PC クラスタを用いた XML データ並列処理方式における性能改善と評価

城戸健太郎† 天笠 俊之†,†† 北川 博之†,††

† 筑波大学大学院システム情報工学研究科 〒 305–8573 茨城県つくば市天王台 1–1–1
†† 筑波大学計算科学研究センター 〒 305–8577 茨城県つくば市天王台 1–1–1
E-mail: †kido@kde.cs.tsukuba.ac.jp, ††{amagasa,kitagawa}@cs.tsukuba.ac.jp

あらまし 近年の XML フォーマットの普及に伴い，大量の XML データを処理する要求が高まっている．この問題に対処するため，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．本論文では，XML データを分割して得られるフラグメントを計算ノードに配置するための，遺伝的アルゴリズムを用いた手法の詳細を述べ，実験によりその有効性を評価する．さらに，システム全体の性能向上を目指し，詳細な性能解析を行ない，パフォーマンスチューニングによる性能の改善を行う．その結果，性能を約 30%改善することができた．
キーワード XML データ管理, 並列・分散 DB，問合せ処理，PC クラスタ

An Improved Parallel Processing of XML Data using PC Clusters and Performance Evaluation

Kentarou KIDO†, Toshiyuki AMAGASA†,††, and Hiroyuki KITAGAWA†,††

† Graduate School of Systems and Information Engineering, University of Tsukuba
1-1-1 Tennodai, Tsukuba, Ibaraki 305-8573, Japan
†† Center for Computational Sciences, University of Tsukuba
1-1-1 Tennodai, Tsukuba, Ibaraki 305-8577, Japan
E-mail: †kido@kde.cs.tsukuba.ac.jp, ††{amagasa,kitagawa}@cs.tsukuba.ac.jp

Abstract Recently, with the rapid spread of XML format, the demand on efficient query processing of huge XML data has been increasing. To fulfill the demand, we have proposed a scheme for parallel processing of XML data using PC-clusters. In this paper, we attempt to give the details of evolutionary search method by which we can obtain pseudo-optimal allocation of decomposed XML fragments to cluster nodes, and evaluate the effectiveness. In addition, we make analysis of the performance of our method, and try to improve the over all system performance. As a result, we obtain 30% of performance gain.
Key words XML, parallel and distributed DB, query processing, PC cluster

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る．また，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データ表現が可能であることが特徴である．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 [2]．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 [3].

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 [4] [5]．具体的には，XML データを DataGuide の各ノードに対応するフラグメントごとに分割する．次に，問合せ（XPath）にかかるコストを見積もり，各計算ノードの負荷が最も下がるようなフラグメントの割り当てを求める．それをもとに，分割された XML データフラグメントを配置する．問合せ処理は，まず，問合せを解析することで得られる問合せグラフを元に問合せプランを導出する．次に，問合せの各オペレーションがどの計算ノードで実行されるかを，計算ノードに割り当てられたフラグメントの情報から決定する．最後に，問合せプランに従って各オペレーションを計算ノード上で並列処理し，実際の検索処理を行う．

しかし，本手法において，XML データフラグメントの割り当てを全数探索で求めるには，計算ノード数を N，フラグメント数を m とすると，$O(m^N)$ のコストがかかり現実的ではない．そこで，本論文では，遺伝的アルゴリズムをこの問題に適用することで，準最適なフラグメントの配置を求める．遺伝的アルゴリズムとは，適当なデータ構造を持つデータを生物の個体と見立てて，生物の進化を計算機上でシミュレートすることにより，与えられた問題に対する準最適的な解を求める手法である．また，本論文では，実験によりその有効性を評価する．その結果，有効なパラメータが得られ，本手法が有効に働いていることを確認することができた．

また，さらなる性能の改善を目指して，CPU 使用率，パケット転送量，問合せ処理時間の内訳を調査し，詳細な性能解析を行う．その結果，問合せ中の選択処理がボトルネックとなり，問合せ処理全体の処理効率が悪化していることが分かった．そこで，本手法における問合せ方法を再検討することにより，性能の約 30%を改善することに成功した．

本論文の構成は次の通りである．第 2 章では前提となる XML 関連技術について説明する．第 3 章ではシステムの概要について説明する．第 4 章では遺伝的アルゴリズムによるフラグメント配置について述べ，その性能について評価する．第 5 章で性能解析とパフォーマンスチューニングについて述べる．第 6 章では関連研究について述べる．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 事 項

2.1 XML と DataGuide

XML は標準のデータ記述フォーマットであり，根ノード，要素ノード，テキストノードによりデータを木構造で表現をすることが可能である．

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．具体的には 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 [6] 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．定義等の詳細については文献 [6] を参照されたい．

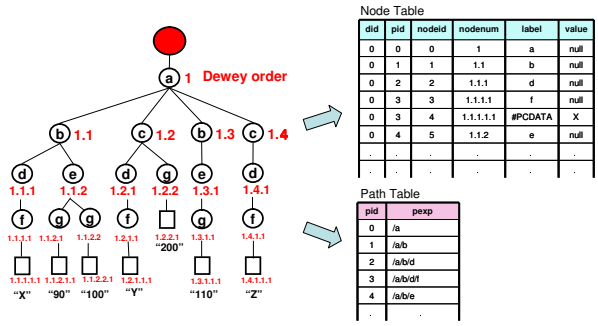

図 1　経路式に基づく関係表へのマッピング

本研究では Strong DataGuide に統計情報（ノード数）を格納し，3.5 節で述べるコストの計算の際に利用する．

3. システムの概要

図 2 に [4] [5] で提案した本研究のシステム概要を示す．計算プラットフォームとしては無共有型の 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．以下では，まず問合せワークロードについて説明し，次に，経路アプローチに基づき XML データを関係表にマップする手法について述べる．

3.1 問合せワークロード

システムに与えられる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とする．ここで，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．□

3.2 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が，本研究では中でも経路アプローチを利用する．これは DTD に依存することなく XML データを格納，検索することが可能となる手法である．

XML データは NodeTable(did, pid, nodeid, nodenum, type, value), PathTable(pid, pexp) の 2 つの関係表に格納する．NodeTable の各属性は，XML 文書 ID，経路式 ID，ノード ID，ノードラベル，要素，属性などのノードの種類，テキスト値を表している．木構造を表現するためのノードラベルには Dewey Order [7] を用いる．PathTable はすべての経路式とその ID を格納する．XML データを関係データベースに格納した例を図 1 に示す．オリジナルの経路アプローチでは，全ての XML ノードを単一の NodeTable に格納する方式を取っているが，この場合，XML データのサイズに応じて関係表のサイズも巨大になっ

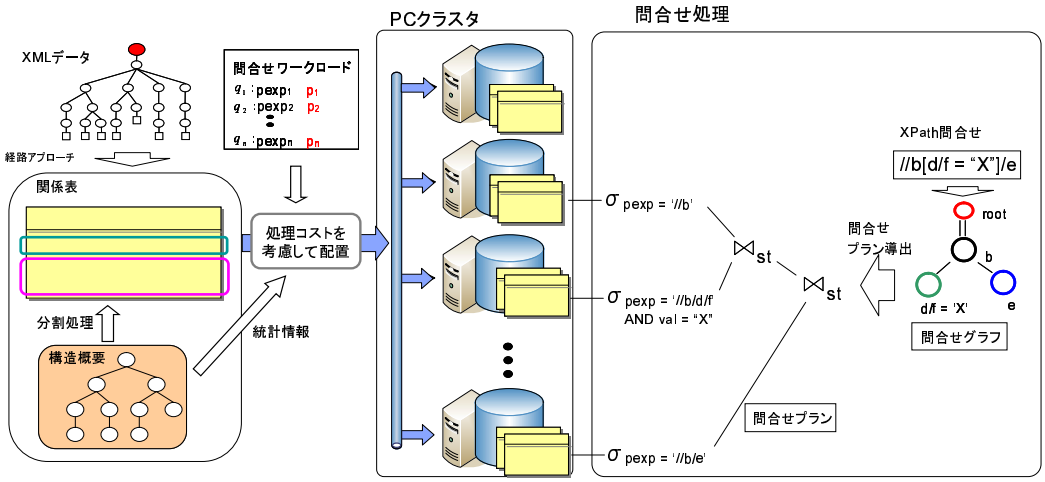

図 2　PC クラスタによる XML データの並列処理手法の概要

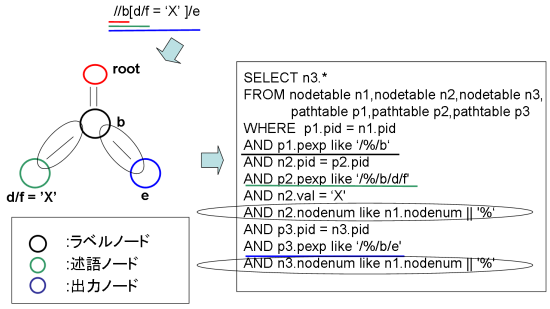

図 3　XPath 式から SQL 問合せへの変換

てしまい，検索性能が悪化するという欠点がある．このため，NodeTable を（何らかの基準で）水平分割し，いくつかの関係表に分割することで，検索性能を向上させることができる．詳細については 3.6 節で述べる．

経路アプローチでは XPath 問合せを SQL に変換した上で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）．問合せ処理は child 軸，descendant 軸，および述語を含むものを対象とする．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．最終的に出力されるノードを出力ノードと呼び，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ように SQL を生成する

- SELECT 句に出力ノードを表すテーブル名を記述する．
- FROM 句に各ノードに対応するテーブル名を記述する．
- WHERE 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．

3.3　XML データの分割，配置処理の概要

3.2 節で関係表に変換した XML データを分割，配置する処理の概要は以下の通りである．

（1） まず DataGuide の各ノードに対応する経路式をすべて抽出する．

（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する．この際，文献 [4] 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．この際，各計算ノードの負荷が，なるべく均等化するような割り当てを導出する．このとき，フラグメントの割り当てを全数探索で求める方法は現実的ではない．そこで，本研究では，システムに与えられる問合せワークロードを手がかりとして，4 章で述べる遺伝的アルゴリズムを用いて準最適な割り当てを探索する．

以下では，まず，与えられた XPath 問合せから各計算ノードの負荷を見積もる方法について説明する．次に，計算ノードに割り当てられたフラグメントの関係データベースへの格納方法について述べる．

3.4　XPath 問合せ処理プランの導出

まず，各計算ノードにかかる問合せコストを計算する際に必要となる問合せ処理プランの導出方法を述べる．

問合せプランは 3.2 節で説明した問合せグラフから導出される．図 2 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 (b) が 2 つの子 (d/f,

表 1　処理コスト

オペレータ	コスト	
	送信側	受信側
$\sigma_{pathexp}(R)$	$\|R\|$	-
$\bowtie_{ST}(R1, R2)$	$\|R1\| \times \|R2\|$	-
$\bowtie_{STC}(R1, R2)$	$\|R1\| \times \log\|R1\| + Trans(R1)$	$\|R2\| \times \log\|R2\| + Trans(R1) + \|R1\| + \|R2\|$
$Trans(R)$	$\alpha\|R\|$	$\alpha\|R\|$

e) 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プランを選択する．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．

3.5　問合せプランのコスト推定

3.4 節のようにして得られた XPath 問合せプランからコストは以下のように推定される．

選択処理のコスト　選択処理 ($\sigma_{pathexp}(R)$) は，関係表の走査を行うものと仮定して，それぞれ選択対象の関係表のサイズ (全タプル数：$|R|$) として見積もる．選択演算の結果得られるタプル数は次の演算子の入力として 使われる．これは，選択 の条件で与えられた経路式に該当するノード数であるので，DataGuide の統計量によって推定できる．選択演算に [] による条件がついている場合は，ヒストグラム等の統計情報を用いた推定手法を用いることが考えられるが，今回の実装では簡単のため，PostgreSQL で用いられているコスト関数を参考にして，等号の場合は 0.5%，不等号の場合は 33.3%といった定数係数を用いることにする．

結合処理のコスト　構造結合は，対象となる関係表が同じ計算ノードにある場合の処理 ($\bowtie_{ST}(R1, R2)$) とそうでない場合の処理 ($\bowtie_{STC}(R1, R2)$) とで処理内容が異なる．

- 結合の対象となる関係表が同じ計算ノードにある場合，処理コストはアルゴリズムが入れ子ループ結合であると仮定して，二つの関係表のタプル数の積で表す ($|R1| \times |R2|$)．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であるとする．
- 同じ計算ノードに無い場合は，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．この場合，通信コストを最小化するために，二つの関係表のうちどちらかデータ量の少ない方を転送する．通信コスト ($Trans(R)$) は送信するタプル数に重みをかけたものとし ($\alpha|R|$)，送信，受信する両方のノードに加算する．また，このときの結合演算は，データベースの結合演算ではなく，実装を我々で行うため，併合結合で実装することができる．そのため，関係表を文書順でソートするコスト ($|R1|log|R1|, |R2|log|R2|$)，結合演算を行うコスト ($|R1| + |R2|$) を加算する．

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．

3.6　フラグメントの格納方式

計算ノードに割り当てられたフラグメントを関係データベースへ格納する方法は，以下の二通りが考えられる．

- 一つの表にまとめて格納する
- フラグメントごとに，別々の表に格納する．

前者の方法では問合せの対象となる表のサイズが大きくなるため，本研究では後者の方法を用いて格納を行い検索性能の低下を防ぐ．関係表の分割方法にはさまざまな方法が考えられるが，本研究では，XML データの検索が経路式に基づくことに着目し，経路式毎に関係表を分割することにする．図 2 を例に説明すると，PathTable の各経路式毎 (/a, /a/b, . . .) に関係表を分割する．その際，関係表の名前には各経路式のパス ID (pid) を付して識別する (fragment1, fragment2, . . .).

この方式は，各関係表のサイズが小さくなることによる性能向上が見込まれるが，//b のような XPath 式を処理する場合，該当する fragmentN 表の集合和 (union) を取る必要がある点に注意が必要である．

4.　遺伝的アルゴリズムによるフラグメント配置方式の評価

4.1　配置方式の概要

一般に，配置するフラグメントの数は計算ノードの数より多い．また，フラグメントによってはワークロード中の使用頻度が大きく異なることがあるので，各計算ノードの負荷がなるべく均等になるようにフラグメントを計算ノードに配置することが重要である．しかし，各計算ノードの負荷を最も下げるような最適なフラグメントの割り当てを全数探索で求めるには，計算ノード数を N，フラグメント数を m とすると $O(m^N)$ のコストがかかり現実的ではない．そこで，本研究では，遺伝的アルゴリズムを用いて準最適なフラグメントの配置を求める．

遺伝的アルゴリズムとは，適当なデータ構造を持つデータを生物の個体と見立てて，生物の進化を計算機上でシミュレートすることにより，ある与えられた問題に対する準最適な解を求める手法である．個体は染色体によって表現され，染色体は遺伝子の集まりから構成される．また，染色体の各位置（遺伝子座）には遺伝子情報が記述されている．個体には適応度が設定され，適合度の高い個体ほど対象となる問題の評価値は最適値に近くなる．

遺伝的アルゴリズムの基本的な処理の動作は次のようになる．まず，固体の初期集合を生成し，各個体に対して適応度の評価を行う．各染色体の適応度が決定されたら，それをもとに自然淘汰を行う．次に，新たな個体集合を生成するために，交叉，突然変異などの遺伝的操作を行う．ここで，自然淘汰とは固体の適応度によって次世代の個体集合を作成する操作，交叉とは一定の確率で選んだ二つの固体の染色体の一部を組み合わせ新

たな個体を作る操作，突然変異とは個体の染色体の一部をランダムに変更する操作である．このようなな世代交代を終了条件を満たすまで繰り返すことにより，最終的に最適解を示す個体を探索することができる．

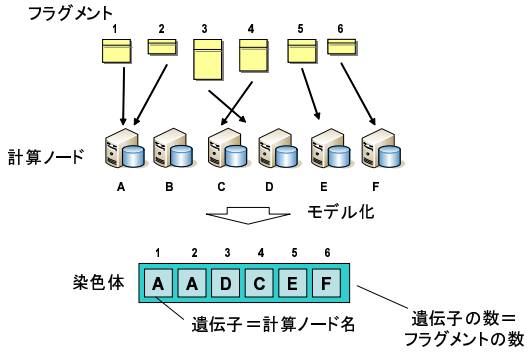

図 4　染　色　体

本研究では，染色体，自然淘汰，交叉，突然変異，適応度の評価を次のように定義することで，遺伝的アルゴリズムをフラグメント配置問題に適用する．

- 染色体：染色体はフラグメントの数だけ要素を持った配列とする．すなわち，遺伝子座はフラグメントの番号を表し，遺伝子は計算ノードの識別番号とする (図 4).
- 自然淘汰：自然淘汰は染色体の集団を評価値の順にソートし，その上位 k 個の染色体を選択する処理とする．
- 交叉：まず，ランダムに 2 つの親となる染色体を選ぶ．次に，ランダムに決定した遺伝子を境に染色体の前半部分と後半部分を取り出し，新たな一つの染色体 を形成する．この処理を自然淘汰された染色体の数だけ繰り返す．
- 突然変異：染色体の中からランダムに遺伝子を取り出し，その中の遺伝子の一つをランダムな値に変更する．
- 評価の方法：まず，各計算ノード毎に問合せにかかるコストを 3.5 で述べた方法を用いて計算する． 具体的には，ワークロード中に含まれる XPath 式の処理コストを，対応する計算ノード毎に合計し，発生頻度によって重み付けを行う．その得られたコストの値から，分散を導き評価値とする．この時，得られ た評価値が高い個体と低い個体の評価値を比較すると，高い個体は低い個体よりも一部の計算ノードに処理が集中し，低い個体は高い個体よりもうまく処理の負荷が分散されていると考える事ができる．
- 終了判定：集団内の最高適応度が一定世代変化しなかった場合に処理を終了する．

遺伝的アルゴリズムでは，設定すべきパラメータ数として，個体数，突然変異率，世代交代数などがある．また，評価をする際には 3.5 節で定義した通信コストの係数 (α) もまた，考慮すべきパラメータである．本研究では，このなかでも通信コストの係数，世代数に着目し，4.2 章で述べる実験を行うことで，その有効性を評価する．

4.2 実　　験

遺伝的アルゴリズムの有効性を検証するため，実験を行った．また，問合せ実行時のシステムのトレースを行い，詳細な性能解析を行った．実装については [5] と同様 MPI と libpq を用いた方法で行っている．

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 8.1.4
MPI	MPICH2

表 3　問合せワークロード

問合せ	問合せ式	freq
Q_1	//authors//email	0.1
Q_2	/article/body/abstract	0.1
Q_3	/article/body/section	0.1
Q_4	//title	0.1
Q_5	//author	0.1
Q_6	//prolog/keywords[keyword=‘ permanet ’]	0.1
Q_7	/article/prolog/dateline[city=‘ Oakland ’]	0.1
Q_8	//author[name=‘ Joanna’]	0.1
Q_9	/article[@id=‘ 2 ’]/prolog/title	0.1
Q_{10}	/article[@lang=‘ en ’]/prolog/title	0.1

4.2.1　実 験 環 境

実験環境は，表 2 のスペックを持つ計算ノードを 10 台用いた．データセットは XBench プロジェクト [8] で公開されているデータ生成プログラムにより，サイズが約 10G 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 16,999 であった．

4.2.2　実 験 方 法

次の実験を行った．

（1） コスト計算に用いた通信コスト α の値による問合せ処理時間の変化を調べる．

（2） 遺伝的アルゴリズムが有効に働いているかを検証するため，遺伝的アルゴリズムの世代交代数を特定の位置で打ち切り，その時点での配置状況による問合せ処理時間，また，処理を打ち切るまでの遺伝的アルゴリズムの計算時間を調べる．さらに，打ち切った世代数ごとのフラグメントの配置状態を調べる．

実験では，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．

4.2.3　実　　験　　1

実験 1 の結果を図 5 に示す．縦軸は処理時間 [second] を，横軸は通信コストの係数 α をそれぞれ示す．α が小さくなるにつれて，処理時間が短くなっている事が分かる． これは，α の値によってフラグメントの分散具合が代わり，それによって各計算ノードにかかる負荷が変化するためである．この場合は

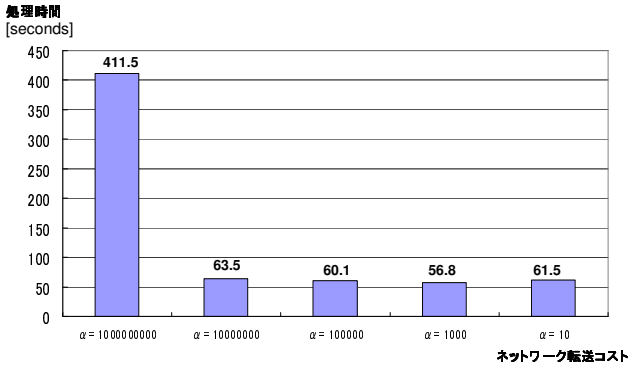

図 5　通信コストの重みに対する処理時間

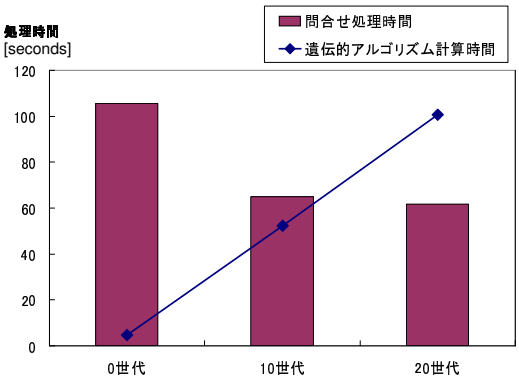

図 6　処理を打ち切る世代数に対する問合せ処理時間と遺伝的アルゴリズムの計算時間

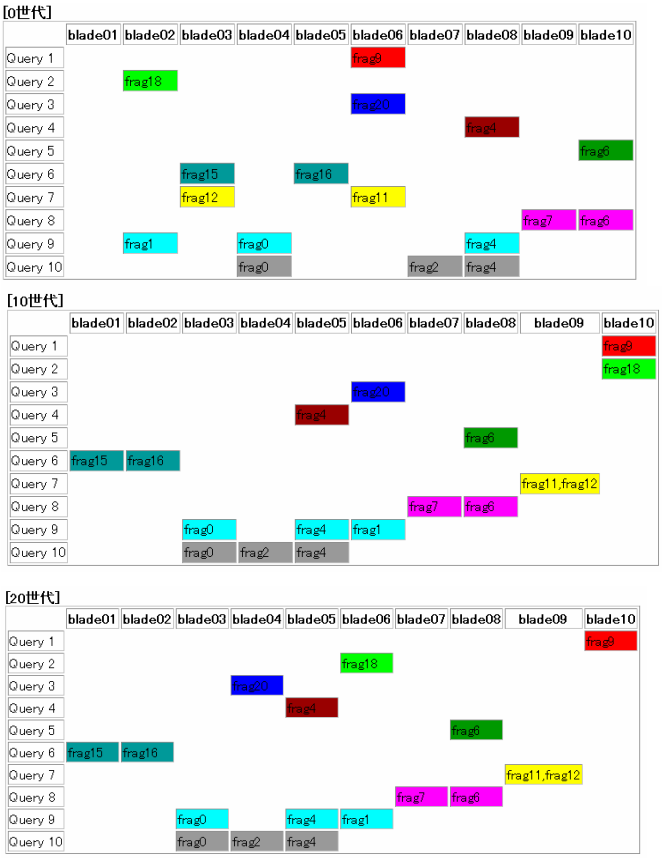

図 7　世代数の変化に対するフラグメント配置状況

$\alpha = 1000$ の場合がもっとも処理時間が早くなっていることが分かる．α が 1000 より大きい場合は，必要以上にフラグメントが一つのノードに固まっている，小さい場合は散らばり過ぎていると考える事ができる．

4.2.4　実　験　2

実験 2 の結果を図 6 に示す．縦軸は処理時間 [second] を，横軸は遺伝的アルゴリズムにおいて処理を打ち切った世代数を示す．このとき，ネットワーク転送コストの係数は実験 1 で最良の結果を示した $\alpha = 1000$ を用いた．処理を打ち切る世代数が長くなる分，問合せ処理の効率がよくなっていることが分かる．しかし，同時に遺伝的アルゴリズムの処理時間を見ると，打ち切る世代数に比例して処理時間が長くなっている．このことから，問合せ処理時間がある程度効率的に行うことができる配置が求められた時点で，遺伝的アルゴリズムの処理を打ち切ることで，システム全体としては効率的になると考えられる．

また，図 7 には世代数の変化に対するフラグメント配置状況を示す．ワークロードに含まれる問合せに必要なフラグメントが，どの計算ノードに割り当てられているかを示している．この図から，0 世代の時の配置と 10 世代の時の配置を比べると，計算ノードが均等に利用されていることが分かる．また，10 世代の時の配置と 20 世代の時の配置を比べると，選択処理の負荷がが高い（フラグメントのサイズが大きい）フラグメントがより，コストが低くなるように配置がなされている．このとき，0 世代とはランダムにフラグメントを計算ノードに割り当てた状態である．

5.　性能解析とパフォーマンスチューニング

4 章では，遺伝的アルゴリズムにより最適なフラグメントの配置を求めることで，問合せ処理の効率化を目指したが，この章では，さらなるシステム全体の性能向上を目指し，CPU 使用率，パケット転送量，問合せ処理時間の内訳を調べ，詳細な性能解析を行った．また，解析結果をもとに問合せ処理方法を再検討することでパフォーマンスチューニングを行った．

5.1　性 能 解 析

4.2.3 節で得られた最適なパラメータを用いた処理時に，システムのトレースを行った．具体的には，問合せ処理時間の内訳，計算ノードの CPU 使用率，パケット転送量を調べた．実験環境，問合せワークロードは 4 章で行った実験と同様である．

5.2　解 析 結 果

性能解析の結果をそれぞれ，図 8-12 に示す．図 8 は，問合せのうち，Q9 の処理の内訳を表したものであり，縦軸は処理時間 [second] を，横軸は問合せを処理している計算ノードを示す．この問合せでは，計算ノード 3，計算ノード 5，計算ノード 6 の三つのノードに配置されたフラグメントを用いている．図 8 の結果を見ると，選択処理が問合せ処理の大半を占めていることが分かる．また，図 9 はワークロードに含まれるすべての問合せ処理の内訳を示した結果とそのオペレーションである．問合せ処理はそれがワークロード中のどの問合せ処理に該当するかで色分けをしている．図 9 を見ると，ワークロード全体で

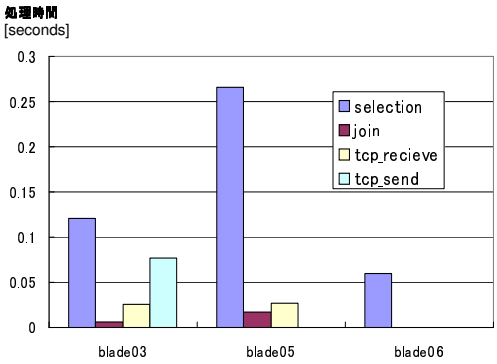

図 8　Q9 の処理の内訳

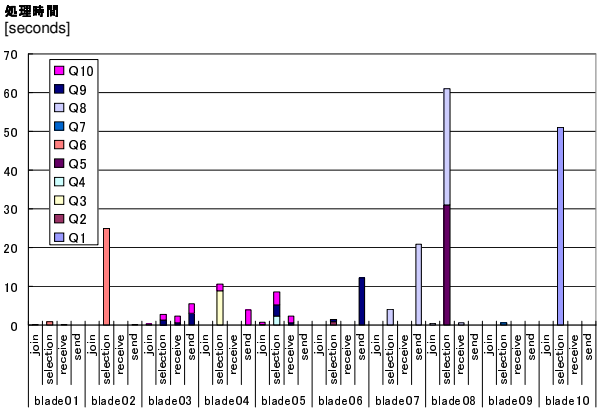

図 9　ワークロード全体の処理の内訳

も，処理の大部分は選択処理によって占められている．なかでも計算ノード 8，計算ノード 10 おいて，選択処理に多くの時間がかかっていることが分かる．

図 10-12 は，それぞれ計算ノード 1, 8, 10 における CPU 使用率，ネットワークの転送量の推移を表したグラフである．計算ノード 8, 10 では，ネットワークの転送処理は行われていないが，計算ノード 8 では Q5，Q8 の選択処理，計算ノード 10 では Q1 の選択処理に，CPU が処理時間全体にわたって使われていることが分かる．また，このとき，計算ノード 1 は処理時間の前半ではネットワークの転送処理を伴う問合せを処理しているが，処理の終了後は計算資源が空いている事が分かる．

以上の考察から，システムのボトルネックとなっている処理は選択処理であり，この処理時間を短縮することで，パフォーマンスの向上が見込まれる．

5.3　処理方法の変更とその評価

3.6 節で述べたように，本システムにおいて各フラグメントは，すでに単一の経路式にのみ対応した状態で関係表に格納されている．このため，問合せによっては SQL における経路式を用いた絞り込みが不要である．このことから，選択処理時に PathTable との結合処理を可能な限り省くことで処理をより効率化する．図 4 の問合せを例に具体的に説明すると，図 4 の SQL は，次のように記述し直すことが可能となる．

```
SELECT n3.*
FROM nodetable n1,nodetable n2,
     nodetable n3
WHERE n2.val = 'X'
AND   n2.nodenum like n1.nodenum || '%'
AND   n3.nodenum like n1.nodenum || '%'
```

この処理方法の変更によりどの程度パフォーマンスが向上するかを検証するため，同様の評価を行った．その結果を図 13 に示す．図 13 と変更前の図 9 を比較すると，計算ノード 8，計算ノード 10 における選択処理にかかる時間が短縮されていることが分かる．また，処理時間を比較すると変更前は 61.6[seconds]，変更後は 39.7[seconds] と，約 30%性能が改善されていることが分かった．

しかしながら，依然として計算ノード間の負荷には大きな差がある．そこで今後は，さらなる選択処理の効率化を目指し，XML データの水平分割を用いることを検討する予定である．

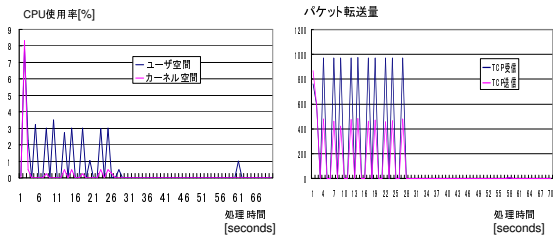

図 10　計算ノード 1 における CPU 使用率とパケット転送量

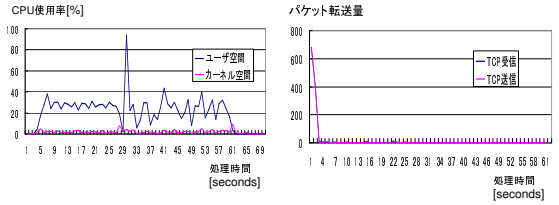

図 11　計算ノード 8 における CPU 使用率とパケット転送量

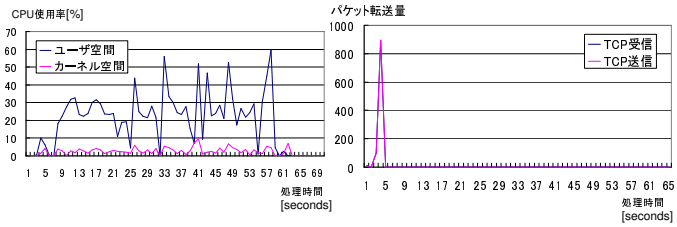

図 12　計算ノード 10 における CPU 使用率とパケット転送量

6.　関 連 研 究

関係データベースを用いた XML データの並列処理に関する研究としては，Qin 等の研究 [9] がある．この研究では，まず，XML データのエッジに着目して XML データをパス表，エッジ表の二つの関係表にマップし，次の方法で計算ノードに配置する．1) 相互に独立した XML データは，別々の計算ノードに配置する．2) 関係のある XML データは同じ計算ノードに配置する．3) サイズの大きい XML データはグラフ分割アルゴリズムにより，より小さいフラグメントに分割する．この研究では，XML データの分割方式が詳細に定められていない点，計算ノードの負荷を考慮しフラグメントが配置されていない点，

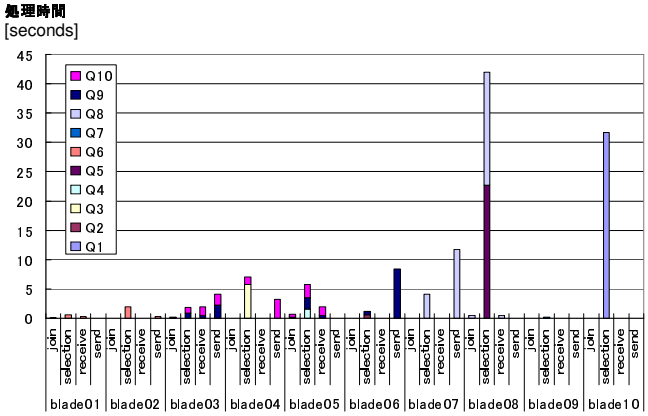

図 13　処理方法の変更後のワークロード全体の処理の内訳

また，評価実験を行っていない点が本研究と異なる．Lu 等もまた，エッジ写像を用いて XML データを関係表にマップし，関係表の水平分割，垂直分割を行うことで並列処理を行う方法を議論している [10]．しかし，単純な関係表の分割方法に基づいている点，評価実験が無いことが本研究と異なる．Tang 等は，並列 XML データベースにおける最適なフラグメントの配置方法として，ワークロードの情報を用い，コストモデルを定めることで準最適なフラグメントの配置を求めるアルゴリズムを提案しているが，各計算ノードにオブジェクト指向 XML データベースを用いている点が本研究と異なる [11]．

関係表の分割に関する研究としては Ailamaki 等の研究がある [12]．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システム対して準最適になるように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．

XML データの分割に関する研究として Brember 等の研究 [13] 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 [14]．

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．本論文では，遺伝的アルゴリズムによるフラグメント配置方式の詳細を述べるとともに，実験により有効性を評価した．また，システム全体のトレースなど，詳細な性能解析を行ない，パフォーマンスチューニングによる性能の改善を図った．

しかし，5.3 節の実験の結果から分かるように，依然として選択処理が処理時間の大半を占めていることから，今後は，さらに選択処理の効率化を目指し XML データの水平・垂直分割の効果的な組み合わせを検討することが挙げられる．また，ワークロードの変化への対応も検討すべき課題である．

謝　　辞

本研究の一部は，科学技術振興事業団戦略的基礎研究推進事業 CREST における研究領域「情報社会を支える新しい高性能情報処理技術」の研究課題「ディペンダブルで高性能な先進ストレージシステム」，科学研究費補助金特定領域研究 (#18049005)，基盤研究 (B)(#15300029)，若手研究 (B)(#17700110) の支援により行われた．

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] 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), **1**, 1, pp. 110–141 (2001).

[3] 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, **3**, 2, pp. 33–36 (2004).

[4] 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”, Proc. DEWS2006 (2006).

[5] 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実装と評価”, Proc. DBWS2006 (2006).

[6] R. Goldman and J. Widom: “DataGuides: Query Formulation and Optimization in Semistructured Databases”, Proc. ICDE (2002).

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

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

[9] J. Qin, S. Yand and W. Dou: “Parallel Storing and Querying XML Documents Using Relational DBMS”, Proc. APPT, pp. 629–633 (2003).

[10] K. Lü and Y. Zhu and W. Sun and S. Lin and J. Fan: “Parallel Processing XML Documents”, Proc. IDEAS, pp. 96–105 (2002).

[11] N. Tnag, J. X. Yu, K. f. Wong and G. Yu: “WIN: An Efficient Data Placement Strategy for Parallel XML Databases”, Proc. ICPADS, pp. 349–355 (2005).

[12] S. Papadomanolakis and A. Ailamaki: “AutoPart: Automatin Schema Design for Large Scientific Databases Using Data Partitioning”, Proc. SSDBM (2004).

[13] J. Bremer and M. Gerts: “On Distributing XML Repositories”, In 6th International Workshop on the Web and Databases WebDB2003, pp. 73–78 (2004).

[14] 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”, Proc. DEWS2001 (2001).