社団法人　情報処理学会　研究報告
IPSJ SIG Technical Report

2006－FI－82（4）
2006－DD－54（4）
2006／3／22

大規模 XML データに対する分散問合せ処理の効率化

栗田裕人[†]　宮崎純[†]　波多野賢治[†]
中島伸介[†]　植村俊亮[†]

本稿では，大規模な XML データを効率良く処理するために，XML データを分割し，それらを複数の計算機に配置した上で，問合せ処理する手法を提案する．データの分割では，データ量とデータ構造を考慮した分割を行い，各計算機へ分割したデータの分散配置を行う．また，特定の計算機へ問合せが集中した場合，データの動的な再配置を行うことで負荷分散を実現する．最終的には各計算機での問合せ処理の負荷を均一にすることで，効率のよい分散問合せ処理を実現できる．評価実験において，複数の計算機からなる分散問合せ処理システムに提案手法を適用したところ，問合せ処理時間が約 20%高速化された．

Efficiency of Distributed Query Processing for Huge XML Data

Hiroto KURITA,[†] Jun MIYAZAKI,[†] Kenji HATANO,[†]
Shinsuke NAKAJIMA[†] and Shunsuke UEMURA[†]

We propose an efficient query processing for huge XML data by partitioning and distributing XML data to multiple computation nodes. This scheme is composed of three parts; XML data partitioning, XML data distributing, and query processing on multiple computation nodes. Data partitioning is based on the size of data as well as its data structure. Each fragment of data is distributed to multiple computation nodes, so that CPU load of query processing on each node can be balanced. In addition, it is important to take the balance of query processing on each computation nodes into consideration to implement efficient query processing. In our experimental evaluation, we could confirm that our proposal has a performance advantage in query processing of huge XML data.

1. まえがき

XML (Extensible Markup Language) [8] は，インターネット上におけるデータ交換を容易にするための基盤技術として，W3C (World Wide Web Consortium) から勧告されている．

XML は，電子的な文書管理を目的として開発された言語である SGML (Standard Generalized Markup Language) [4] のサブセットとして，簡明な表現が行えるよう言語仕様が規定されるとともに，電子的なデータ交換の役割も担えるよう設計が行われている．このため，XML は文書の表現形式のみにとどまらず，より広いデータ交換の形式として急速に普及しつつある．

また，様々なデータが XML で記述されるにつれて，その利用法も広がってきているが，その中でも特許データやゲノム情報データ，気象観測記録のデータなど，数百メガバイトから数テラバイト規模のデータサイズを持つ XML データも出現しており，今後もこのような大規模データが増加していくと考えられる．

XML データの処理は，通常各プログラミング言語で用意された API(Application Programming Interface) を用いて XML データをパーズし，アプリケーションで処理しやすい情報に変換する．

API には，主に DOM (Document Object Model) [7] や SAX (Simple API for XML)☆が用いられるが，どちらの API で XML データを処理する場合でも，多くの場合その処理コストは XML データのサイズに依存する．そのためギガバイトを超えるような大規模 XML データの処理では処理コストが高くなり，ユーザからの問合せが頻繁に発生するような環境では，特に効率のよい問合せ処理の実現が急務である．

一方，大規模 XML データを単一の計算機で処理する場合，CPU の処理能力やメモリ，ハードディスク容量など，ハードウェアの制限に直面することが多い．こうした問題に対処するには，大規模 XML データを

† 奈良先端科学技術大学院大学情報科学研究科
〒 630–0192 けいはんな学術研究都市
Graduate School of Information Science, Nara Institute of Science and Technology (NAIST)
Keihanna Science City, Ikoma, Nara 630–0192, Japan

☆ http://www.saxproject.org/

効率よく処理できるよう，XML データを分割し，複数の計算機に分散配置して，分散問合せ処理をする方法が考えられる．

XML データの分散処理の利点としては，並列処理による処理速度の向上，頻繁な問合せによる負荷の分散と，データ容量拡大の際のスケーラビリティ，障害発生時の可用性の確保などが挙げられる．

大規模 XML データに対する分散問合せ処理環境を構築する際の留意点を以下に述べる．

(1) XML データの分割
XML データの分割は，データが木構造であることから，単に分割するのは容易である．しかし，問合せ処理を効率よく行うためには，各計算機に配置するデータサイズを均等にし，分割したデータの木構造を保持しなければならない．このため，データのサイズと構造を考慮した分割アルゴリズムを考える必要がある．

(2) XML データの分散配置
分割したデータの分散配置は，データサイズと構造を考慮する必要がある．XML の処理コストはデータサイズに依存するため，各計算機のデータサイズが均等になるようにデータの配置を行う．また，問合せを効率よく処理するには，配置される XML データの構造も考慮しなければならない．なぜなら構造を考慮せずにデータを配置した場合，特定の計算機に問合せが集中してしまうことが考えられるからである．その場合，分散処理の利点である並列処理の効率が失われる．このため，データサイズと問合せによる負荷を, 各計算機で均一にする手法の提案が必要となる．

(3) 分散問合せ処理
分散問合せ処理では，分割されたデータがどの計算機に配置されているのかを，データ配置を行う際に問合せを管理する計算機が認識しておく必要がある．問合せを管理する計算機が，ユーザから受け取った問合せを解析し，問合せに関係するデータが配置されている計算機へ問合せを発行する．分散問合せ処理では，問合せ処理を行う計算機から問合せを管理する計算機に結果が返されるため，最終結果を得るためには，それらの統合処理が必要となる．

(4) XML データの動的な再配置
データサイズと構造を考慮したデータの分割，配置を行ったとしても問合せによるデータへのアクセス頻度や，問合せの発行頻度が変化した場合，各計算機の負荷に偏りが生じることが考えられる．その際データの動的な再配置を行うことで偏った負荷を均一にする処理が必要である．データの動的な再配置は，移動するデータの判断や，データ移動の方法，データ移動後の問合せ処理方法などが必要となる．

本稿では，実際に複数の計算機を用いて分散問合せ処理システムを構築し，先に説明した四つの留意点を考慮した実装を行うことで，問合せ処理が効率化できることを確認する．

2. 関 連 研 究

リレーショナルデータベースにおける分散処理に関する研究は，これまで多くの研究がなされているが，XML データに対する分散問合せ処理に関する研究は最近始められたばかりである．

分散リレーショナルデータベースのデータ分割および動的なデータ再配置は古典的な問題であり，様々な手法が提案されている [2,3,9]．互いに独立した小さなタプルの集合であるリレーショナルデータベースであるがゆえ，データ分割および再配置は簡単である．しかしながら，XML データベースの場合，データベース自体に構造を持つため，データ分割は容易ではなく，データの動的再配置もリレーショナルデータベースのようにタプル単位の細かな制御を行うことができない．

数少ない XML の分散処理手法の一つに Bremer らの研究がある [1]．彼らの研究は，XML データに対する分散問合せ処理の効率化に関する研究であり，Repository Guide と呼ばれる構造概要を用いて分割された XML データを管理し，それと三種類の索引を関連付けることで，分散環境での問合せ処理の効率化を実現している．分割の方法としては，XPath 式によってデータの垂直・水平分割を定義しており，データの構造を考慮した分割法といえる．しかし，特定の計算機に問合せ処理の負荷が集中する場合に対応できないといった問題がある．これに対し我々の手法は，分割された XML データへの不均一なデータアクセスに対しても，分割データの動的な再配置処理を行うことで対応できる．

また中尾らは，XML データに対する問合せ処理を効率化するための XML データ分割方式を提案している [5]．文献 [5] では，問合せによるデータアクセスの偏りを元に XML データを分割し，問合せ処理に必要な部分データだけを用いて問合せ処理を行うことで，処理の効率化を実現している．データ分割の際にはコスト関数を定義し，各問合せの発行頻度と処理コストから，処理効率が最適となる断片候補を導き出し，データの分割を行っている．

しかし，問合せの発行頻度が既知である場合を想定しているため，問合せ頻度に変化がある場合には対応できないという問題点がある．これに対し我々の手法

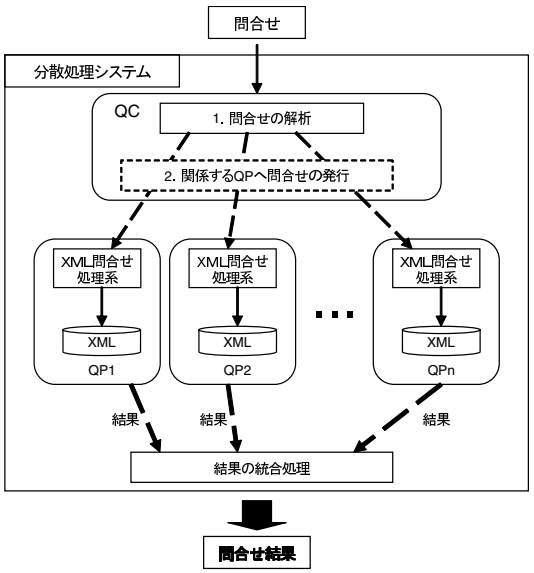

図 1 問合せ処理の流れ

は，問合せが発行されるたびに分割された XML データの問合せ処理時間を計測した上で，データの動的な再配置処理を行っているため，そうした問題は起こらない．

3. 分散問合せ処理システム

本節では，提案する XML データに対する分散問合せ処理システムについて述べる．1 節で述べたように，大規模 XML データに対する効率のよい問合せ処理の実現には，XML データの分割，分割したデータの分散配置，発行された問合せの分散処理，そして，分割 XML データに対する問合せ処理時間に基づいたデータの動的な再配置を考慮する必要がある．以下システムの概要から順に詳述する．

3.1 分散問合せ処理システムの概要

提案システムの概要を図 1 に示す．以下，図 1 を用いながら問合せ処理の流れを説明する．

まず，準備段階として大規模 XML データを分割し，断片データを複数の計算機へ分散配置する．ユーザからの送られてきた問合せは，図 1 の上部に位置する計算機 (Query Contoroller: 以下 QC と表記) が受け取り，問合せの解析処理を行う．QC は分割された XML データが配置された計算機 (Query Processor: 以下 QP と表記) に対して，問合せを送信する．QC から問合せを受け取った QP は，XML 問合せ処理系を介して，問合せに関連する分割 XML データに対してだけ問合せを発行する．QP から得られた処理結果は QC に送られ，必要であれば結果を結合し，問合せ結果をユーザに出力する．

3.2 XML データの分割処理

XML データの分散処理を行う場合，各 QP の処理コストを一定にすることが重要である．つまり，各 QP に配置するデータサイズと問合せによる負荷が均等になるようなデータの配置を考えなければならない．そこで本稿では，XML データを分散配置する各 QP の処理コストを一定にするために，データサイズとデータ構造の両方を考慮したデータ分割の方法を提案する．

提案手法により，分割される XML データは同程度のサイズとなり，かつ元データの構造をできるだけ保持できるようになる．このような設計をした理由は，各 QP が持つ分割 XML データのサイズを均等にしなければ，サイズの大きな分割 XML データが高負荷になる可能性があるからである．また，XML データは部分木ごとにまとまった意味を持つことが多く，構造を考慮した XML データ分割を行わなければ，問合せ処理時に複数の分割 XML データに同時にアクセスされる可能性が高く，問合せ処理のコストが高くなってしまうからでもある．つまり，分割 XML データのサイズと構造を考慮した分割を行うことで，複数の分割 XML データに問合せが発行されないよう工夫し，かつ各 QP の処理コストを均一にする効果を期待した設計となっている．

XML データのサイズを M，XML データの分割数を N，分割後の XML データの理想サイズを $\alpha = \frac{M}{N}$，許容サイズを $\alpha(1-\epsilon) \leq L \leq \alpha(1+\epsilon)$ としたとき，本稿で提案する XML データの分割アルゴリズムを以下に示す．

(1) XML データを一度パーズし，各ノードにおける子ノードの数とそのノードを root とする部分木のサイズを取得する．

(2) XML データのあるノード n において，その子ノード n_c のサイズ S_{n_c} を評価する．S_{n_c} が許容範囲のサイズ L を満たした場合，n_c を root とする部分木を断片データとして保持する．

(3) S_{n_c} が許容範囲のサイズ L に比べ大きな場合，n の子ノードのうちサイズの大きな子ノードから順に，n_c の子ノード (n の孫ノード) n_g にアクセスし，(2) の処理を再帰的に行う．

(4) S_{n_c} が許容範囲のサイズ L に比べ小さな場合，文書順に n_c の弟ノード n_cb のサイズ S_{n_cb} を取得する．$S_{n_c}+S_{n_cb}$ のサイズが α の値を超えるまで, 文書順に弟ノードを結合する．α の値を超えた時点で, 結合したノード n を root とする部分木を断片データとして保持する．

(5) $S_{n_c}+S_{n_cb}$ の値が α を超えないうちに, 弟ノードが無くなった場合は $S_{n_c}+S_{n_cb}$ のサイズを S_n とし,n の兄弟の長男から順に処理 (4) を繰

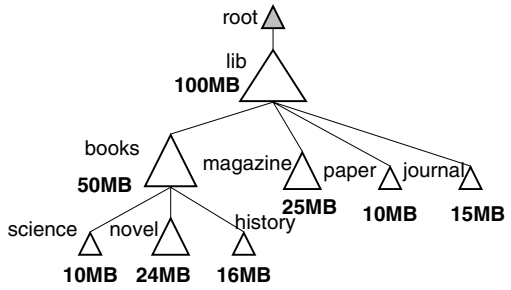

図 2 XML データの例

り返す.

(6) (1)〜(5) の処理を分割対象である XML データの root ノードから文書順に処理を行う．全てのノードを評価して処理を終了する．

この分割アルゴリズムの処理例を図 2 を用いながら説明する．各ノードに付随している数字は，そのノードを root ノードとした場合の部分木のサイズを表している．例えば books ノードは，子ノードとして science, novel, history ノードを持ち，これら三つのノードサイズを合せた 50MB が，部分木 books のサイズとなる．この XML データを四つの断片データに分割する場合，分割アルゴリズムを用いると以下のようになる．初期状態として，データサイズ $M = 100$, 分割数 $N = 4$ を入力し，理想サイズ $\alpha = 25$ となる．また許容範囲 $\epsilon = 0.1$ とすると，許容サイズは $22.5 < L < 27.5$ となる．この情報を基に処理を実行する．

(1) まず，ノード lib の子ノードは books，magazine，paper，journal の四つであり，文書順にノード books から処理する．

(2) ノード books のサイズは L の値を超えるため，books の子ノードを評価する．books の子ノードは science，novel，history の三つであり，それぞれ文書順に評価する．

(3) ノード science のサイズは L より小さいため，ストックしておく．

(4) 次に，ノード novel のサイズは許容範囲のサイズ L を満たすため，ノード novel を root とする部分木を断片データとして保持する．

(5) ノード history のサイズは L より小さいため，先にストックされていたノード science と結合させる．この際，結合後のデータサイズが L を満たすため，この結合後のデータを断片データとして保持する．

以上のようなアルゴリズムに従って XML データの分割処理を行った結果，図 3 のような四つの断片データができる．また，問合せ処理のために，各断片データに識別子（以下，データ id と表記) を付け，親ノードをたどり root ノードまでのタグを補完する．

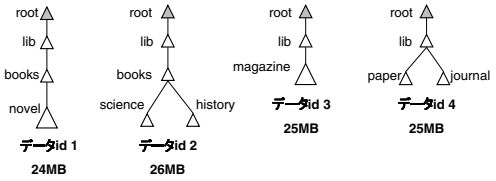

図 3 分割後の XML データ

この分割アルゴリズムは，XML データのサイズ M と分割数 N を初期状態とし，子ノードの数とサイズの情報を用いて処理を行っている．したがって，XML データサイズとその構造の両方を考慮した分割を行うことができる．

3.3 データの分散配置

分割した断片データを分散配置する段階では，実際にどのような問合せが，どのような頻度で発行されるかはわからない．そこでまず，各ノードに対するアクセス頻度が均等であると仮定して配置することを考える．3.2 節で述べた分割アルゴリズムは，データサイズと構造の両方を考慮した分割を実現しているため，各 QP に断片データを同数配置すれば各 QP の負荷は均一になると予測できる．

実際には，発行される問合せによって，アクセスされるデータには偏りが生じるため，QP の負荷は均一にならない．この問題を解決するために，本稿では QP に配置した断片データをアクセス頻度に応じて，分割 XML データを動的に再配置する方法 (3.5 節に詳述) の提案も行っている．そのため，データ配置の段階で，データの動的な再配置を考慮した配置を行わなければならない．本システムのデータの配置は一つの QP に複数の断片データを配置する方法で行う．

断片データを配置した後，各断片データをどの QP に配置したかを示すマップ (以下データマップと表記) を作成し QC に保持させる．このデータマップは，キーとして root ノードから各ノードへの path (ロケーションパス) を持ち，そのキーに関連するデータ id と，その断片データを配置した QP の識別子（以下 $QPid$ と表記）を値として保持する．

データマップは，断片データに含まれるロケーションパスの組合わせに依存するため，構造を考慮せずにデータを分割した場合データマップのサイズは大きなものとなる．一方，構造のみでデータを分割した場合はデータマップのサイズは小さくなるが，分割された断片のサイズが一定とはならない．しかしながら，提案するデータ分割方法は，構造とデータサイズを考慮しているため，データマップのサイズは，データサイ

ズのみでデータを分割する方式よりも大きくはならないという利点がある．

3.4 分散問合せ処理

ユーザから QC に送られた問合せは，QC で解析処理を行い，問合せに関係する断片データを持つ QP に対して送信する必要がある．まずデータ配置の際に作成したデータマップから，問合せに関係する断片データのデータ id と $QPid$ を取得する．

QC は $QPid$ から問合せを送信する QP を認識し，該当する QP へ問合せとデータ id を送信する．問合せが複数の断片データに関係する場合は，異なるデータ id を持つ問合せを同時に複数送信する．

各 QP ではデータ id を基に問合せに関係する断片データを判断し，問合せ処理系を用いて問合せを評価し QC へ結果を送信する．QP から結果を受け取った QC は，その結果をユーザへ送信する．複数の断片データに対して同時に発行した問合せの場合は，各 QP から受け取った結果の統合処理が必要である．

3.5 データの動的な再配置処理

XML データの問合せ処理は，データサイズと問合せに依存するため，提案した分割アルゴリズムを用いてデータサイズが均等になるようなデータの分割及び配置を行ったとしても，問合せによっては，特定の QP に負荷が集中することが考えられる．この場合，負荷の高い QP の処理がボトルネックとなり，システム全体のパフォーマンスが低下してしまう．

そこで，一度配置した断片データを動的に再配置することで，負荷を分散し処理効率を上げる手法を提案する．本システムでは，各 QP の問合せ処理時間を計測し，一定時間毎に QP の中で最も処理実行時間が長い QP(LQP) と短い QP(SQP) を判断する．さらに，LQP に配置された断片データの中で最も問合せ実行時間が長いデータ (以下断片データ L と表記) と，SQP の中で最も実行時間の短い断片データ (以下断片データ S と表記) を交換することで，データの動的な再配置を実現する．

QC は，一定時間毎に各 QP と配置されている断片データの実行時間を取得し，断片データ L と断片データ S を決定する．次に断片データ L と断片データ S を持つ QP にメッセージを送り，二つの QP 間でデータの交換を行う．データ交換が成功すれば，各 QP は QC に対してメッセージを送り，QC は保持するデータマップを更新する．

4. XML 分散問合せ処理システムの評価

本節では，分散問合せ処理を評価するための予備実験と，3 節で提案した分散問合せ処理システムを構築して行った評価実験の結果から，得られた知見について述べる．

4.1 実験の目的

予備実験では，複数の計算機を用いて分散問合せ処理を行うことで，単一の計算機で処理する場合に比べ，処理時間がどの程度効率化されるのかを確認する目的で行う．また，データサイズ，データの分割数，計算機の数がどの程度問合せ処理時間に影響するかを評価する．

また本実験では，予備実験で明らかになったデータの動的な再配置処理の有効性を評価するため，提案システムに基づいた分散問合せ処理を行う．そして，データの動的な再配置処理を実行した場合と，実行しなかった場合とを比較することによって，実際に特定の QP に集中した負荷を分散できることを確認する目的で行う．

4.2 実験の準備

システム構築の際に使用した計算機は，NEC 社製並列計算機 (CPU: AMD Opteron 2.4GHz × 2，16GB のメモリ，73GB の Ultra320 SCSI HDD，64 ノード構成) であり，OS は Red Hat Enterprise Linux WS 3.0 である．構築した分散問合せ処理環境では，この並列計算機のうち，QC に 1 台，QP に 16 台の計 17 台の計算機を用いており，それらは互いにギガビットネットワークで繋がっている．

各計算機上で動作するプログラムは，Sun Microsystems の Java2 Platform Standard Edition Development Kit 5.0 Update 6 を使って作成し，問合せを評価する XML 問合せ処理系には XSQ version 1.0 [6] を用いた．そのため，本分散問合せ処理システムで扱うことができる問合せは，XPath による問合せ式であり，扱う XPath の範囲は (/，//，.，[ ]) である．また述語 [ ] を含む問合せは，述語の中に/や//を含まないものを使用し，join の操作が必要な問合せについては，本稿では扱っていない．

実験に使用したデータは，XMark プロジェクトで公開されている XML データ生成プログラム xmlgen によって，10MB, 100MB, 1GB(正確には 10.5MB, 105.2MB，1.05GB) の XML データを生成し，問合せには, 様々な状況を想定して自作した 24 種類の問合せ式を用いている．表 1[☆]に問合せ式を示す．

4.3 予 備 実 験

4.3.1 予備実験の結果と評価

表 2〜4 は，10MB, 100MB, 1GB の XML データに対して行った各問合せ処理時間の結果を，データの分割数 N ごとにまとめたものである．ここで表 2〜4 における問合せ処理時間とは，問合せが QC に入力

☆ 表 1 は 100MB のデータに対する問合せ式である．10MB, 1GB のデータに対する問合せでは，$Q11$〜$Q15$,$Q18$〜$Q22$ で属性部分の数値が変わる．

表 1 実験で用いた問合せ

	問合せ式		問合せ式
Q1	//date	Q13	/site/regions/namerica/item[@id="item10000"]//text
Q2	//name	Q14	/site/regions/namerica/item[@id="item13000"]//text
Q3	//seller	Q15	/site/regions/namerica/item[@id="item15000"]//text
Q4	/site/regions//item	Q16	/site/regions/samerica/item[quantity="1"]//mail
Q5	/site/regions/namerica/item	Q17	/site/categories/category/name
Q6	/site/open_auctions/open_auction/initial	Q18	/site/people/person[@id="person12000"]
Q7	/site/regions/europe/item	Q19	/site/open_auctions/open_auction[@id="open_auction35"]
Q8	/site/people/person/name	Q20	/site/open_auctions/open_auction[@id="open_auction3000"]
Q9	/site/regions/asia/item[quantity="1"]	Q21	/site/open_auctions/open_auction[@id="open_auction5600"]
Q10	/site/regions/australia/item[payment="Creditcard"]	Q22	/site/open_auctions/open_auction[@id="open_auction8300"]
Q11	/site/regions/europe/item[@id="item5000"]	Q23	/site/closed_auctions/closed_auction/price
Q12	/site/regions/europe/item[@id="item7500"]	Q24	/site/closed_auctions/closed_auction//text

表 2 分割数 N 毎の各問合せ処理時間 (秒) 10 メガバイト

	1	2	4	8	16
Q2	2.09	1.17	1.01	0.90	0.58
Q4	23.97	23.05	12.52	6.21	3.14
Q5	10.83	10.55	10.35	5.57	2.98
Q6	0.99	0.76	0.66	0.27	0.20
Q9	2.61	2.14	1.93	1.89	1.85
Q11	0.61	0.26	0.16	0.08	0.05
Q16	0.78	0.43	0.30	0.23	0.21
Q19	0.67	0.44	0.28	0.20	0.12

表 3 分割数 N 毎の各問合せ処理時間 (秒) 100 メガバイト

	1	2	4	8	16
Q2	20.33	10.88	9.16	7.94	4.25
Q4	224.92	225.15	120.09	61.71	29.32
Q5	107.65	103.65	101.84	56.57	29.27
Q6	9.39	6.90	5.10	2.54	1.31
Q9	24.06	20.54	19.99	18.62	18.36
Q11	6.20	2.44	1.21	0.70	0.31
Q16	8.83	5.13	4.00	3.34	3.10
Q19	6.03	3.73	1.95	1.17	0.49

されてから QP が全ての検索結果を QC に送信するまでの時間を指す．つまり，問合せ処理時間は，QC から QP への問合せの発行時間，XSQ による XML データへの問合せ処理時間，QP から QC へ結果データの送信が完了するまでの時間の三つからなる．

これらの表中の 1, 2, 4, 8, 16 は，3 章で説明したデータの分割数 N を表しており☆，以後，QP 数 N と表記した場合，N 個の断片データを QP1〜QPN に配置した環境のことを指す．

また，表 5 は 10MB, 100MB, 1GB の XML データに対して各問合せを発行した際に，検索結果として返される XML データのサイズを示したものである．ここで，表 2〜4 における問合せ処理時間とは，問合せが QC に入力されてから QP が全ての検索結果を QC に送信するまでの時間を指す．

表 2〜4 を比較すると，8 種類の問合せ全てにおいて，元データのデータサイズと処理時間には，比例関係がある．つまり，処理時間は扱うデータサイズに依存し，単一の計算機で問合せ処理を行うよりも処理効率は向上することがわかった．

4.3.2 予備実験の考察

前節で述べたように，どの問合せも QP 数が増えれば, ある程度速度は向上していることがわかる．その理由は，QP 数が増えることで，QP 一つあたりが問合せ処理を行う XML データサイズが小さくなることが挙げられる．

しかし，必ずしも配置されているデータサイズに比例して処理時間が短縮されているわけではない．例えば，表 3 に示す Q2 の処理時間を見てみると，QP 数 4 (問合せが発行される QP の数は 2) の処理時間は QP 数 1 の処理時間の 2.2 倍となっているが，QP 数 8 (問合せが発行される QP の数は 5) の処理時間は 2.6 倍となっており, QP 数が 2.5 倍に増えているにも関わらず，それに見合っただけの処理速度の向上は見られない．

この原因は，問合せ処理を実行する各 QP の処理時間を分析すると理解できる．Q2 を QP 数 8 で処理した場合，5 台の QP が使用されるが，図 4 が示すように QP5 の処理時間だけが極端に長いため，この部分がボトルネックとなっているからである．実際，Q2 の各 QP での処理は，QP3 → QP1 → QP2 → QP4 → QP5 の順に完了しており，QP4 までの処理は問合

表 4 分割数 N 毎の各問合せ処理時間 (秒) 1 ギガバイト

	1	2	4	8	16
Q2	191.02	109.85	88.10	79.06	43.26
Q4	2,242.01	2,212.58	1,146.19	605.83	288.19
Q5	1,062.86	1,034.33	977.58	549.68	279.32
Q6	92.17	70.24	52.57	26.76	14.19
Q9	240.67	202.63	185.96	182.98	183.40
Q11	60.20	22.97	11.02	6.31	2.96
Q16	86.29	49.18	36.78	31.40	29.04
Q19	59.69	37.01	18.31	12.11	4.39

表 5 問合せ結果データのサイズ (KB)

	10MB	100MB	1GB
Q2	138	1,388	13,843
Q4	5,538	54,757	547,622
Q5	2,543	25,241	254,461
Q6	29	294	2,945
Q9	476	4,621	46,299
Q11	3	3	2
Q16	46	726	6,959
Q19	2	6	3

☆ $N = 1$ はデータを分割せずに，一つの QP に配置して処理する場合をいう．

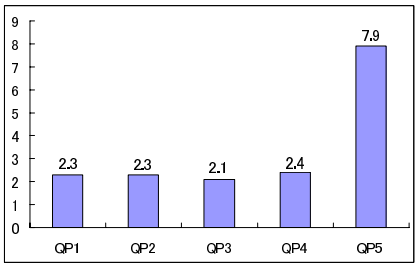

図 4 Q2 : QP 数 8 の問合せ処理時間 (秒)

せ処理開始後，約 2.4 秒ほどで完了しているが，QP5 の問合せ処理が 7.9 秒かかっている．つまり，複数の QP によって問合せ処理がされる場合は，最も処理の遅い QP の処理時間に依存するのである．QP5 の処理が遅くなる原因は問合せの検索結果のサイズが大きいためである．処理時間は問合せ処理の結果データサイズの大きさにも依存するため，このような問合せに対しては処理が遅くなる．

以上のことから，XML データに対する分散問合せ処理の有効性は確認できた．しかし，問合せによっては，分散処理に使用する計算機が増えても，その数に比例した処理速度の向上は望めない場合がある．そのため，問合せに関係するデータが多く配置されている QP は，処理に時間のかかる問合せが集中してしまい，並列処理の効率が悪くなる．したがって，問題の解決法として配置された分割 XML データの動的な再配置処理を考慮する必要がある．

4.4 本 実 験

4.4.1 実験の結果と評価

本節では，前節の予備実験により必要性が明らかとなった，データの動的な再配置処理の有効性を実験により評価する．比較のために，分割 XML データの再配置処理を実行しない場合と，再配置処理を実行した場合の問合せ処理時間を計測した．

データは 100MB の XML データを分割アルゴリズムを用いて，16 個の断片データ (データ 1〜16) に分割し，4 個の QP(QP1〜4) にそれぞれ配置した．このように複数の小さな断片データを各 QP に配置するのは，特定の断片へ負荷が集中した際に，データの再配置とともに負荷も同時に移動する現象を減らすためである．並列 GRACE ハッシュ結合演算においても同様の手法が利用されている [2].

なお，問合せは 24 個の問合せ式を，ランダムに 10 回，100 回繰り返し発生させた 240 個，2400 個の問合せセットを用いた．

図 5 と図 6 に表したグラフは，240 個の問合せセットを実行したときの，各 QP の問合せ処理時間を示す．図 5 は再配置処理を実行しない場合，図 6 は再配置処理を実行した場合のグラフである．グラフ中の点線は，各 QP の問合せ処理時間の平均値を表す．また，図 7 は 2400 個の問合せセットを実行し，再配置処理を行った場合のグラフである．なお，本評価実験の問合せ処理時間は，QP における XSQ の処理時間の合計をいう．さらに表 6 に，二つの問合せセットで実験を行った際の，各 QP で要した問合せ処理時間，四つの処理時間の平均値と標準偏差，及び実験開始から終了するまでにかかった総処理時間を示す．

表 6 各 QP の問合せ処理時間 (秒)

	240Query		2400Query	
	再配置なし	再配置あり	再配置なし	再配置あり
QP1	1,391	1,078	14,374	8,697
QP2	1,559	1,207	15,860	9,244
QP3	269	671	2,592	9,213
QP4	261	715	3,634	8,961
平均値	895	918	9,115	9,029
標準偏差	674	265	6,970	255
総処理時間	1,540	1,213	15,650	12,519

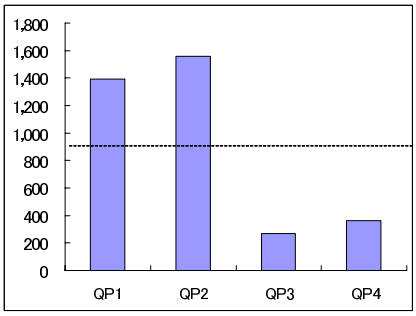

図 5 各 QP の問合せ処理時間：再配置なし (秒)

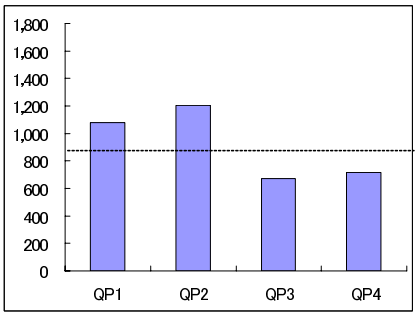

図 6 各 QP の問合せ処理時間：再配置あり (秒)

4.4.2 評価実験の考察

図 5 より，再配置処理を行わない場合は QP1 と QP2 の処理時間が長く，それらに負荷が集中していることがわかる．しかし，再配置処理を行った場合，図 6 より集中した負荷が他の QP へ分散していることが確認できる．また，2400 個の問合せセットで再配置処理を実行した場合は，図 7 から各 QP の負荷がほぼ均一になっていることがわかる．

一方，再配置処理を行った場合とそうでない場合の

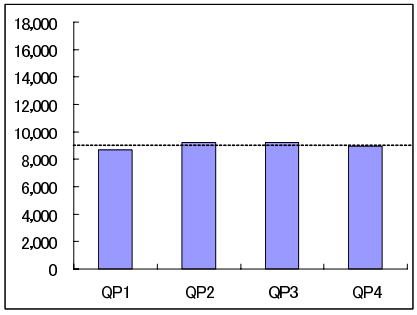

図 7 各 QP の問合せ処理時間 (2400Query)： 再配置あり (秒)

総処理時間を比較すると，表 6 より再配置処理を行ったほうが総処理時間が約 20%短縮されることがわかる．このことから，再配置処理を行うことで，特定の QP に集中した負荷を各 QP にほぼ均一に分散させることができ，問合せ処理効率の向上につながることが示せた．

5. おわりに

本稿では，大規模な XML データを効率良く処理するために，XML データを分割し，複数の計算機へ分散させて，問合せを処理する手法を提案した．本稿で提案した XML データの分割アルゴリズムと，問合せ処理時間を考慮した分割 XML データの動的な再配置法により，大規模 XML データに対する問合せ処理の効率化が実現することを評価実験から確認し，提案手法の有効性を示すことができた．

提案手法を XML 分散問合せ処理システムに適用すれば，データへのアクセス頻度が既知でない環境においても問合せ処理の効率化を図ることが可能である．また評価実験では，XML 問合せ処理系に XSQ を用いたが，Shunsaku ☆をはじめとする XPath をベースとした XML 問合せ処理系を用いれば，本手法は適用可能である．

本稿では，データサイズと構造を考慮したデータ分割法を提案したが，発行される問合せによっては，各計算機に配置するデータのサイズを均一にすることが，必ずしも負荷の均一化に繋がるとは限らない．したがって，サイズのみを考慮した分割方法や，構造のみを考慮した分割方法との比較を行い，提案アルゴリズムの有効性を評価する必要がある．また動的な再配置処理に関しても，処理を行うタイミング，交換するデータの判断基準などを改良し，更に効率のよい分散問合せ処理の実現を目指す予定である．

☆ http://interstage.fujitsu.com/jp/shunsaku/

謝辞 本研究の一部は，科学技術振興事業機構 戦略的基礎研究推進事業「情報社会を支える新しい高性能情報処理技術」プログラム，ならびに日本学術振興会科学研究費補助金 基盤研究 (A)(2) (課題番号: 15200010)，若手研究 (B) (課題番号: 16700103, 17700109)，および日本データベース学会・富士通株式会社共催「富士通 Shunsaku アカデミック支援プログラム」によるものである．ここに記して謝意を表す．

参考文献

1) Jan-Marco Bremer and Michael Gertz. On Distributing XML Repositories. In *Proc. of the 6th International Workshop on the Web and Databases*, pp. 73–78, June 2003.
2) Lilian Harada and Masaru Kitsuregawa. Dyanamic Join Product Skew Handling for Hash-Joins in Shared-Nothing Database Systems. In *Proc. of the 4th International Conf on Database Systems for Advanced Applications (DASFAA95)*, pp. 246–255, 1995.
3) Kien A. Hua and Chiang Lee. An Adaptive Data Placement Scheme for Parallel Database Computer Systems. In *Proc. of the 16th VLDB Conf.*, pp. 493–506, 1990.
4) International Organization for Standardization. Information processing – Text and office systems – Standard Generalized Markup Language (SGML). `http://www.iso.org/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=16387`. ISO 8879: 1986.
5) 中尾 伸章, 天笠 俊之, 的野 晃整, 植村 俊亮. アクセス頻度を考慮した XML 文書分割方式の提案. In *Proc. of Data Engineering Workshop*, February/March 2005.
6) Feng Peng and Sudarshan S. Chawathe. XSQ: A Streaming XPath Engine. *ACM Transactions on Database Systems*, Vol. 30, No. 2, pp. 577–623, June 2005.
7) World Wide Web Consortium. Document Object Model (DOM) Level 3 Core Specification. `http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407/`. W3C Recommendation 07 April 2004.
8) World Wide Web Consortium. Extensible Markup Language (XML) 1.0 (Third Edition). `http://www.w3.org/TR/2004/REC-xml-20040204/`. W3C Recommendation 04 February 2004.
9) 渡辺明嗣, 横田治夫. 分散ディレクトリコストを考慮した並列データアクセス制御. 電子情報通信学会論文誌 D-I, Vol. J85-D-I, No. 9, pp. 877–886, September 2002.