論文 / 著書情報
Article / Book Information

論題(和文)	部分文書処理コストを考慮した XML データの分割配置とアクセス手法
Title(English)	A Distributed XML Data Placement and Access Method Considering Subtree Processing Cost
著者(和文)	吉野悠二, 梁文新, 横田治夫
Authors(English)	Yuuji YOSHINO, Wenxin LIANG, Haruo YOKOTA
掲載誌(和文)	データ工学ワークショップ2008 論文集
Citation(English)	Proceedings of Data Engineering Workshop2008
Vol, no, pages	, ,
発行日 / Pub. date	2008, 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) 2008 Institute of Electronics, Information and Communication Engineers.

部分文書処理コストを考慮したXMLデータの分割配置とアクセス手法

吉野　悠二[†]　　梁　　文新[††,†††]　　横田　治夫[†††,††††]

† 東京工業大学 工学部 情報工学科
†† (独) 科学技術振興機構
††† 東京工業大学 学術国際情報センター
†††† 東京工業大学 大学院 情報理工学研究科 計算工学専攻
E-mail: {†yoshino, ††,†††wxliang}@de.cs.titech.ac.jp, †††,††††yokota@cs.titech.ac.jp

あらまし　近年増加している大規模なXMLデータを格納する手段の一つとして，処理の効率化や管理機能の観点から，複数の関係データベースシステムに分散させる方法が考えられる．このとき，処理効率向上のために分散された処理コストが均衡するような分割配置方法が重要となる．XMLデータの処理には，検索の走査コストと，木構造へ再構築するコストを考慮する必要がある．本稿では，大規模なXMLに特徴的な構造を前提に，各部分データが同等の意味単位となるように分割した上で，検索対象となる要素の文字列に着目して分割したデータのクラスタリングを行うことで処理コストを均衡化させる手法を提案する．各部分データの処理コストにおいては，部分データのサイズとノード数等を考慮する．また，分散配置されたクラスタの位置情報取得のためのインデックス構造に関しても提案を行う．Wikipediaの XMLデータを，複数のPostgreSQLサーバに提案手法によって分割配置した実験によって評価を行う．
キーワード　XML, 半構造化文書, 処理コスト

A Distributed XML Data Placement and Access Method Considering Subtree Processing Cost

Yuuji YOSHINO[†], Wenxin LIANG[††,†††], and Haruo YOKOTA[†††,††††]

† Department of Computer Science, Faculty of Engineering, Tokyo Institute of Technology
†† Japan Science and Technology Agency
††† Global Scientific Information and Computing Center, Tokyo Institute of Technology
†††† Department of Computer Science, Graduate School of
Information Science and Engineering, Tokyo Institute of Technology
E-mail: {†yoshino, ††,†††wxliang}@de.cs.titech.ac.jp, †††,††††yokota@cs.titech.ac.jp

Abstract　Recently, we have to handle very large XML documents. To provide efficient retrieval and management functions for the large-scale XML documents, it is effective to store those XML documents into distributed RDB systems. For the approach, it is important to realize a distributed XML data placement for balancing processing costs of traversal and subtree reconstruction. In this paper, we propose a method to balance the costs of processing distributed XML data. First, we fragment an XML document into subtrees representing the same or similar meaningful unit. Next, we cluster the fragmented subtrees based on the strings of specific nodes and calculate the processing cost of each cluster using the size and the number of nodes in each subtree. Then, we allocate the clusters in distributed RDB systems to make the processing cost equal. We also propose an index structure to derive the information about data location. We evaluate the effectiveness of proposed method by experiments storing XML data of the Wikipedia into multiple PostgreSQL servers.
Key words　XML, semi-structured data, processing cost

1. はじめに

近年，XML(Extensible Markup Language) の普及にともない，そのサイズが増大している．なかでも，Web 上のオンライン百科事典である Wikipedia [1] や，コンピュータ科学分野の文献を保持する Digital Bibliography & Library Project(DBLP) [2]，タンパク質データベースである Universal Protein Resource(Uniprot) [3] 等，構造に特徴をもつ大規模な XML データが登場してきている．

このような XML の大規模化により，検索や管理といった XML データ処理のコストも増大している．既存の関係データベースの検索や管理の機能を利用することは，そのような大規模 XML にとっても有効である．特に，大規模化した XML に対しては，複数の関係データベースに分散してデータを配置させることで，処理性能を向上させることが可能となる．

そのような分散環境において処理効率を向上させるためには，分散された各 DBMS 間の処理コストが均衡化するように，XML を分割して配置する方法が重要となる．特に，本稿では検索の走査コストと，分割されたデータから要求される木構造に再構築するコストに着目して，処理コストの均衡化を目指す．

一般には XML の構造は自由度が高いが，2. 章で述べるような大規模 XML 文書では，同じ意味単位となる XML の部分木が多数存在する．本稿では，そのような XML の構造を前提に，XML を分割配置する手法について提案を行う．まず，各 XML 部分データが同等の意味単位となるように分割した上で，検索対象となる要素の文字列に着目して分割したデータのクラスタリングを行うことで処理コストを均衡化させる．また，分散配置されたクラスタの位置情報取得のためのインデックス構造の提案を行う．さらに，提案手法の有効性確認のために，Wikipedia の XML データを，複数の PostgreSQL サーバに提案手法によって分割配置した実験を行う．

これまでにも，XML データを分割格納する様々な提案がなされているが，我々の前提とする環境には合わない．夏目等は XML データを分散格納する手法について提案している [4]．データの断片がそれぞれ意味を持つよう分割している点で本研究と類似しているが，配置先の決定に処理コストを考慮していない点や，DBMS を使用せず検索だけを行っている点で本研究と異なる．Brember 等は，Repository Guide と呼ばれる構造概要を，XPath 式によって定義される水平・垂直分割を行う手法を提案している [5]．この手法では同じ Path 式を持つノードを区別できないため，本研究で対象とするような大規模な XML 文書に適用すると，大量のノードが一つの断片に入ってしまい，分散先の容量に大きな差ができてしまう．Yaxin Yu 等は PSPIB, NSNRR という二つの分割手法を提案している [6]．PSPIB は枝の分岐により分割する手法であり，NSNRR は文書順にノードを Round-Robin で配置していくというものである．PSPIB は分割数を任意に指定できないという点で本研究と異なり，NSNRR はノード単位で分割され各格納先に散らばるため，木構造を再構築するコストが高くなってしまう．Nan Tang 等が提案する WIN [7] は，ワークロードから INode とよばれるノードを選択し，これを根とする部分木に分割する手法である．また，その配置先を求めるアルゴリズムも提案している．しかしこの手法では，同じ親を持つ部分木は同じ配置先に格納するという手法のため，本研究の対象である大規模 XML 文書では格納量が偏った配置になってしまう．

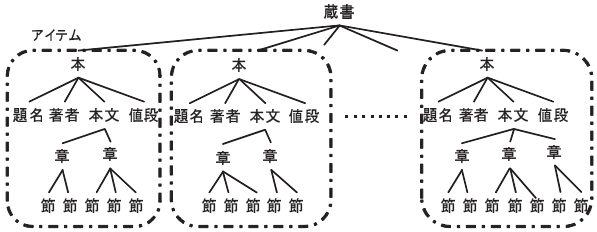

図 1　大規模 XML 文書の例

以下，本稿の以後の構成を示す．まず，本研究の前提とする大規模 XML 文書の特徴を 2. 章で述べる．次に，3. 章で部分木の処理コストを用いた分散配置手法を提案する．4. 章では，分散配置されたクラスタの位置情報取得のためのインデックス構造の提案を行う．5. 章では提案手法の評価実験について述べ，最後に 6. 章でまとめと今後の課題を述べる．

2. 大規模 XML 文書の特徴

本章では，本研究で前提とする大規模 XML 文書について，一般の XML 文書と比較しながら，構造的な特徴と利用方法の特徴を順に示し，それらの特徴が分散配置に与える影響について述べる．

2.1 構造的な特徴

本研究で対象とする大規模 XML 文書は，図 1 で表されるような，ある特定ノード (図 1 では “蔵書”) の子ノードの数が非常に多い XML 文書と定義する．このような大規模 XML 文書の例として，Wikipedia [1] や，DBLP [2]，Uniprot [3] 等がある．それぞれ特定ノードの子の数は，170 万，70 万，20 万程になっている．

また，その子ノードを根とした部分木は，それぞれ互いに似通った意味を持っている．この部分木一つ一つを以後 “アイテム” と呼ぶことにする．

大規模 XML 文書：　子ノードを非常に多く(注1) 持つノードが存在する XML 文書

アイテム：　大規模 XML 文書において，兄弟の数が非常に多いノードを根とし，その子孫を全て含む部分木

アイテムの例としては，Wikipedia に対する一用語の説明，DBLP に対する一つの文献情報，Uniprot に対する一タンパク質情報等が挙げられる．

2.2 利用方法の特徴

これらの大規模 XML 文書に対する利用者からの問い合わせは，一般的に XML を操作する言語である XPath [8] や

(注1)：ノード数を特定することができないため，ここでは “非常に多く” という曖昧な表現の定義となっているが，例として挙げた Wikipedia, DBLP, Uniprot の例をイメージされたい．

XQuery [9] によるものではなく，キーワード検索が主であるという特徴がある．これはデータベースに対する利用者の種別が多岐にわたるため，格納されているデータ構造を利用者が熟知していなくとも，簡単に欲しい情報を得ることができるようにするためである．

そして問い合わせに対する返値にも特徴がある．通常の問い合わせでの返値は，問合せにより指定されたノード群であるが，大規模 XML 文書での返値は，アイテムが最小単位になることが多い．このように，ノードを一つ一つとして見ていくのではなく，アイテムを一塊りとして扱うことが多いということである．

2.3 分散配置に与える影響

配置の決定において重要なのは，問い合わせに対する処理コストの削減である．本研究では処理コストとして検索の走査コストと木構造の再構築コストを考える．前述したような構造的な特徴と利用方法の特徴を大規模 XML 文書が持つため，その分割とデータベースへの分散配置は，これらの特徴を考慮することが重要である．

まず検索の走査であるが，キーワード検索が主なため，検索対象となる文字列によってアイテムを分類しておくことにより，その走査対象を絞ることが考えられる．走査対象を絞ることにより，走査の高速化が望まれる．

次に，返値がアイテムを単位とすることから，木構造の再構築をアイテム単位で生成することを考える．同一アイテム内のノード全てを同一の処理ノード (PE:Processing Element) に保持することにより，再構築時に他の格納先 PE と通信を行わずに済むようになる．従って，ノード毎または Path 式を基にした配置先の決定ではなく，アイテム毎に適した配置先を定めていく．

3. XML 文書の分散配置

提案手法は，まず検索対象文字列を用いてアイテムをクラスタリングし，検索の走査対象を絞る．次にクラスタ毎に処理コストを算出し，格納先 PE を決定するという手法である．

以後この章では，我々が提案する手法について，分割，クラスタリング，コスト計算，配置という順を追って詳細を述べる．

3.1 XML 文書の分割

最初に大規模 XML 文書をアイテム単位に分割する．子ノードへの分岐枝を多く持つノードを指定し，そのノード以下で切断する．同等の意味単位となるような部分木を見つける手法は，最小共通祖先 (LCA) 要素を用いる手法 [10], [11] や，構造の類似性を見る方法 [12] 等があるが，本研究では分割後の分散配置に焦点を当てているため，その分割の手法は問わない．

これにより，格納対象となる文書は多数のアイテムと，ルート要素を含むひとつの部分木に分割される．例として DBLP であれば，論文一部の情報をもつアイテムが多数と，それらの統括情報であり元の根要素を含む部分木とに分かれる．

3.2 部分木のクラスタリング

まず，各アイテムからそのアイテムを代表するようなノードで，かつ検索対象となるようなノードを指定する．アイテムを代表するノードとは，アイテムに含まれるノード群の中で，そのノード以下のテキストが意味的に高い重要度を持つと思われるノードとする．例えば “日付，題名，ページ数” という 3 種のノードが存在した場合，“題名” ノードが選択される．検索対象となるノードとは，利用者の検索において与えられるキーワードが，より多く含まれるノードである．キーワード検索において意味をなさない，“識別子” ノードや “文字の装飾” ノードなどを除外する．

表 1　着目ノード内文字列とクラスタ先対応例

	着目ノード内文字列	頭文字集合	クラスタ先
アイテム 1	**a**mazon **r**iver	{a,r}	$Cluster_{ar}$
アイテム 2	**d**ata **m**ining	{d,m}	$Cluster_{dm}$
アイテム 3	**r**ing **a**rch **r**idge	{r,a}	$Cluster_{ar}$
アイテム 4	**m**ulti **d**imensional **a**rea	{m,d,a}	$Cluster_{adm}$

これにより得たノード内の文字列を用い，アイテム群をいくつかのクラスタに分類する．ここでは，着目する文字列に含まれる単語の頭文字の，重複を取り除いた集合が等しいものを同じクラスタとする．表 1 の例では，アイテム 1“amazon river” の頭文字が ‘a’ と ‘r’，アイテム 3 “ring arch ridge” の頭文字が ‘r’, ‘a’, ‘r’ であり，その重複を取り除いた集合がそれぞれ {a, r} で等しいため，アイテム 1 とアイテム 3 は同じクラスタとなる．

しかし，現れる文字種全てをそのまま考慮すると，文字種が n_a 種のとき，クラスタの種類数が 2^{n_a} 種という膨大な数になってしまう．そこで，複数の文字種を一つの文字に割り当てることにより，クラスタの種類数を削減することを考える．まず，全ての文字種を { アルファベット 26 種 (大文字小文字を区別しない)，数字，その他 } の 28 種に一旦まとめる．その後，この 28 種を出現頻度昇順に並べ，最初の m_1 種を文字 $\mathbb{A}$ に，次の m_2 種を文字 $\mathbb{B}$ に，と順にそれぞれ $\mathbb{A}, \mathbb{B}, \mathbb{C}, \ldots$ という M 種の文字に集約することで，クラスタ種類総数を 2^M 種に減らすことができる．

集約数を定める際に，分散 PE 数との兼ね合いが必要である．集約数が少ないと 3.4 節で述べる配置先決定の自由度が低く，各 PE の処理コストが十分均衡化できない．また，集約数が多いとクラスタの総数が膨大なため，4. 章で述べる配置インデックスのサイズが大きくなってしまう．よって集約数として適当な値を定めることとする．その定め方は今後の課題としている．

3.3 処理コストの計算

本研究では，各クラスタの処理コストを，そこに含まれるアイテムの処理コストの総和として求める．また，アイテムの処理コストは，関係データベースに木構造を分断されて配置された XML 文書の走査と再構築は，文書サイズが等しいときでもノード数が異なるとき，その処理コストに差が出ると思われるため，その処理コストを求める変数として，アイテムのサイズとノード数を考慮する．

ここで，処理コストへのサイズとノード数の影響の度合いと，ノード数を考慮に入れることの妥当性を示す事前実験を行った．

まず図 2 に，サイズとノード数をそれぞれ変化させた XML

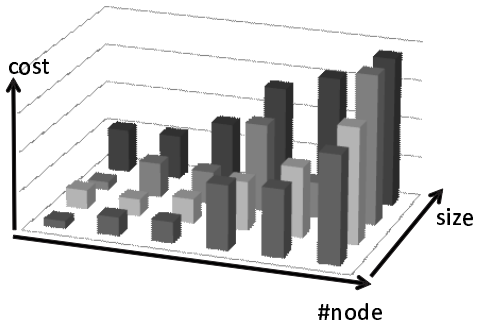

図 2　サイズとノード数を変化させたときの処理コスト変化

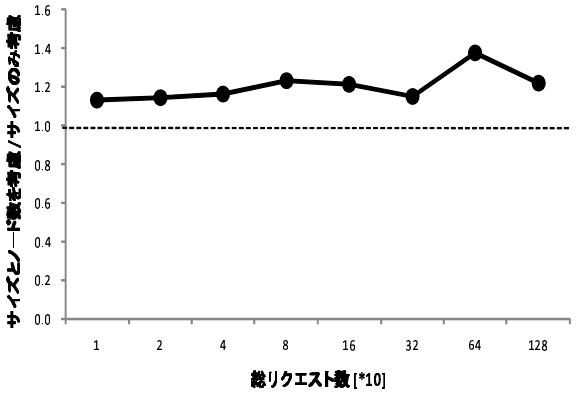

図 3　コスト計算式の比較

文書に対しての処理コストを測定した実験結果を示す．この実験結果によると，処理コストの増加は，文書サイズの増加に対しての増加オーダより，ノード数の増加に対しての増加オーダのほうが大きいという結果になった．

次にこの実験の結果を基に，各アイテムの処理コスト $ProcCost_{item}$ を，サイズ s のみを考慮したもの (式 1) と，サイズ s とノード数 n を考慮したもの (式 2) を考えた．

$$ProcCost_{item} = \log s \tag{1}$$

$$ProcCost_{item} = (\log s + 1) \cdot \frac{n^2}{\log n + 1} \tag{2}$$

但し，サイズやノード数が極端に大きい少数のアイテムによる影響を削減するため，定義域の上界を設定する．また，上の計算式で log を用いるため，その値が負にならないよう定義域の下界も設定し，その下界の値で正規化するという作業を行う．この作業により，処理コストの値が必ず正となる．

実際に 5 台の実機上に式 1 と式 2 の 2 種類のコスト計算式を用いた提案手法で XML 文書を分割配置させてスループットを測定した．その他の環境は 5. 章での実験と同じである．図 3 にこの事前実験の結果を示す．この図は式 1 を用いて分割配置させたものの値で正規化したもので，総リクエスト数が少ない場合から多い場合まで，一貫して 20%前後の性能差が見られた．

以上の 2 つの事前実験により，本研究では各クラスタの処理コスト $ProcCost_{cluster}$ をアイテムサイズ s，ノード数 n を用いた次の式で見積もる．

表 2　識別子とクラスタの対応 (文字種が 4 種の場合)

識別子	クラスタ	識別子	クラスタ	識別子	クラスタ
0	$\emptyset$	6	$\mathbb{AC}$	12	$\mathbb{BD}$
1	$\mathbb{A}$	7	$\mathbb{ACD}$	13	$\mathbb{C}$
2	$\mathbb{AB}$	8	$\mathbb{AD}$	14	$\mathbb{CD}$
3	$\mathbb{ABC}$	9	$\mathbb{B}$	15	$\mathbb{D}$
4	$\mathbb{ABCD}$	10	$\mathbb{BC}$		
5	$\mathbb{ABD}$	11	$\mathbb{BCD}$		

ID	Cluster	Cost [%]		ID	Cluster	Cost [%]	
0	$Cluster_{\Phi}$	0	PE 1	8	$Cluster_{AD}$	7	PE 3
1	$Cluster_{A}$	8		9	$Cluster_{B}$	9	
2	$Cluster_{AB}$	6		10	$Cluster_{BC}$	5	
3	$Cluster_{ABC}$	3		11	$Cluster_{BCD}$	2	PE 4
4	$Cluster_{ABCD}$	2		12	$Cluster_{BD}$	8	
5	$Cluster_{ABD}$	5	PE 2	13	$Cluster_{C}$	10	
6	$Cluster_{AC}$	8		14	$Cluster_{CD}$	9	PE 5
7	$Cluster_{ACD}$	5		15	$Cluster_{D}$	13	

図 4　クラスタの分散例 (文字種が 4，格納先 PE 数が 5 の場合)

$$ProcCost_{cluster} = \sum_{item \in cluster} ProcCost_{item}$$

$$ProcCost_{item} = (\log s + 1) \cdot \frac{n^2}{\log n + 1}$$

3.4　配置先の決定

各クラスタの配置先 PE の決定には，3.2 節で着目した頭文字に順序を付け，それをキー値としてクラスタを値域分散させるという手法をとる．

まず，着目文字列に含まれる単語の頭文字に使われる文字種を全てを抜き出す．次に，3.2 節で述べた手法で文字種を集約する．集約された文字種毎の出現頻度を調べ，出現頻度の昇順を辞書順として，各クラスタに識別子を付与する．例として，集約された文字種が $\{\mathbb{A}, \mathbb{B}, \mathbb{C}, \mathbb{D}\}$ の 4 種で，その出現頻度昇順が $\mathbb{A}, \mathbb{B}, \mathbb{C}, \mathbb{D}$ であるとき，各クラスタに付される識別子は表 2 のようになる．この識別子をキー値として，クラスタの処理コストの和が格納先 PE 間で均等になるように値域分散させる (図 4)．以後，アイテムの挿入があったときは，そのアイテムが含まれるべきクラスタを保持している格納先 PE へ追加する(注2)．

4.　クラスタ配置インデックス構造

クライアントからの入力であるキーワードから，対象となるアイテムの格納先クラスタ候補を求めるために，図 5 に示すような構造の，クラスタ配置インデックスを付加する．

キーワードの先頭文字種を辞書順に並べてクラスタを作成したことから，頭文字種からクラスタへの対応を付ける表は，辞書順に現れる文字種の再帰的な構造を考慮した，表 3 に示す定義から作成できる．これにより作成された表をクラスタ配置インデックスとして用いる．表 4 に文字種が 4 種の場合の対応表例を示す．

(注2)：ルート要素を含む部分木は，全ての PE へ複製を配置する．

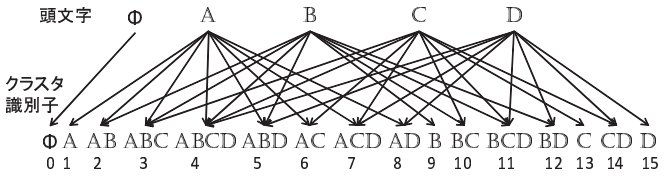

図 5　クラスタ配置インデックス構造

表 3　頭文字，クラスタ対応定義

頭文字，クラスタ識別子対応表 $E_N[\,]$　(N：文字種総数)
(但し，出現頻度昇順が i 番目の文字種を含む
クラスタ識別子群が，$E[i]$ に入るものとする)

$\mathrm{Length}(E_1) = 2$

$E_1[0] = \{0\}$

$E_1[1] = \{1\}$

$$E_N[i] = \begin{cases} \{0\} & \text{if } i = 0 \\ \bigcup_{k=1}^{2^{N-1}} \{k\} & \text{if } i = 1 \\ \{n+1, n+2^{N-1} | \forall n \in E_{N-1}[i-1]\} & \text{otherwise.} \end{cases}$$

表 4　頭文字，クラスタ識別子対応表例 (文字種が 4 種の場合)

頭文字	クラスタ識別子
Ø	0
𝔸	8, 7, 6, 5, 4, 3, 2, 1
𝔹	12, 11, 10, 9, 5, 4, 3, 2
ℂ	14, 13, 11, 10, 7, 6, 4, 3
𝔻	15, 14, 12, 11, 8, 7, 5, 4

アルゴリズム getClusterIDs
入力: 探索キーワード群 $w[\,]$
出力: クラスタ識別子群 C

```
begin
  C ← Union
  i ← 1
  while i ≦ length(w) do begin
    s ← Initial(w[i])
    C ← C ∩ E_N[Freq(s)]
    i ← i + 1
  end;
  return C
end;
```

(Freq(s): 文字 s の出現頻度昇順序を返す)

図 6　アクセス先クラスタ識別子の取得アルゴリズム

キーワードからアクセス先クラスタは，そのキーワード毎に頭文字から対応クラスタ識別子を取得し，それらの積集合によって求められる (アルゴリズムを図 6 に示す)．表 4 の例で，“Magic Number” という二つの単語を両方含むアイテムを探す例を考える．ここで，M は 𝔹 に，N は 𝔻 に集約されているとする．まず，キーワードの頭文字から集約先の文字を求め，$\mathbb{B}, \mathbb{D}$ となる．表 4 から 𝔹 の行 {12,11,10,9,5,4,3,2} と 𝔻 の行 {15,14,12,11,8,7,5,4} を取り出す．次に，取り出された 2 つの集合の積集合を求め，得られた集合 {12,11,5,4} が “Magic Number” を含む可能性があるクラスタの識別子である．

この手法により取得したクラスタを保持する PE のみに走査を行うことで，問合せに対する結果を少ない処理 PE 数で得ることが可能となる．

表 5　実験システムの構成

#PE:	2,4,8 (Server),　4 (Client)
CPU:	AMD Athlon XP-M 1800+ (1.53 GHz)
Memory:	PC2100 DDR SDRAM 1 GB
Network:	1000BASE-T
HDD:	TOSHIBA MK3019GAX (30 GB, 2.5 inch)
Java VM:	J2SE 1.5
Server RDB:	PostgreSQL 8.2.5 (for Server nodes)
JDBC:	postgresql-8.2-507.jdbc3 (for Server nodes)

表 6　SUCXENT++のスキーマ

テーブル名	属性名
Document	(DocId, Name)
Path	(PathId, CPathId, PathExp, Length)
PathValue	(DocId, PathId, LeafOrder, BranchOrder, BranchOrderSum, LeafValue)
DocumentRValue	(DocId, Level, RValue)

5. 実　　験

複数の実機上に関係データベースを実装し，実データを分散配置させた実験を行った．データの分散数を変化させた実験と，格納するデータのサイズを変化させた実験を行い，それぞれのスループットを測定した．

また，それぞれの実験において提案手法の有用性を示すために，NSNNR [6] と同様に XML データを文書順に配置する手法を実装，比較した．ただし NSNNR ではノード毎に Round-Robin で PE へ配置していくが，本稿での分割配置は，分割数を N とすると，格納する XML 文書をサイズが均等になるように文書順先頭から $1/N$ を PE_1 に，次の $1/N$ を PE_2 に，… ，最後の $1/N$ を PE_N に格納する手法として実装した．比較手法で分割配置されたデータへの問い合わせは，全ての PE に問い合わせを送信することにより処理する．

5.1 実験準備

実験に用いた実機の構成を表 5 に示す．

PE として用いる実機上に PostgreSQL を実装し，それぞれに格納する XML 文書を，SUCXENT++ [13] を用いて関係データベースの表形式へマッピングを行った．SUCXENT++ は，XML データのスキーマ構造を利用せず，XML データを木構造としてテーブルへマッピングを行う．また，XML データの葉ノードのみをタプルとして格納し，共通祖先ノードを得ることで問い合わせに答えるという格納方法であり，この手法の関係テーブルのスキーマは，表 6 となっている．格納する大規模 XML 文書として，スキーマが表 7 に示される English Wikipedia Abstract の一部を，実験内容により文書サイズを変化させて用いた．

次に，提案手法で用いるアイテムとクラスタリングに用いる文字列を定める．アイテムは doc 要素を根とする部分木とし，クラスタリングに用いる文字列には，title 要素内の文字列に

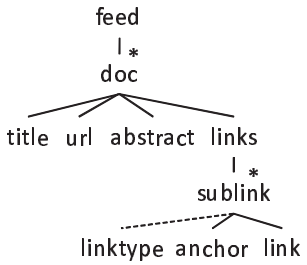

図 7　English Wikipedia Abstract のスキーマ

着目，使用した．また，頭文字として現れる文字について，{ アルファベット 26 種，数字，その他 } の 28 種を，出現頻度昇順の最初の 7 種を文字 $\mathbb{A}$ に，次の 6 種を文字 $\mathbb{B}$ に，と順に 7,6,5,4,3,2,1 種をそれぞれ $\{\mathbb{A},\mathbb{B},\mathbb{C},\mathbb{D},\mathbb{E},\mathbb{F},\mathbb{G}\}$ という 7 種に集約したものを用い，クラスタ種総数を $128(=2^7)$ とした．

処理の実装は全て Java 言語を用いてプログラムし，PostgreSQL に格納されたデータの処理には Java Database Connectivity(JDBC) を用いた．

5.2 実験内容

まず，4 台のクライアントノードで複数のスレッドを立て，全てのスレッドから同時にキーワード検索のリクエストを PE へ送信する．送信されるキーワードは，実際に配置されているデータの title 要素内に含まれる単語から無作為に 200 種選択したキーワード群から，80%の確率で 1 種類，20%の確率で 2 種類を選択した．

次に，リクエストを受け付けた PE は，クラスタ配置インデックスを参照し，与えられたキーワードが配置されている可能性を持つクラスタ識別子を得る．得た識別子から，そのクラスタを保持する PE を調べ，その PE へ走査を依頼する．走査依頼を受けた PE は，キーワードとクラスタ識別子から，PostgreSQL に SUCXENT++のテーブルスキーマで格納された XML データを操作する SQL 文を作成し，JDBC を用いて SQL 文を実行する．得られたタプルを SUCXENT++の再構築アルゴリズム [13] により木構造へ再構築し，依頼元 PE へ部分木を返す．全ての走査依頼先から部分木を受け取った PE は，部分木全てを一つの XML 文書としてまとめ，クライアントへ返す．

各クライアントはリクエストに対する PE からのレスポンスを受け取った後，一旦 PE との接続を切断し，次のリクエストを行うために接続を再び確立する．

これら一連の操作を 3 分間繰り返し，クライアントへ返されたリクエストの総数をスループットとして測定した．測定は時間をおいて 7 回行い，最大値と最小値を除去した後平均を取り，実験の結果の値とした．

表 7　データサイズ変化実験に用いた XML 文書

データ	サイズ [MB]	ノード数
DocumentS	29.8	710,303
DocumentM	59.5	1,417,508
DocumentL	119.3	2,837,618

5.3 PE 数変化

PE の数を 2, 4, 8 台と変化させ，それぞれについてスループットをクライアントスレッド数を変化させて測定した．この実験では，English Wikipedia Abstract の一部 (データサイズ: 29.8MB, ノード数:710,303) を格納 XML 文書として用いた．

実験結果を図 8 に示す．また，クライアント 1 台あたりのスレッド数が 128 のときの結果の比較を図 9 に示す．

この結果から，提案手法は比較手法に対し，クライアント数が多く負荷が高い状況において，より良いスループットを出すことが分かる．その差は PE 数が少ないときには小さいが，サーバ PE 数の増加に伴い差が広がっていく．PE 数 8，クライアント 1 台あたりのスレッド数 128 の結果において，提案手法は比較手法の約 120%となっているが，これは一リクエストに対して処理を行う PE 数の比較手法と提案手法の差から表れると思われる．走査 PE 数に関する考察は 5.5 節で行う．

5.4 データサイズ変化

格納する XML 文書を，English Wikipedia Abstract の一部を，サイズを変えて 3 種類 (表 7) 用意し，それぞれについてスループットをクライアントスレッド数を変化させて測定した．この実験では，PE 数を 8 とした．

実験結果を図 10 に示す．また，クライアント 1 台あたりのスレッド数が 128 のときの結果の比較を図 11 に示す．この結果から，スレッド数が多く負荷が高い状況では提案手法が 15〜20%程優位であるということが分かる．

提案手法においてデータサイズの増加は，その増加分のデータの単語出現頻度分布が著しく変化しなければ，分割配置手法，

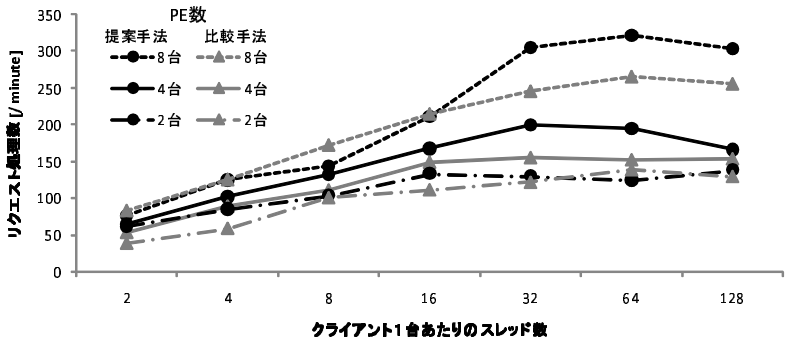

図 8　PE 数変化結果

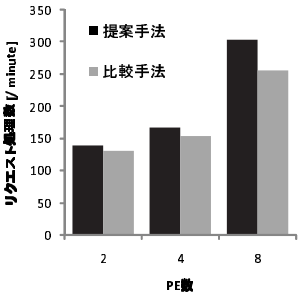

図 9　PE 数変化結果 (スレッド数 128)

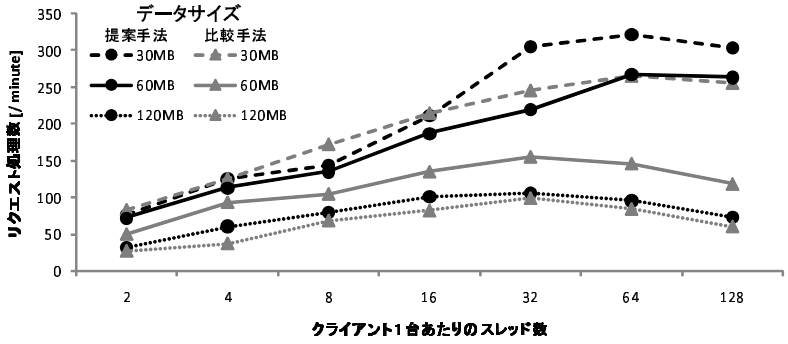

図 10　データサイズ変化結果

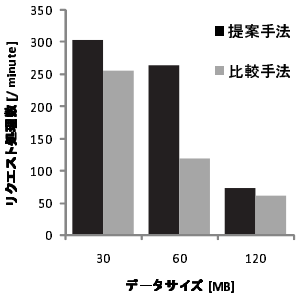

図 11　データサイズ変化結果 (スレッド数 128)

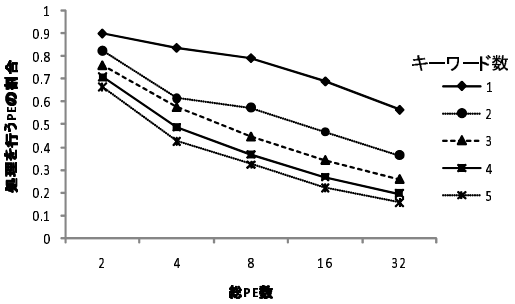

図 12　提案手法における，処理を行う PE 数の割合の変化

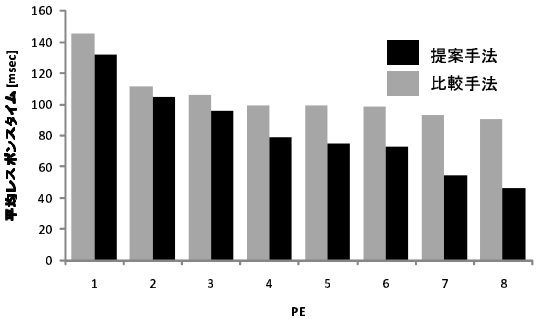

図 13　各 PE ごとの処理時間平均 (データサイズ 30MB)

表 8　処理時間の分散 (データサイズ 30MB)

	最大	最小	平均
提案手法	4.46	0.00	0.42
比較手法	7.08	0.10	0.63

配置インデックス構造とも変化しないため，走査を行う PE 数は変化しない．よって，5.3 節で示した比較手法との差がそのまま結果に現れると思われる．また，データサイズの増加により，PE 内でのキーワード走査対象量が増えるが，走査処理におけるコストの増加率は提案手法と比較手法に差が無いと思われる．したがってデータサイズを変化させたとき，提案手法と比較手法の差は PE 数を変化させたときの差にほぼ一致する．

5.5　提案手法の効果

実験において提案手法の目的である，走査 PE 数の減少，PE 内走査データ量の減少，PE 間の処理コストの均衡化がどのようであったかを調べた．

まず走査 PE 数の減少に関して，図 12 は PE 数とリクエストあたりのキーワード数に対して，問い合わせ処理を行う PE 数が，全体に占める数の割合の変化を示したものである．これによると，提案手法では分散数の増加に伴い処理を行う PE 数の全体に占める割合が下がっていく．また，キーワード数の増加に対してもその割合が下がっていく．今回の実験環境では $8\cdot(0.8\cdot0.79+0.2\cdot0.57)=6$ となり，全 8 台中 6 台で処理が行われるため，インデックス処理のコストを差し引き，20%程度の性能向上が見られたものと思われる．参考に総 PE 数が 32, キーワード数が 2 のとき，処理を行う PE 数は全体の 36%ほどになるため，この状況下では単純計算によると，提案手法は比較手法から 150%程度の性能向上が見られると推測できる．同様に総 PE 数が 32, キーワード数が 5 のとき，処理を行う PE 数は全体の 15%ほどになるため，500%程度の性能向上と推測できる．さらに PE 数が増加したとき，性能向上幅がより大きくなると考えられる．キーワード数の増加についても同様に性能が向上していくと考えられる．

次に PE 内走査データ量の減少について，リクエスト毎の，各 PE に対するレスポンスタイムの平均値を図 13 に示す．ただし，提案手法において走査が行われない PE のレスポンスタイムは 0[msec] として平均してある．

この結果から全ての PE について平均レスポンスタイムが減少していることがわかる．これは各 PE における問い合わせ処理が，比較手法は保持しているすべてのデータを走査するのに対し，提案手法は配置インデックスで限定された一部のクラスタのみを走査したためであると考えられる．

また，処理を行った PE に対してのキーワード毎のレスポンスタイムの分散を表 8 に示す．提案手法の分散の最小は処理を一つの PE で行ったときで 0 となっている．

分散の最大値，最小値，平均値ともすべて提案手法が良いという結果になっており，提案した手法の効果が実際に現れているといえる．

6.　まとめと今後の課題

本稿では，同等の意味単位となる部分木が多数存在するような大規模 XML 文書を，複数の関係データベースシステムに配

置する場合に，各データベースの処理コストが均衡化するような分割配置手法を提案した．提案手法では，文字列検索が多いという前提のもとに，分割した XML データを，走査の対象となる文字列によりクラスタリングし，クラスタ毎の処理コストをサイズとノード数により算出し，コスト値で分散配置している．また、分散配置されたクラスタの位置情報取得のため，クラスタの文字列情報に基づくインデックス構造の提案を行った．

処理コストを算出する際に，部分 XML データのサイズだけでなくノード数も考慮することで，性能の向上が見られるということを事前実験によって示した．また，実際に Wikipedia の XML データを分割し，複数の PostgreSQL サーバに提案手法によって配置した実験を行った結果，提案手法は比較手法に対して，分散数が多く，負荷が高いときに 20%程度の処理効率の向上が見られることを示した．加えて，分散 PE 数，リクエストキーワード数の増加にともない，処理効率がさらに向上するであろうことを示した．

今後の課題として，XML に用いられるタグの階層情報や検索キーワードの発現頻度を，処理コスト計算やクラスタリング手法に用いることを検討する．また，文字種を集約する数とその手法を，分散数に応じて適したものに決定する方法を加えて検討していく．最後に，分散 PEh 数とデータサイズをさらに増やした環境での実験も考える．

文　　献

[1] non-profit Wikipedia Foundation. Wikipedia. http://en.wikipedia.org/.

[2] Michael Ley. Digital Bibliography & Library Project. http://www.informatik.uni-trier.de/~ley/db/.

[3] Uniprot Consortium. Universal Protein Resource. http://www.ebi.uniprot.org/.

[4] 夏目亘, 横田治夫. 分散ディスクへの XML データの分割格納方法. In *Data Engeneering Workshop(DEWS)*, 2001.

[5] Jan-Marco Bremer and Michael Gertz. On distributing XML repositories. In *International Workshop on the Web and Databases(WebDB)*, 2003.

[6] Yaxin Yu, Guoren Wang, Ge Yu, Junan Hu, and Nan Tang. Data placement and query processing based on RPE parallelisms. In *27th Annual International Computer Software and Applications Conference(COMPSAC'03)*, 2003.

[7] Nan Tang, Guoren Wang, Jeffrey Xu Yu, Kam-Fai Wong, and Ge Yu. WIN: An efficient data placement strategy for parallel XML databases. In *11th International Conference on Parallel and Distributed Systems(ICPADS'05)*, 2005.

[8] World Wide Web Consortium. XML Path Language. http://www.w3.org/TR/xpath.

[9] World Wide Web Consortium. XML Query. http://www.w3.org/TR/Query.

[10] Yunyao Li, Cong Yu, and H.V.Jagadish. Schema-Free XQuery. In *International Conference on Very Large Data Bases(VLDB)*, 2004.

[11] Yu Xu and Yannis Papakonstantinou. Efficient keyword search for smallest LCAs in XML databases. In *SIGMOD*, 2005.

[12] Wenxin Liang, Xiangyong Ouyang, and Haruo Yokota. An XML subtree segmentation method based on syntactic segmentation rate. In *International Workshop on Advanced Storage Systems(ADSS)*, 2007.

[13] Sandeep Prakash, Sourav S. Bhowmick, and Sanjay Madria. Efficient recursive XML query processing in relational database systems. In *International Conference on Conceptual Modeling(ER)*, 2004.