2006－DBS－140（Ⅱ）（93）
2006／7／14

PCクラスタを用いたXMLデータ並列処理方式の評価

城戸　健太郎[†]　　天笠　俊之[†,††]　　北川　博之[†,††]

概要

近年，XMLは標準のデータ記述フォーマットとして急速に普及しており，数百MBから数GBサイズの大規模なデータをXMLで記述することが一般的に行われるようになってきてている．このような巨大なXMLデータに対して効率的に検索を行う手法として，我々はこれまでPCクラスタを用いたXMLデータの並列処理方式を提案してきた．本稿では，提案方式におけるXPath問合せのコスト計算の詳細を定めるとともに，MPIとlibpqを用いて実装を行いより詳細な性能評価を行う．実験結果より，提案手法は単純な関係表の水平分割アプローチに比べて，約4倍～30倍程度高速であることがわかった．

An Evaluation of a Scheme for Parallel Processing of XML Data using PC Clusters

Kentarou KIDO[†],　Toshiyuki AMAGASA[††,‡‡]　Hiroyuki KITAGAWA[††,‡‡]

Abstract

Recently, with the rapid spread of XML format, it has become popular that large-scale data, whose sizes range from several hundreds of MB to several GB, are described by XML. For the purpose of efficient query processing of huge XML data, we have proposed a scheme for parallel processing of XML data using PC-clusters. In this paper, we attempt to give the details of cost calculation methods by which we can choose efficient query execution plans, discuss an implementation using MPI and libpq, and report an experimental evaluation. From the results, we found that our scheme is 4 to 30 times faster than the baseline approach in that simple horizontal partitioning of relational tables is applied.

1　はじめに

XML（Extensible Markup Language）[1]は，データ記述フォーマットとして急速に普及し，関連技術の研究及び実装が盛んに行われている．XMLデータ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．最近では，大規模なデータをXMLで記述することも多くなっており，数百MBから数GBのデータサイズを持つ大規模なものも少なくない．天文学での観測記録，バイオインフォマティクスにおける遺伝子データなどがその記述フォーマットの例として挙げられる．そのような背景から，大規模なXMLデータを効率的に扱うことのできるXMLデータベースが重要となってきている．

XMLデータベースの実現方法には何通りかの方法が提案されている．中でも関係データベースを利用する方法は，すでに稼動しているシステムが多数あることや，既存の情報資源との連携が取りやすいことなどから，主要な実現方法のひとつである．また，関係XMLデータベースを構築する方法として，XMLデータをノード単位に分解し，それぞれをルートノードからの経路式とともに関係表にマッピングする手法（経路アプローチ）がある[12]．この手法の利点は，実用的なXPathのサブクラスをSQLの機能だけで処理可能な点である．商用システムでもMicrosoft SQLServer 2005がこの手法を採用している．しかし，関係データベースにおけるXMLデータの処理はいくつかの理由から高コストであるため，大規模なXMLデータでは処理効率が悪化することが指摘されている[15]．

この問題に対処するため，我々の研究グループは大規模XMLデータの高速な処理を目的として，PCクラスタによるXMLデータの並列処理方式を提案してきた[14] [9]．具体的には，XMLデータの分割方式を与え．それを用いて，XMLデータを問合せワークロードの各問合せに必要なデータごとに分割する．得られたXMLデータフラグメントは，問合せ（XPath）

[†]筑波大学大学院システム情報工学研究科
Graduate School of Systems and Information Engineering, University of Tsukuba
[††]筑波大学計算科学研究センター
Center for Computational Sciences, University of Tsukuba

のコスト関数をもとに，問合せ全体の処理コストを準最適化するような配置するな配置法を求め，計算ノードへ配置する．本稿では，XPath 問合せのコスト計算の詳細を定めるとともに，MPI と libpq を用いて実装を行いより詳細な性能評価を行う．実験結果より，提案手法は経路アプローチにおけるノード表を水平分割により各計算ノードに割り当て並列的に処理した場合より，約 4 倍〜30 倍程度高速であることがわかった．また，本手法の台数効果を調べ，処理のボトルネックを調べるために問合せ処理時間の内訳について調べた．

本稿の構成は次の通りである．第 2 章では関連研究について述べる．第 3 章では前提となる XML 関連技術について説明する．第 4 章では 提案手法について説明する．第 5 章では提案手法の実装について述べ，第 6 章では評価実験について述べる．最後に第 7 章において，まとめと今後の課題について述べる．

2 関連研究

まず，関係表の分割に関する研究として Anastassia 等の研究について述べる [10]．この研究は，関係データベースの自動チューニングに関する研究である．前提として，問合せワークロードが既知であると仮定し，関係表をグリーディ法により分割するアルゴリズムを提案している．これは問合せワークロードを用い，グリーディ法により関係表を分割している点が本研究に類似しているが，並列処理に着目した処理を考慮してない点，XML データの処理を考慮していない点が異なる．

次に，XML データの分割に関する研究として Brember 等の研究 [7] について述べる．これは，XML データの広域分散環境での格納，検索を可能にするための研究である．このため，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 データの水平・垂直分割を XPath 式によって定義し，分割されたデータを 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．この研究が本研究と本質的に異なる点は，分割された XML データがすでに分散環境に配置されていると仮定している点である．これに対して，本研究では XML データをワークロードを用いて動的に分割し，問合せ処理を最適化する手法を用いている．また，並列処理を考慮していない点が本研究と異なる．また夏目等は XML ファイルに意味を持たせて分割するアルゴリズムを提案すると共に，分割したファイルを自律ディスク上に格納し，履歴を用いて検索する手法を提案している [13].

3 準備

3.1 XML データ

本研究で想定する XML データのモデルを示す．XML データ D は $D = (V, E, r)$ で定義される．ここで，V はノード集合，E は枝集合，r は根ノードであり $r \in V$ である．図 1 に XML データの例を示す.

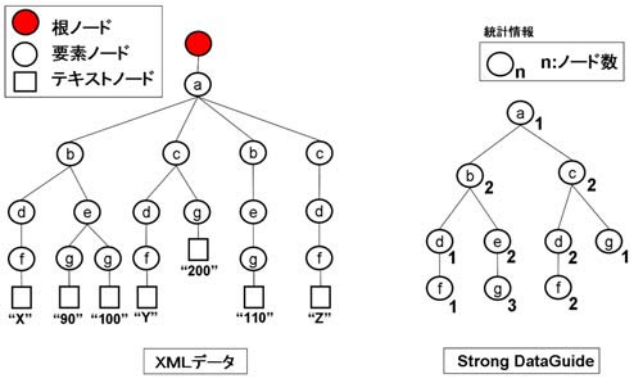

図 1: XML データと対応する DataGuide

3.2 DataGuide

XML データの問合せ処理を効率的に行うためには，XML データの全体の構造を知る手がかりが必要になる．このため，本研究では構造概要と呼ばれる半構造データのための索引構造を利用する．構造概要は，これまでにいくつかの方法が提案されている．本研究ではその中でも XML データを含む半構造データの構造をシンプルに木構造を用いて表現することができる Strong DataGuide[8] を用いる．Strong DataGuide とは，情報源において共通のラベル経路を持つノードを一つのノードに集約し，木構造表現したものである．なお，DataGuide では，情報源をラベル付きの木として用いているが，これを XML に適用する場合は，XML データの要素ノードの名前をラベルとみなす．定義等の詳細については文献 [8] を参照されたい．図 1 の左側に XML データを，右側に XML データから作成した Strong DataGuide ををを示す.

本研究では Strong DataGuide に統計情報（ノード数）を格納し，4.3 節で述べるコストの計算の際に利用する．

3.3 XPath (XML Path Language)

本研究では問合せ言語として XPath[2] を用いる．XPath は 1999 年に W3C 勧告として公開された，XML データの特定の部分を指定するための汎用言語である．XPath ではさまざまな問合せ表現が可能であるが，その表現能力について理論的な側面から評価した研究がある [11]．XPath の言語クラスは XP(L) を用いて表すことができ，L には記述可能な構文を表す．本研究では XP(/, //, []) を対象とする．これはノードテスト，親子関係を指定する child 軸，先祖子孫関係を指定する descendant 軸，および述語を含む問合せを記述することができることを意味する．

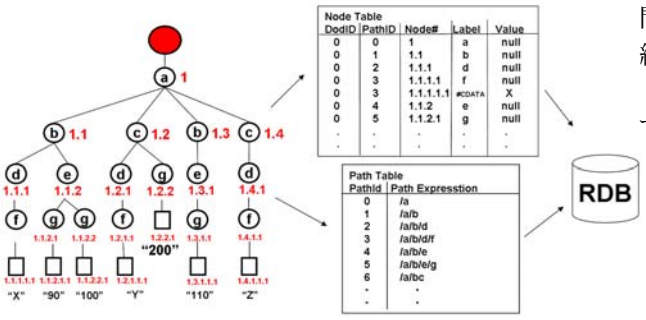

図 2: 経路式に基づく関係表へのマッピング

3.4 関係データベースへの XML データの格納

XML を関係データベースに格納する手法はこれまで多数提案されている．なかでも XML データを経路式に基づき関係データベースにマッピングすることで，DTD や要素の型に依存することなく格納，検索することが可能となる手法がある．これを経路アプローチといい，代表的な手法として XRel[12] が挙げられる．この手法では XML データを解析して得られる木構造をノード単位に分割し，関係表に格納する．関係表に格納されたデータはもとの XML データに対してラベル付けを行うことにより，XML データとして再構築することが可能である．さらに， 問合せ処理の高速化のために，B+木，R 木などの索引構造を利用することができる．本研究では XRel を参考に，XML データを NodeTable(did，pid, nodeid, nodenum, type, value)，PathTable(pid, pexp) の 2 つの表に格納する．関係表にマップされたデータは Dewey Order[3] に基づいてラベル付けをする．XML データを関係データベースに格納した例を図 2 に示す．

問合せ処理については，XPath 式を関係データベースの機能により評価する．具体的には，XPath を解析することによって得られる問合せグラフを SQL に変換する．ここでは話を簡略化するため，述語の入れ子を含まない問合せについて考える（図 3）．

問合せグラフはラベルノード，述語ノード，出力ノードの三つのノードからなる．ラベルノードは経路式に対応し，[] に対応するノードを述語ノードと呼ぶ．最終的に結果として出力される出力ノードであり，問合せグラフ中には必ず一つ，出力ノードが存在する．

問合せグラフを解析することにより，経路式を得ることができる．例えば，`//b[d/f='X']/e` といった問合せからは，`//b`，`//b/d/f`，`//b/d/e` の 3 つの経路式が得られる．

こうして得られた経路式から，次に示す項目を満たす用に SQL を生成する

- SELECT 句には出力ノードを表すテーブル名を記述．
- FROM 句には各ノードに対応するテーブル名を記述．
- SELECT 句では，まずそれぞれの問合せグラフの各ノードの経路式に対応するノードを，テーブルから絞り込むための条件を記述する．
- 述語ノードに関しては述語の内容を満たすように属性 val の条件を記述する．
- 最後にそれぞれのノードの親子関係を満たすように属性 nodenum を用いて条件を記述する．

4 提案手法

4.1 提案手法の概要

図 4 に本研究で提案する提案手法の概要を示す．システムは無共有型の PC クラスタを想定し，各計算ノードには関係データベースが搭載されているとする．システムには入力として検索対象となる XML データと，問合せワークロードが与えられる．システムは，まず経路アプローチに基づき XML データを関係表

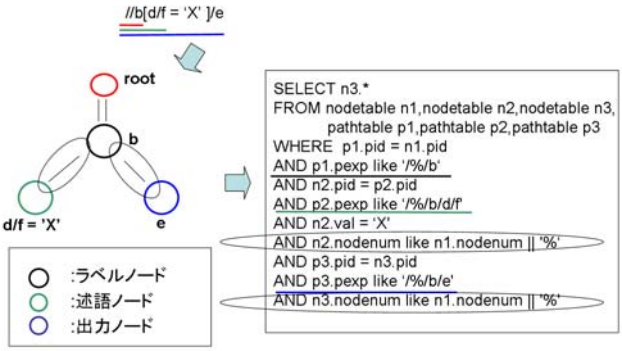

図 3: XPath から SQL の生成

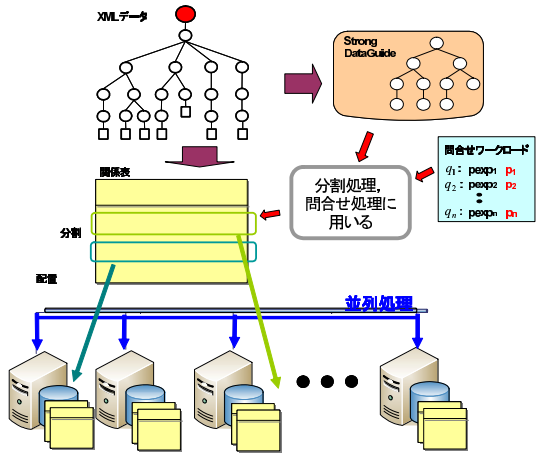

図 4: PC クラスタによる XML データの並列処理

にマップする．その際，XML データの構造概要である DataGuide を同時に生成する．次に，問合せワークロードと DataGuide の情報に基づき関係表を分割し，各計算ノードに割り当てる．問合せ処理を行う際には，問合せの内容からそのデータを保持する計算ノードが特定される．問合せは特定されたいくつかの計算ノードに転送され問合せプランに基づいて協調的に処理される．

4.2 問合せワークロード

本研究で想定する問合せワークロードの定義を示す．問合せワークロードとは XPath 式とその発行頻度の対の集合である．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，その発生確率を合計が 1 になるように正規化した上で用いる．具体的な定義は以下の通りである．

定義 1 (問合せワークロード) 問合せワークロード W は，*XPath* 式 q_i とその発行頻度 r_i の対の集合 $W = \{(q_1, r_1), (q_2, r_2), \ldots, (q_n, r_n)\}$ とする．ただし $\sum_i^n r_i = 1.0$ である．なお，W に含まれる全ての問合せを $W_q = \{q_1, q_2, \ldots, q_n\}$ で表すものとする．□

4.3 XPath 問合せの処理プランとコスト算出

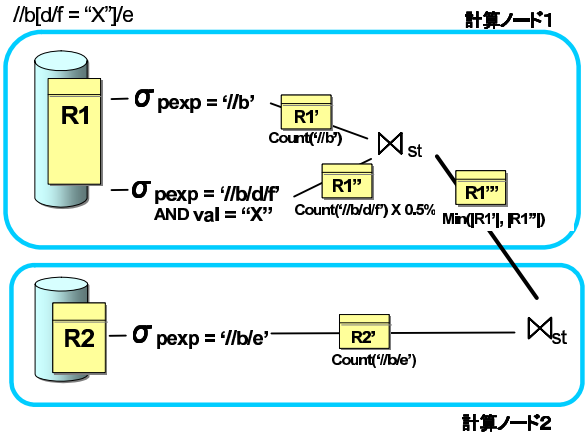

図 5: 問合せ処理

次に，XPath 問合せの処理プランとコスト計算について説明する．

本システムでは，問合せプランは 3.4 節で説明した問合せグラフから導出される．図 3 の問合せグラフを例として説明すると，最初に問合せグラフの葉ノードから出発し，それを対応する関係表からの選択処理に置き換える．葉ノードの処理が終わると，親ラベルノードに着目し，同様に関係表からの選択処理に置き換える．次に，親子間のエッジを XML 木上の構造結合に置き換える．このとき，図のようにあるラベルノードが 2 つの子を持つ場合は 2 段階の構造結合が必要になるが，その順番は後述のコスト関数に基づいてよりコストが少さく見積もられる方のプランを選択することになる．このようにして，出力ノードが得られるまで同様の処理を繰り返すことによって，問合せプランが得られる．

このようにして得られた XPath 問合せプランの処理プランから，実際にコストを算出する方法を説明する．例えば，図 5 のような問合せプランが得られたと

き，まず，関係表 $R1$，$R2$ から経路式//b，//b/d/f，//b/e に該当するタプルを選択する処理 $Sel('//b', R1)$，$Sel('//b/d/f', R1)$，$Sel('//b/e', R2)$ のコストを計算する．これらは，関係表の走査を行うと仮定して，それぞれ選択対象の関係表のサイズ（全タプル数）として見積もる．具体的にはそれぞれ $|R1|$，$|R2|$ となる．選択演算の結果得られるタプル数（次の演算子の入力として使われる）は選択の条件で与えられた経路式に該当とするノード数であるので，DataGuide によってその概数を求めることができる．これをここでは関数 count(pexp) によって表す．なお，選択演算に [] による条件がついている場合は，既存のヒストグラム等の統計情報を用いた推測手法を用いることが考えれるが，ここでは簡単のため等号の場合は 0.5%，不等号の場合は 33.3%などの概数を用いることにする．

次に，選択処理によって得られた関係表 $R1'$，$R1''$ を構造結合するコストを見積もる．ここで，$R1'$ と $R1''$ は同じ計算ノードにあるので，該当する処理オペレータは $Join(R1', R1'')$ である．そのコストは，処理アルゴリズムが入れ子結合であると仮定して $|R1'| \times |R1''|$ と見積もる．また，結果のノード数は構造結合しようとするノード集合のうち，子孫にあたるノード集合のサイズ ($|R1''|$) であるとする．これは $R1''$ の経路式が $R1'$ の経路式を必ず含むという性質による．

最後に，$Join(R1', R1'')$ によって得られた関係表 $R1'''$ と $R2'$ の結合コストを見積もる．このとき，$R1'''$ と $R2'$ は同じ計算ノードにないので，オペレータは $JoinC(R1''', R2')$ である．この場合，結合のコストに加えて関係表を転送するための通信コストが必要となる．このとき，通信コストを最小化するために，$R1'''$ と $R2'$ のうちどちらかデータ量の少ない方を転送する．その通信コストは $Trans(R1''') = \alpha|R1'''|$ であるとする．ここで，α は通信コストの重みを与える係数である．結合演算のコストは，データベースの結合演算ではなく，後述する MPI での実装を我々で行うことに加えて，入力となる $R1'''$ と $R2'$ があらかじめ文書順でソートされていることを仮定できるので，併合結合で実装することができる．このため，コストは $|R1'''| + |R2'|$ となる．

各処理オペレータに対するコスト定義をまとめたものを表 1 に示す．

表 1: 処理コスト

オペレータ	コスト
$Sel(pexp, R)$	$\|R\|$
$Join(R1, R2)$	$\|R1\| \times \|R2\|$
$JoinC(R1, R2)$	$\|R1\| + \|R2\|$
$Trans(R)$	$\alpha\|R\|$

4.4 処理概要

提案手法における処理の概要は以下の通りである．

1. まず，ワークロード $W = \{(q_1, r_1), (q_2, r_2), \ldots, (q_n, r_n)\}$ の XPath 式から，節 3.4 で述べた方法により，全ての経路式を抽出する．なお，述語を含む問合せの場合は，二つ以上の経路式が抽出されることもある．
2. 1 で得られた経路式を元に XML データをフラグメント化し，初期フラグメントを構成する．ここで基本的なルールとして，初期フラグメントに対応するノードを根ノードとする部分木を一つのフラグメントとして扱うこととする．このとき，いつくかの経路式が XML 木上で重複部分を持つことがあるかもしれないが，その場合は，重複が起こらないような分割を求める．具体的には，文献 [14] で提案したスキーマグラフの部分分割による XML データの水平分割，垂直分割を用いる．
3. 2 で得られたフラグメントを計算ノードに配置する．この問題は本質的には組み合わせ最適化問題なので，グリーディアルゴリズムや遺伝的アルゴリズムなどの方法を用いて解を探索する．このとき，どのような配置方法が最適とするかには，いくつかの考え方がある．例えば，ワークロード全体の処理速度が最高となるものを最適とするという考え方，または，各計算ノードの負荷が均等となるものを最適とするといった考え方があり得る．本研究では，まず前者のアプローチをとることで，処理の効率化を狙う．前者のアプローチでは 4.3 節で導入した方法により各計算ノード毎にコストを見積もり，転送コストが最小となるものが最適な配置となる．

5 実装

この章では，提案手法の実装について説明する．

以前，[14] で行った実装では，Java からデータベースを操作する API である JDBC を用いていた．しかし，十分なパフォーマンスを得ることができなかったので，別の手段として MPI (Message Passing Interface)

[4] の利用を検討した．MPI とは，分散メモリ型の並列計算機で，複数のプロセス間でデータをやりとりするために用いるメッセージ通信操作の仕様標準である．ネットワーク転送速度を両者で比較したところ，MPI の方が約 40%ほどよい性能を出すことがわかったので，本稿では MPI により実装を行った．MPI にはいくつかの実装があり，本研究では MPICH2 を用いた．

ここで，MPI プログラムから関係データベースへ問合せを行うためのインターフェースが必要となる．本稿では関係データベースに PostgreSQL を用いており，PostgresSQL サーバへ問合せを行うインターフェースとして libpq[5] を用いた．

6 実験

4 で述べた提案手法において，転送コストがなるべく低くなるようにフラグメントを配置した場合（ワークロード全体の処理効率の向上を狙った場合）に，どれだけの性能向上が得られるかを検証するために実験による評価を行った．

6.1 実験環境

用いた実験環境は，表 2 の通りである．

表 2: 実験環境

CPU	Intel(R) Xeon(TM) 3.0GHz x 4
OS	Red Hat Enterprise Linux 3.0
メモリ	1GB
DB	PostgreSQL 8.1.0
MPI	MPICH2

データセットは XBench プロジェクト [6] で公開されているデータ生成プログラムにより，サイズがそれぞれ 10M，100M，1G であるニュース記事データを生成した．この際，XBench からは複数の XML 文書が生成され，ファイル数はそれぞれ，26，266，2666 であった．

6.2 実験方法

提案手法と比較するためのベースラインとして，XML 文書をその数が均等になるように各計算ノードに割り当て，経路アプローチによって関係データベースに格納する方法を考える．これは，単純な経路アプローチにおけるノード表を水平分割により各計算ノードに割り当て並列的に処理する手法に相当する．

下の 3 つの場合について実験を行った．

1. ワークロードの発生頻度を 5 種類与え，それぞれについて提案手法と比較手法の処理時間を調べる．ワークロードの種類は表 3 の通りである．
2. ワークロードが freq1 のとき，計算ノードの台数を 1, 5, 10 台と変化させた場合に提案手法の処理時間を調べる．
3. また，処理のボトルネックを検証するため，問合せ `/article[prolog/authors/author/name=‘Akira’]/body` について処理時間の内訳を調べる．

1，2 においては，表 3 のワークロードに従う 100 個の問合せを生成し，すべての問合せが実行し終える間での時間を計測した．ワークロード中の問合せに関しては，Q1〜Q5 は述語を含まない単純問合せで，結果サイズは比較的大きくなる．一方，Q6〜Q10 では述語を含み結果のサイズは比較的小さい．

6.2.1 実験 1

実験 1 において，比較手法の結果を図 6 に，提案手法の結果を図 7 に示す．縦軸は処理時間 [second] を，横軸はデータサイズをそれぞれ表す．

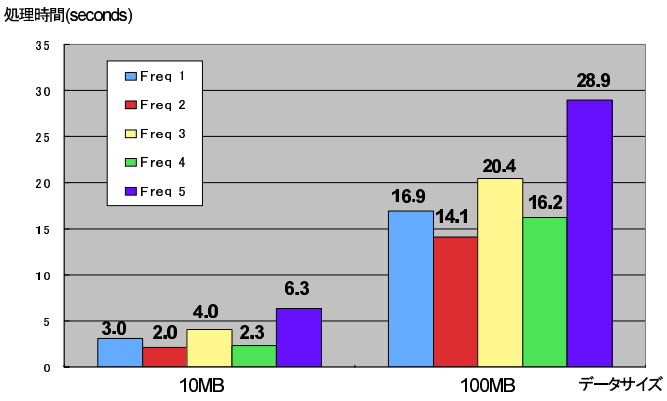

図 6: 実験結果 1.1

まず，実験結果 1.1 と実験結果 1.2 の処理時間を比較すると提案手法を用いた場合が，比較手法より約 4 倍〜30 倍ほど処理が高速であることがわかる．これは，提案手法により，各問合せに必要なデータがフラ

表 3: 問合せセット

問合せ	問合せ式	freq1	freq2	freq3	freq4	freq5
q_1	//authors/email	0.1	0.16	0.04	0.02	0.5
q_2	/article/body/abstract	0.1	0.16	0.04	0.02	0.2
q_3	/article/body/section/@heading	0.1	0.16	0.04	0.02	0.1
q_4	//title	0.1	0.16	0.04	0.03	0.05
q_5	//author	0.1	0.16	0.04	0.03	0.03
q_6	//prolog[keywords/keyword=‘ permanet ’]	0.1	0.04	0.16	0.03	0.03
q_7	//author[name=‘ Joanna ’]	0.1	0.04	0.16	0.05	0.03
q_8	/article[@id=‘ 2 ’]/prolog/title	0.1	0.04	0.16	0.1	0.0
q_9	/article[@lang=‘ en ’]/prolog/title	0.1	0.04	0.16	0.2	0.02
q_{10}	/article[prolog/authors/author/name=‘ Akira ’]/body	0.1	0.04	0.16	0.5	0.02

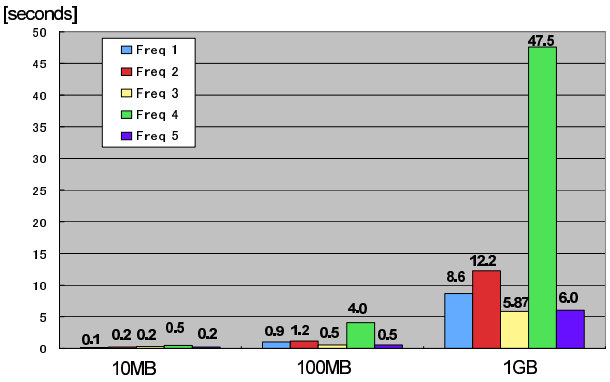

図 7: 実験結果 1.2

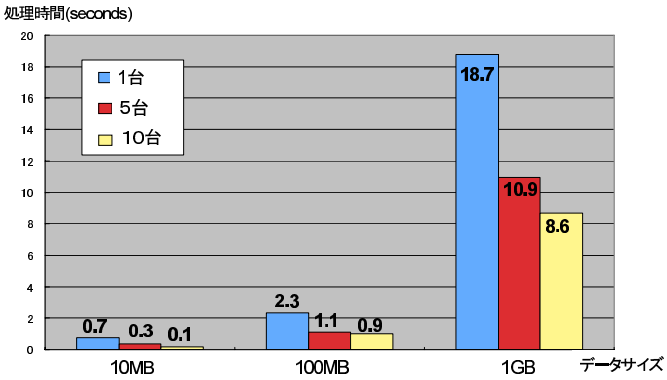

図 8: 実験結果 2.1

グメントとしてまとめられているためであると考えられる．

また，発生頻度の違いに着目すると，比較手法は，比較的複雑な問合せが多く含まれる場合に，処理の効率が悪くなっている．提案手法についてはせ，処理結果のタプル数が多い問合せが多い場合，処理の効率が悪くなっていることがわかる．

6.2.2 実験 2

縦軸は処理時間 [second] を，横軸はデータサイズをそれぞれ表す．

計算ノードの台数が多くなるにつれて，処理速度が上がっていることがわかる．しかし，1 台と 10 台の場合を比較すると，速度はおよそ 2 倍程度しか向上していないことがわかるので，さらに台数効果を追求する必要がある．このため，実験 3 において処理時間の内訳を検討した．

6.2.3 実験 3

提案手法について，それぞれ選択処理，データ転送処理，結合処理が処理全体の中で占める割合を示す．

比較的多くの部分が選択処理が占めていることがわかる．このことから，タプル数が多いフラグメントをさらに水平分割によって計算ノードに配置し，並列的に処理することで選択処理の処理効率を高めることなどが考えられる．

7 まとめ

本研究では，PC クラスタを用いた XML データの並列処理方式について議論した．具体的には，提案方式における XPath 問合せのコスト計算の詳細を定め

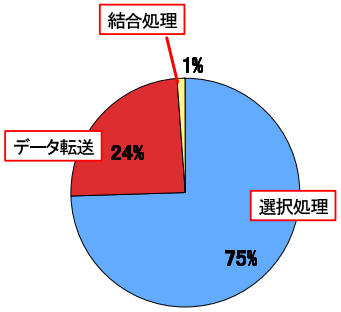

図 9: 実験結果 3

るとともに，MPI と libpq を用いて実装を行い，より詳細な性能評価を行った．実験結果より，提案手法は経路アプローチにおけるノード表を水平分割により各計算ノードに割り当て並列的に処理した場合より，約 4 倍〜30 倍程度高速であることがわかった．また，本手法の台数効果を調べ，処理のボトルネックを調べるために問合せ処理時間の内訳について調べた．

今後は，さらに台数効果を高めるための分割手法を導入することや，フラグメントの配置アルゴリズムを実装し，評価を行うことが挙げられる．

謝辞

本研究の一部は，科学研究費補助金特定領域研究 (#18049005)，若手研究 (B)(#17700110) の支援により行われた．

参考文献

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

[2] World Wide Web consortium: XML Path Language (XPath) Version 1.0, http://www.w3.org/TR/1999/REC-xpath-19991116. W3C Recommendation 16 November 1999.

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

[4] The Message Passing Interface (MPI) standard. http://www-unix.mcs.anl.gov/mpi/.

[5] libpq. http://www.postgresql.jp/document/-pg814doc/html/libpq.html.

[6] A Family of Benchmarks for XML DBMSs (XBench). http://se.uwaterloo.ca/ ddbms/projects/xbench/-Publications.html.

[7] J. Bremer and M. Gerts. On Distributing XML Repositories. *In 6th International Workshop on the Web and Databases WebDB2003*, pp. 73–78, 2004.

[8] R. Goldman and Jennifer Widom. DataGuides: Query Formulation and Optimization in Semistructured Databases. In *Proc. ICDE*, 2002.

[9] Kentarou Kido, Toshiyuki Amagasa, and Hiroyuki Kitagawa. Processing XPath Queries in PC-Clusters Using XML Data Partitioning. In *Proc. SWOD*, 2006.

[10] S. Papadomanolakis and A. Ailamaki. AutoPart: Automatin Schema Design for Large Scientific Databases Using Data Partitioning. In *Proc. SSDBM*, 2004.

[11] Thomas Schwentick. XPath Query Containment. In *ACM SIGMOD Record*, 2004.

[12] M. Yoshikawa, T. Amagasa, T. Shimura, and S. Uemura. XRel: A path-based approach to storage and retrieval of XML documents using relational databases. *ACM Transactions on Internet Technology(TOIT)*, Vol. 1, No. 1, pp. 110–141, 2001.

[13] 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 *Proc. DEWS2001*, 2001.

[14] 城戸健太郎, 天笠俊之, 北川博之. PC クラスタを用いた XML データ並列処理方式の提案. In *Proc. DEWS2006*, 2006.

[15] 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, Vol. 3, No. 2, pp. 33–36, 2004.