえば，図 1 の XML 文書を 表 3 で示す式によって分割したときに問合せ式 //section を処理する場合を考える．まず，構造概要上の section 要素から，分割後の section 要素が f_A のみに存在することがわかる．次に，構造概要の section 要素を先祖，子孫の方向へたどる．section 要素の親である body 要素は異なる断片に記述されているため，断片 f_A へ発行する問合せ式は断片の最上位に生成した要素を root として /root/section となる．また，子孫方向では異なる断片に記述された要素が無いことがわかる．よって，問合せ式 //section は，断片 f_A への問合せ式 /root/section に変換できる．また，//journal を処理する場合，同様に構造概要上の journal 要素から，まず分割後の journal 要素が f_B に存在することがわかる．この場合，先祖方向は publist 要素まで断片 f_B に記述されていることがわかるため，断片 f_B への問合せは /root/publist/journal となる．次に，子孫方向への探索から，title 要素が断片 f_A に記述されていることがわかるため，断片 f_A への問合せ /root/title も併せて生成できる．ただし，断片 f_A 内に存在する title 要素は，元の XML 文書において paper 要素の子要素であるものと journal 要素の子要素であるものの両方を含む．このため，破断したノードが持つ構造概要上の識別子が 10 という条件でフィルタリングを行う．最後に，得られた結果を，破断されたエッジ部分が持つ情報の比較によって統合することで，最終的な結果を得ることができる．結果の統合については，次で述べる．

```
アルゴリズム analyzeQuery (sd = (V_s, E_s) : 構造概要, q : 問合せ)
begin
  Q_trans = ∅
  foreach v in MatchQuery(sd, q)
    Q_trans = Q_trans ∪ makeQuery(v, sd)
    Q_trans = Q_trans ∪ makeQueryDesc(v, sd)
  end
  return Q_trans
end

アルゴリズム makeQuery (v : ノード, sd : 構造概要)
begin
  R = ∅
  q_v = addStep(v, q_v)
  ef_v = getExistFragment(v, sd)
  u = getParent(v)
  while ef_v == getExistFragment(u, sd)
    q_v = addStep(u, q_v)
    u = getParent(u)
  end
  R = (ef_v, q_v)
  return R
end

アルゴリズム makeQueryDesc (v : ノード, sd : 構造概要)
begin
  R = ∅
  q_v = addStep(v, q_v)
  ef_v = getExistFragment(v, sd)
  foreach u ∈ v の子ノード
    if ef_v == getExistFragment(u, sd)
      makeQueryDesc(u, sd)
    else
      q_v = addStep(u, q_v)
      ef_u = getExistFragment(u, sd)
      R = (ef_v, q_v)
    endif
  end
  returnR
end
```

図 6　問合せの解析，変換アルゴリズム

3.3.2　結果の統合

問合せの変換後，発行先が複数の断片となった場合，それぞれの断片から得た結果をソートや結合によって統合する必要がある．結果の統合は破断したエッジ部分に付加した情報を利用することで実現できる．統合してできた結果が最終的な問合せ結果であり，分割前の XML 文書への問合せ結果に等しいことが保証される．たとえば，図 1 の XML データを 図 4 の構造概要が示すように分割した後，前節のように問合せ //journal を評価する場合，f_A と f_B から得た結果のうち，破断されたエッジの両端にあたるノードを結合する必要がある．この場合，各断片から得られた結果のうち，title 要素と journal 要素は，3.2.3 節で示した，破断されたエッジの両端のノードであるため，結合に必要な情報を持つ．よって，title 要素と journal 要素からエッジの識別子が一致するものを結合する．また，// を含むような問合せでは，複数の断片から得た結果を，必ずしも結合を行わなくてもソートする必要がある．本手法では，破断したエッジの識別子に文書順を利用しているため，その値によってソートを行うことができる．ただし，エッジの識別子には文書順以外にも，Dewey Order [7] に基づく値等，他の識別子を利用してもよい．

4　実　　験

本章では，実験によって提案手法を評価する．実験は，分割

表 5　問合せセットに基づく断片の候補

断片の候補	断片を規定する式	イベント数	処理対象とする問合せ
f_1	//date	3033	q_1, q_4, q_6
f_2	//interval/start	360	q_2, q_3
f_3	//interval − {./start}	962	q_3
f_4	//closed_auction − {./date}	7583	q_4
f_5	//person/name	766	q_5
f_6	/site/region/asia − {./item/mailbox/mail/date}	1597	q_6
f_7	/site/catgraph	30	q_7
f_8	/site から f_1〜f_7 を除いた部分（紙面の都合上省略）	53410	-

表 4　実験で用いた問合せセットとそれらの発行頻度

問合せ	問合せ式	発行頻度
q_1	//date	0.3
q_2	//interval/start	0.2
q_3	//interval	0.15
q_4	//closed_auction	0.125
q_5	//person/name	0.1
q_6	/site/region/asia	0.05
q_7	/site/catgraph	0.025

表 6　併 合 結 果

断片	併合した断片の候補	イベント数	処理対象とする問合せ
f_A	f_1, f_4	10616	q_1, q_4, q_6
f_B	f_2, f_3, f_5, f_6, f_7	3715	q_2, q_3, q_5, q_6, q_7
f_C	f_8	52410	-

における断片の候補の併合と，分割前後での問合せ処理時間の比較を行った．実験環境は，Pentium 4 1.8 GHz の CPU と 1024 MB のメモリを搭載した計算機を用いた．問合せ処理は，SAX を用いた問合せプログラムを Java 言語により実装して用いた．なお，SAX パーサは公開されている Xerces2 Java(注2) を用いた．

4.1　用いた XML 文書と問合せセット

実験に使用したデータは XMark プロジェクト [13] で公開配布されている XML 文書生成プログラム xmlgen によって 1 MB，10 MB のデータを生成して用いた．

問合せセット及びそれらの発行頻度として，表 4 に示す 7 種類の問合せ式を用いた．これらの問合せ式に基づく表 5 のような 8 個の断片の候補を，3 つの断片へと併合し，分割を行った．なお，今回は SAX によって問合せ処理（結果の統合を含む）を行うことから，コスト関数の処理コストは，イベント数に基づいて計算し，C_{eval} は処理する断片をパースする上で発生するイベント数，$C_{integration}$ は問合せ結果となるデータのパースで発生するイベント数に基づいて計算した．

4.2　実 験 結 果

4.2.1　断片の併合

併合の結果決定した三つの断片は表 6 のようになった. 本手法では，コスト関数として処理コストに関わる要因（実験ではイベント数）と発行頻度が考慮されている．処理コストの面から併合が起こりやすい組合せは二つ考えられる．一つは併合によって問合せで処理する断片の数が一つになり，統合が不必要になるような組合せである．この場合，併合によって統合の必要が無くなるため，統合にかかる処理コスト分だけコスト関数の値が減少するため併合が起こりやすくなる．表 6 では，断片の候補 f_1 と f_4，および f_2 と f_3，はこの理由によって併合が行われたと考えられる．

二つ目は，処理コストの少ない組合せである．この場合，併合によるコスト関数の値の増加が少ないため，併合が起こりやすくなる．表 6 では，比較的処理コストの少ない f_2, f_3, f_5, f_6, f_7 がこの理由によって併合されたと考えられる．

しかし，併合によって統合の必要が無くなるにもかかわらず，断片 f_1 と f_6 の併合は行われなかった．これは，発行頻度による影響が強いと考えられる．問合せ q_1 と q_6 では発行頻度の差が大きいため，併合によって q_1 にかかる処理時間のコスト増加の比重が大きくなるため併合が行われなかったと思われる．

4.2.2　問合せ処理時間

次に，問合せセットの 7 つの問合せにかかる問合せ処理時間を分割の前後で比較した．実験結果はデータサイズごとに 図 7, 8 のようになった．ただし，処理時間を分割前後の比で表現しているため，グラフの高さが 1 以下のとき，分割によって処理が高速化したことを示している．なお，問合せ処理時間を問合せの解析，断片の評価，結果の統合のそれぞれにかかった処理時間によって色分けしている．

結果，ほとんどの場合で問合せ処理の高速化がみられた．用いた XML 文書のサイズが大きい方（10MB）が処理が高速化しているのは，パースやファイルの読み込みにかかるオーバヘッド分の処理時間の影響が少なくなることに起因すると考えられる．問合せ処理がどの程度高速化されるかは分割後の断片のデータサイズに依存する．これは，問合せに必要な部分の局所性に関連して変化する．今回の実験では，元の XML 文書に対して f_A が約 17 %，f_B が約 6 %のサイズとなり，元の XML 文書のサイズが異なってもほぼ一定であった．このことから，元の XML 文書のサイズを大きくした場合，およそこれらの割合の程度まで高速化されると考えられる．

問合せの解析にかかる処理時間は元の XML 文書のサイズに関係無くほぼ一定であった．これは，使用したデータが同じスキーマに基づいて生成されているため，データサイズが増加し

(注2)：http://xml.apache.org/xerces2-j/

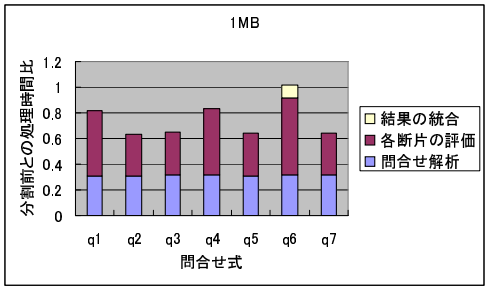

図 7　分割前後の処理時間比（1 MB）

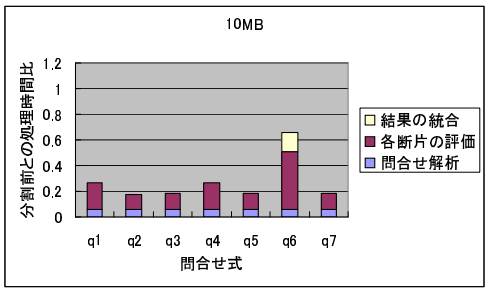

図 8　分割前後の処理時間比（10 MB）

ても構造概要のサイズがほぼ一定なためだと考えられる．このことから，構造概要を使用した本手法は，スキーマが同じデータであれば，XML 文書のサイズが大きいほど問合せの解析にかかる時間の割合が少なくなるため有効であると考えられる．

結果の統合にかかる処理時間は 3.2.2 節で述べたように統合の処理方法によって異なる．今回の実験では，問合せ q_6 において統合が必要となった．q_6 の場合，問合せ結果の規模が小さいことや，統合に用いた SAX での処理の影響などの要因からそれほど統合の時間はかかっていないが，問合せ結果の大きさやそれに伴うファイルへの一時的な書き込みの必要性の有無など，実際にはより処理時間がかかるケースも考えられる．それらを判断し，併合時の処理時間コストを算出することで，処理コストが高い統合を併合によって回避できる．

5　おわりに

本論文では，アクセスの偏りを考慮して XML 文書の分割し，問合せ処理を効率化する手法を提案した．問合せセットの問合せ式毎に，評価に必要な部分を断片の候補とし，それらを処理コストと発行頻度を考慮して併合した．これにより，問合せ処理を効率化するよう，XML 文書を任意の数へと分割することができる．さらに，分割後の問合せを，構造概要によって解析，変換することによって可能にした．また，提案手法の実装とその実験結果により，問合せセットとその発行頻度に基づいて問合せ処理を高速化できることを示した．具体的には，アクセスされる部分の局所性や，問合せの発行頻度の偏りに基づいて分割を行うことで，アクセスの偏りに応じて問合せ処理を効率化できた．また，XML 文書のサイズと処理時間の変化，構造概要の特徴から，スキーマが同じ XML 文書では，データサイズが大きいほど提案手法が有効であるとわかった．

今後の課題として，述語を含む問合せやデータ更新への対応が挙げられる，また，組合せ最適化のアルゴリズムとして欲張り法以外の手法を用いることや，重複部分の複製を行う場合での処理についても検討したい．

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 16016243），日本学術振興会科学研究費補助金（課題番号 15200010，15700097）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] 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 *ICDE2002*, pp. 141–152, 2002.

[2] Jan-Marco Bremer and Michael Gertz. An efficient XML node identification and indexing scheme. In *CSE-2003-04*, January 2003.

[3] Jan-Marco Bremer and Michael Gertz. On Distributing XML Repositories. In *In 6th International Workshop on the Web and Databases WebDB2003*, pp. 73–78, 2003.

[4] Open Directory Project. `http://dmoz.org/`.

[5] Roy Goldman and Jennifer Widom. DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases. In *VLDB'97, Proceedings of 23rd International Conference on Very Large Data Bases*, pp. 436–445. Morgan Kaufmann, 1997.

[6] Amélie Marian and Jérôme Siméon. Projecting XML Documents. In *VLDB 2003: Proceedings of 29th International Conference on Very Large Data Bases, September 9–12, 2003, Berlin, Germany*, pp. 213–224, Los Altos, CA 94022, USA, 2003. Morgan Kaufmann Publishers.

[7] Online Computer Libraly Center. Introduction to the Dewey Decimal Classification. `http://www.oclc.org/dewey/versions/abridgededition14/intro.pdf`.

[8] Philip Bohannon and Juliana Freire and Jayant Haritsa and Maya Ramanath and Prasan Roy and Jérôme Siméon. From XML Schema to Relations: A Cost-Based Approach to XML Storage. In *ICDE '02: Proceedings of the 18th International Conference on Data Engineering (ICDE'02)*, p. 64. IEEE Computer Society, 2002.

[9] Simple API for XML. `http://www.saxproject.org/`.

[10] World Wide Web Consortium. Document Object Model (DOM). `http://www.w3.org/DOM/`.

[11] World Wide Web Consortium. Extensible Markup Language (XML) 1.0 (Third Edition). `http://www.w3.org/TR/2004/REC-xml-20040204/`, February 2004. W3C Recommendation 04 February 2004.

[12] World Wide Web Consortium. XQuery 1.0: An XML Query Language. `http://www.w3.org/TR/xquery/`, October 2004. W3C Working Draft 29 October 2004.

[13] An XML Benchmark Project (XMark). `http://monetdb.cwi.nl/xml/index.html`, 2003.

[14] 天笠俊之, 植村俊亮. 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. 日本データベース学会 Letters, pp. 33–36, September 2004. Vol.3, No.2.

[15] 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. 研究報告 - データベースシステム, 第 2004-71 巻, July 2004.

[16] 中尾伸章, 天笠俊之, 植村俊亮. 部分圧縮を用いた大規模 XML データ処理方式の提案. 電子情報通信学会技術研究報告, 第 104-102 巻, pp. 1–6, june 2004. DE2004-1.

[17] 中島哲, 小田切淳一, 井谷宣子, 吉田茂. XML 高速処理技術 SPlit-DOM の機能拡張と Web アプリケーションへの適用評価. 電子情報通信学会第 15 回データ工学ワークショップ (DEWS2004), March 2004. I-5-5.