部分木検索のための走査と結果構築コストを考慮した XML 分割配置と位置情報取得手法

A Data Placement and Access Method Considering Transversal and Result Construction Cost for XML Subtree Retrieval

吉野 悠二 † 梁 文新 ††
横田 治夫 †††

Yuuji YOSHINO Wenxin LIANG
Haruo YOKOTA

近年大規模化している XML データでは，検索対象が個別のノードではなく部分木毎であることが多い．また，複数の RDB に格納された XML データを効率よく走査・結果部分木構築を行うためには，各 DBMS 間での処理の均衡化が必要である．本稿では，大規模な XML データに特徴的な構造とキーワード検索を前提に，検索走査と結果部分木構築の処理コストが DBMS 間で均衡化するような XML 分割配置手法を提案する．XML の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．各クラスタは，含まれる部分データのサイズとノード数から算出したコストを考慮して各 RDB に配置する．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．

Recently, large-scale XML data are often retrieved in the unit of subtrees instead of nodes. To achieve efficient transversal and result construction of the XML data stored in distributed RDB systems, it is important to balance the processing costs among DBMSs. In this paper, we propose a method to balance the costs of processing distributed large 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 we allocate the clusters in distributed RDBs based on the costs calculated with the size and the number of nodes in each subtree. We also propose an index structure to achieve efficient data access. We perform experiments to evaluate the effectiveness of the proposed method using the XML data of English Wikipedia stored in multiple PostgreSQL servers.

1. はじめに

近年，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] といった，キーワードで検索するような大規模な XML データが増加してきている．このような XML データに対しての検索は，ノード一つ一つが結果として求められているのではなく，キーワードを含む特定の部分木が求められている．また，増加し続ける XML データ量に対して複数の RDB 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ている．

そのような XML データを分散させた環境において検索走査や結果部分木構築処理を効率よく行うためには，分散された各 DBMS 間で処理コストが均衡化するような XML を分割して配置する手法が必要となる．一般には XML の構造は自由度が高いが，前述したような大規模 XML 文書では構造的な特徴として，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 の構造とキーワード検索を前提に，複数の RDB 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．

以下に本稿の構成を示す．まず，２章では関連手法について，３章では本稿が前提とする大規模 XML 文書の特徴について述べる．次に，４章で我々が提案する XML 文書の分散配置手法を示す．５章ではクラスタの位置情報取得のためのインデックス構造を提案する．提案手法の評価実験について６章で述べ，最後に７章でまとめと今後の課題を述べる．

2. 関連手法

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．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 [3]．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稿と類似しているが，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稿と異なる．Brember 等は，Repository Guide と呼ばれる構造概要を，Path 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 [4]．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，本稿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．Yu 等は PSPIB, NSNRR という二つの分割手法を提案している [5]．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．Tang 等が提案した WIN[6] 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，本稿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．

3. 大規模 XML 文書の特徴

3.1 構造的な特徴

本稿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] 等がある．それぞれ特定ノードの子の数は，170 万，70 万程になっている．また，その子ノードを根

† 正会員　株式会社ジャステック　yoshinoy@jastec.co.jp
†† 正会員　(独) 科学技術振興機構　wxliang@de.cs.titech.ac.jp
††† 正会員　東京工業大学学術国際情報センター　yokota@cs.titech.ac.jp

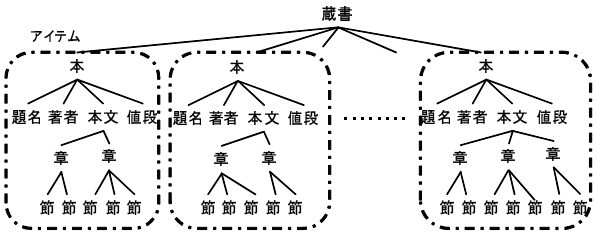

図 1 大規模 XML 文書の例
Fig. 1 An example of large XML Document

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[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が挙げられる．

3.2 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[7] など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．

3.3 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．本稿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で，その走査対象を絞ることを考える．走査量の削減により，走査の効率化が望まれる．次に，返値がアイテムを単位とすることから，検索結果部分木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．

4. XML 文書の分散配置

この節で我々が提案する XML データの分割配置手法を述べる．

4.1 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 [8, 9] や，シンタックスに基づく構造の類似性を見る方法 [10] 等があるが，本稿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．

[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 の例をイメージされたい．

表 1 着目ノード内文字列とクラスタ先対応例
Table 1 Example clusters corresponding to specific nodes

	着目ノード内文字列	頭文字集合	クラスタ先
Item1	**a**mazon **r**iver	{a,r}	$Cluster_{ar}$
Item2	**d**ata **m**ining	{d,m}	$Cluster_{dm}$
Item3	**r**ing **a**rch **r**idge	{r,a}	$Cluster_{ar}$
Item4	**m**agnetic **d**isk **s**ize	{m,d,s}	$Cluster_{dms}$

4.2 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．

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 4.4 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，5 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．

4.3 処理コストの計算

本稿では，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．

サイズとノード数を変化させた複数の人工的 XML 文書をそれぞれ PostgreSQL に格納し，それらに特定のキーワードを含む部

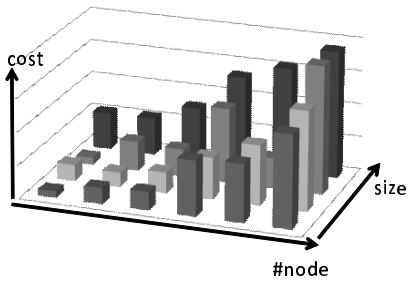

図 2　サイズとノード数を変化させたときの処理コスト
Fig. 2　Processing cost when changing item size and node number

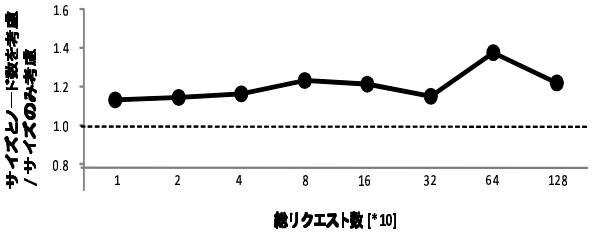

図 3　コスト計算式の比較
Fig. 3　Comparison of cost calculation formula

分木を構成して出力する同じ問合せを行い，そのレスポンスタイムを処理コストとして測定した。図 2 にこの事前実験の結果を示す．この結果によると，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 6 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，本稿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．但し $ProcCost_{item}$ は式 2 で定義されるものとする．

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

4.4　配置先の決定

各クラスタの配置先 PE の決定には，4.2 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，4.2 節で述べた手法で文字種を集約する．

表 2　識別子とクラスタの対応 (文字種が 4 種の場合)
Table 2　Example Clusters and their corresponding identifiers (case of 4 characters)

識別子	クラスタ	識別子	クラスタ	識別子	クラスタ
0	Ø	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	$\mathbb{BCD}$		

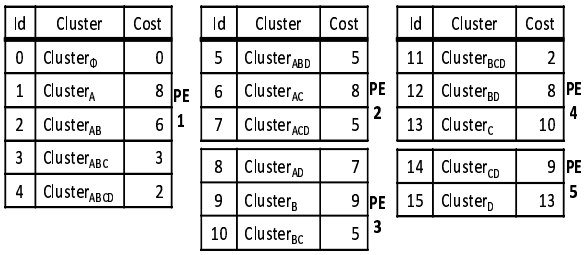

図 4　クラスタの分散 (文字種が 4，格納先 PE 数が 5 の場合)
Fig. 4　Distributed placement of clusters (case of 4 characters and 5 PEs)

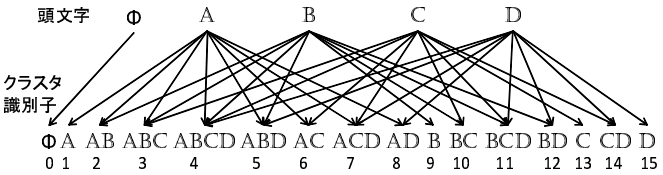

図 5　クラスタ配置インデックス構造
Fig. 5　Cluster placement index structure

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[2]．

5.　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を辞書順に現れる文字種の再帰的な構造を考慮して，表 3 のような表をクラスタ配置インデックスとして用いる．

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．表 3 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 $\mathbb{B}$ に，N は $\mathbb{D}$ 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 3 から $\mathbb{B}$ の行 {12,11,10,9,5,4,3,2} と $\mathbb{D}$ 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

[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．

表 3 頭文字とクラスタ識別子対応表 (文字種が 4 種の場合)
Table 3 Cluster IDs and their corresponding acronym (case of 4 characters)

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

表 4 実験システムの構成
Table 4 Experiment system environment

#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 5 English Wikipedia Abstract のスキーマ
Table 5 Schema of English Wikipedia Abstract

```
<!ELEMENT feed (doc)*>
<!ELEMENT doc (title, url, abstract, links)>
<!ELEMENT links (sublink)*>
<!ELEMENT sublink (linktype, anchor, link)>
```
※他は全て (#PCDATA). 但し linktype は ATTRIBUTE

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.

6. 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[5] 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.

6.1 実験準備

実験に用いた実機の構成を表 4 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++[11] 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る．格納する大規模 XML 文書として，スキーマが表 5 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.

6.2 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 [11] 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.

6.3 実験結果：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 6 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 7 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 6.5 節で行う.

6.4 実験結果：データサイズ変化

格納する XML 文書として，English Wikipedia Abstract の一部をサイズを変えて 3 種類 (表 6) 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 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，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.

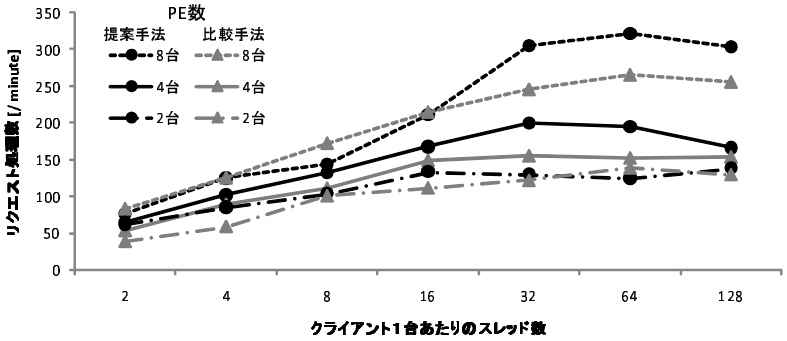

図 6　PE 数変化結果
Fig. 6　Result of changing PE number

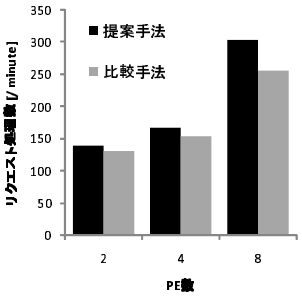

図 7　PE 数変化結果 (128 スレッド)
Fig. 7　Result of changing PE number (case of 128 threads)

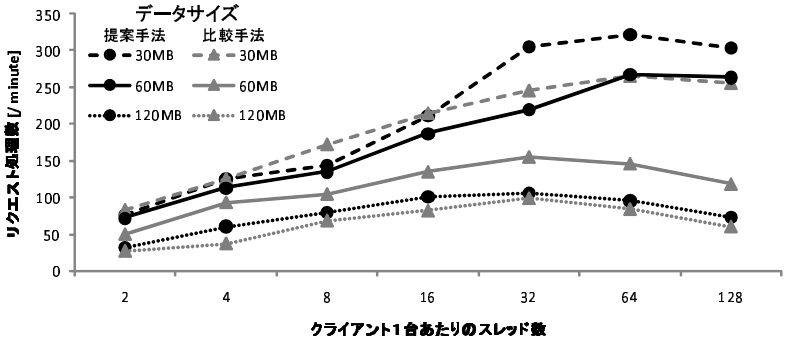

図 8　データサイズ変化結果
Fig. 8　Result of changing data size

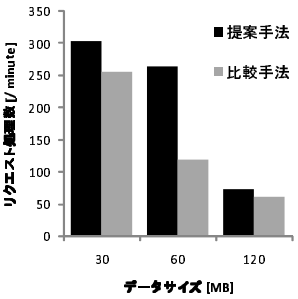

図 9　データサイズ変化結果 (128 スレッド)
Fig. 9　Result of changing data size (case of 128 threads)

表 6　データサイズ変化実験に用いた XML 文書
Table 6　Data size of XML documents used in the experiments

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

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，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，6.3 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．

6.5　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 数の削減，PE 内走査データ量の削減，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 数の削減に関して，図 10 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 $8 \times (0.8 \times 0.79 + 0.2 \times 0.57) = 6$ 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

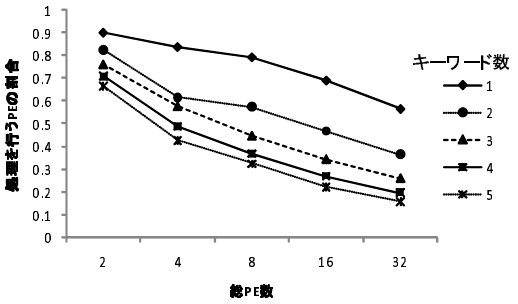

図 10　提案手法における，処理を行う PE 数の割合
Fig. 10　PE utilization ratio with the proposed method

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% 程になるため，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% 程になるため，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 内走査データ量の削減について，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 11 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

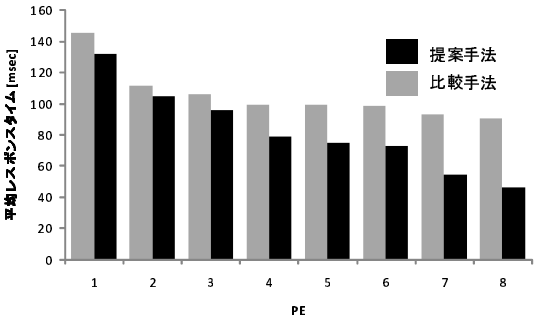

図 11　各 PE ごとの処理時間平均 (データサイズ 30MB)
Fig. 11　Average processing time for each PE (data size: 30MB)

表 7　処理時間の分散 (データサイズ 30MB)
Table 7　Distribution of processing time (data size: 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

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 7 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表れているといえる．

7.　まとめと今後の課題

本稿では，キーワード検索が行われる大規模 XML 文書で，同等の意味単位となる部分木が多数存在するもの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 DBMS 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，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走査コストとしてサイズから，結果部分木構築コストとしてノード数から算出し，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各 PE の実行時間の分散が小さく，提案手法の効果が表れていることを示した．実際の処理性能の向上度合いは，分散数が多く負荷が高いときに 20% 程度向上となっている．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可能性が高いことを示した．

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や，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検討していく．また，今回は対象 XML データは検索に使用される頻度が高いと考え挿入のコストを考慮に入れていないが，提案手法の適用による挿入コストの増加度合いを調べる実験を行っていく．最後に，分散 PE 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] 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”，Data Engeneering Workshop(DEWS)（2001）.

[4] Bremer, J. and Gertz, M.: “On distributing XML repositories”, International Workshop on the Web and Databases(WebDB) (2003).

[5] Yu, Y., Wang, G., Yu, G., Hu, J. and Tang, N.: “Data placement and query processing based on RPE parallelisms”, 27th Annual International Computer Software and Applications Conference(COMPSAC) (2003).

[6] Tang, N., Wang, G., Yu, J. X., Wong, K. and Yu, G.: “WIN: An efficient data placement strategy for parallel XML databases”, 11th International Conference on Parallel and Distributed Systems(ICPADS) (2005).

[7] World Wide Web Consortium: “XML Path Language”, http://www.w3.org/TR/xpath.

[8] Li, Y., Yu, C. and H.V.Jagadish: “Schema-Free XQuery”, International Conference on Very Large Data Bases(VLDB) (2004).

[9] Xu, Y. and Papakonstantinou, Y.: “Efficient keyword search for smallest LCAs in XML databases”, SIGMOD (2005).

[10] Liang, W., Ouyang, X. and Yokota, H.: “An XML subtree segmentation method based on syntactic segmentation rate”, International Workshop on Advanced Storage Systems(ADSS) (2007).

[11] Prakash, S., Bhowmick, S. S. and Madria, S.: “Efficient recursive XML query processing in relational database systems”, International Conference on Conceptual Modeling(ER) (2004).

吉野 悠二 Yuuji YOSHINO
株式会社ジャステック製造本部製造一部所属．2008 東京工業大学工学部情報工学科卒業．日本データベース学会正会員．

梁 文新 Wenxin LIANG
(独) 科学技術振興機構 CREST 研究員．東京工業大学学術国際情報センター特別研究員．2006 東京工業大学大学院情報理工学研究科計算工学専攻博士課程修了．博士 (工学)．主に XML データベースに関する研究に従事．IEEE, IEEE Computer Society, 情報処理学会，日本データベース学会，ACM SIGMOD 日本支部各会員．

横田 治夫 Haruo YOKOTA
東京工業大学学術国際情報センター教授．1980 東京工業大学工学部電子物理工学科卒業．1982 同大大学院理工学研究科情報工学専攻修士課程修了．同年富士通（株）．同年 6 月（財）新世代コンピュータ開発機構研究所．1986（株）富士通研究所．1992 北陸先端科学技術大学院大学情報科学研究科助教授．1998 東京工業大学大学院情報理工学研究科計算工学専攻助教授．2001 より現職．工学博士．日本データベース学会理事．電子情報通信学会フェロー．情報処理学会，人工知能学会，IEEE，ACM 各会員．