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

2007－DBS－143 (37)
2007／7／3

Remote Proxyを利用した並列分散XML問合せ処理手法の提案

油井 誠† 宮崎 純† 植村 俊亮†† 加藤 博一†

† 奈良先端科学技術大学院大学 情報科学研究科 〒630-0192 奈良県生駒市高山町 8916-5
†† 奈良産業大学 情報学部 情報学科 〒636-8503 奈良県生駒郡三郷町立野北 3 丁目 12-1
E-mail: †{makoto-y,miyazaki,uemura,kato}@is.naist.jp

あらまし 本稿では，分散XML問合せ処理におけるノード間のデータ交換に着目し，分散XML問合せ処理における問題と従来のデータベースにおける分散処理との違いを述べ，分散XML問合せを効率的に扱うためにRemote Proxyを利用した並列分散問合せ処理手法を提案する．これまで，分散XML問合せ処理の研究は値渡しによるデータ交換のみが行われてきたが，値渡しによるデータ交換は，シーケンスの一部の要素のみが利用される場合に無駄な通信が発生するといった問題や，オペレータ間の並列性が阻害されるという問題がある．我々の提案手法ではRemote Proxyを用いた参照渡しにより，データ交換を行う．そして，call-by-need形式でデータ交換を行うことで，不要なノード間通信と計算資源の過剰な消費を回避する．更に，我々の手法が並列の問合せ実行とその制御を容易に実現しうることを示す．

キーワード 並列・分散問合せ処理，XML問合せ処理，XQuery, 分散オブジェクト

A Scheme for Distributed and Parallel XML Query Processing using Remote Proxy

Makoto YUI†, Jun MIYAZAKI†, Shunsuke UEMURA††, and Hirokazu KATO†

† Graduate School of Information Science, NAIST Takayama 8916-5, Ikoma, Nara, 630-0192 Japan
†† Faculty of Informatics, Nara Sangyo University Misato Tateno-Kita 3-12-1, Ikoma-gun, Nara, 636-8503 Japan
E-mail: †{makoto-y,miyazaki,uemura,kato}@is.naist.jp

Abstract In this paper, we focus on one aspect of distributed XQuery processing which is data exchange between processor elements. We firstly address the problem of distributed XQuery processing, and explain how the problem differs from traditional database problem. For the efficient XML query processing, we propose a scheme for distributed and parallel query processing which adopts the use of *Remote Proxy*. Previously proposed methods, which use *pass-by-value* semantics, have often suffered from redundant communication between processor elements and limited inter-operator parallelism. Our scheme makes the use of *pass-by-reference* semantics via *Remote Proxy* for data exchanges. Then, the network traffic and buffer occupation by exchanging data are reduced due to the fruit of *call-by-need* semantics. Moreover, our scheme naturally allows parallel query execution.

Key words Distributed and Parallel query processing, XML Query Processing, XQuery, Distributed Object

1. はじめに

XMLが組織間のデータ交換形式として標準の地位を確立したことで，グリッド・コンピューティングからWeb上のデータ処理まで，ネットワーク上に分散して存在する複数のXMLデータを統合する需要が増えている．例えば，塩基配列[1]やタンパク質データベース[2]といったバイオデータベースやその検索システム[3]は，データの相互利用のためにデータをXMLとして出版する機能を有する．また，Webコンテンツの更新情報の概要を配信するRSSやATOMデータは，Web空間上で飛躍的に増加している．このように情報交換にはXMLが頻繁に利用されるため，ネットワーク上に分散して存在するXMLデータを効率的に統合することは重要である．本稿では，データの供給者から与えられたXMLデータが主情報源となる場合を議論する．ここで，利用者はデータ供給者の所有する情報源に対して限定されたアクセス権限を持つユーザと想定する．

従来の分散データベースの研究では，複数の計算ノードにデータを分配し，計算ノード間で計算負荷を分散させることで，問合せを効率的に処理する研究が行われてきた[4]．本稿で扱うXMLデータの分散という問題が，従来の分散データベース処理の問題と異なる点として，予めデータベースにデータを読み込み，索引を付加するといった前処理が必ずしも適用できな

いことにある．供給者の情報源が常に更新されるような状況では，供給者から配布される XML データの内容は頻繁な更新が行われることなる．そこでは，常に最新の情報にアクセスすることが重要となる．こうした状況の顕著な例としては，ニュースサイトの RSS 購読が挙げられる．現在の RSS リーダは一定間隔で購読サイトにアクセスすることで利用者が購読する情報を更新しているが，自動収集された数時間前のニュースではなく，最も最近発信されたニュース (例えば災害情報やスポーツの試合結果) を確認したいという場合がある．

大量の XML データに対して実時間で問合せ処理するという問題を考えたとき，単一の計算ノードで XML データの統合処理を行うことは，現在の計算機のプロセッサ，記憶域，帯域幅をもってしても現実的とは言えない．そこで，本稿では複数の計算機を用いて XML 問合せ処理を行う分散 XML 問合せ処理について議論する．複数の計算ノードで分割統治を実施することによって，ネットワーク上に分散した XML データに対して実時間の問合せ処理が現実的となる．複数の XML 情報源の統合 (結合や集約処理) を扱う上での XML 問合せ言語としては，高い表現能力を持つ XQuery [5] を用いる．

本稿では，分散 XQuery 処理において基礎技術となる計算ノード間のデータ交換に焦点を当てる．分散 XQuery 問合せ処理の研究は *XQueryD* [6] を始めとして既に幾つかなされてきたが，既存研究では，計算ノード間のデータ交換に値渡しのみが行われてきた．値渡しによるデータ交換は，巨大な部分木の一部分だけをリモートの計算ノードが利用する場合やリモートサイトの実行結果の一部の要素のみが必要とされる場合などに必ずしも効率的ではない．そこで，*Remote Proxy* を用いた参照渡しによるデータ交換手法を提案する．クライアントが，*call-by-need* でサーバに部分結果を要求することで，不要なノード間通信とバッファの消費を回避する．更に，我々の手法を用いることで並列の問合せ実行とその制御を容易に実現しうることを示す．

2. 分散 XQuery 処理

2.1 Data Shipping と Query Shipping

XQuery を用いた分散 XML 問合せ処理を実現する上でまず問題となるのは，XQuery の言語仕様 [5] では，*fn:collection* 関数か *fn:doc* 関数を通じた *Data Shipping* 実行モデルのみがサポートされている点である．*Data Shipping* では，全てのデータソースが問合せを発行した計算ノードに収集されて，その計算ノードで全ての問合せ処理が実行される．図 1(a) が *Data Shipping* を行う問合せの例である．この XQuery 問合せは，ローカルサーバのファイルシステム上に存在する X.xml とリモートサーバに存在する Y.xml という 2 つの XML 文書間の Join を行う問合せである．このとき，リモートサーバから収集されるデータ (Y.xml) が巨大であるが，ローカルサーバで利用するのは小さな部分木である場合，通信コストや計算ノードが消費する記憶域の増加という形で問合せ処理性能に深刻な影響が生じる．このような事態を回避するためには，*Data Shipping* に加えて，*Query Shipping* 実行モデルをサポートする必要がある．

我々の提案手法において図 1(a) の問合せを *Query Shipping* で処理するように変更したのが，図 1(b) である．リモート問合せを行うのが，*remote-eval* 関数である．関数の引数は (1) リモートエンドポイント (xs:string 型)，(2) リモートサイトで実行する問合せ式 (xs:string 型)，(3) リモートの呼び出し先で用いるパラメタ (item()*型) である．ここでは，パラメタ (3) の $varsToShip に，リモートサイトで利用されるローカルサイトの変数$x を束縛している．

```
for $x in fn:doc("file://X.xml")/a/b[@c="123"]
let $y :=
  fn:doc("http://Y.com/Y.xml")(α)/d/e[@f=$x/@g]/h
return
  <answer> <x> { $x/u } </x> <y> { $y/v } </y> </answer>
```

(a) Data Shipping

```
for $x in fn:doc("X.xml")/a/b[@c="123"] (γ)
let $varsToShip := $x,
    $y := remote-eval (
 (1) "http://Y.com",
 (2) "for $z
      in fn:doc('Y.xml')/d/e[@f=$z/@g(β)]
      return $z",
 (3) $varsToShip
    })/h
return <answer><x>{ $x/u }</x><y>{ $y/v }</y></answer>
```

(b) Query Shipping

図 1 問合せ例

XQueryD [6] や *XRPC* [7] では，リモート問合せのために構文糖衣を用意しているが，提案システムにおける 3 点のパラメタと同様のパラメタによって，リモート関数呼び出しが行なわれるという点で同一である．なお，パラメタ (2) については，文字列の代わりにコンパイル済み式を用いる方式 [8] や，リモート問合せを関数呼び出しに制限し，リモート呼び出し元にスケルトンを，リモート呼び出し先にスタブを用意する方式 [7] (注1) がある．

2.2 分散 XQuery 処理の問題点と対策

本節では，分散 XQuery 処理を扱う上で，考慮に入れるべき問題を挙げる．既存手法を紹介すると共に，問題に対する我々のアプローチを述べる．

i). Dependent Joins

分散 XQuery 処理の既知の問題の一つに，*Dependent Joins* [6] がある．*Dependent Joins* の一例が，図 1(b) の下線部 (β) である．図 1(b) のリモート問合せはローカルサイトの変数$x に依存する．そのため，単純な問合せ処理を行った場合，(γ) のループが実行されるごとにリモート問合せが実行される．このことは，リモート問合せの実行回数が多くなる場合に，大きな性能低下として現れる．*XQueryD* [6] ではこの問題に対処するために，文字列置換により等価な *Dependent Joins* を含まない式に書き換える手法を提案している．この文字列置換による等価変換が行われるのは，書き換え対象式の変数が文字列である場合に限られており，実用性に乏しい．*XRPC* [7] では，*Dependent Joins* を効率的に処理するために，複数回のリモート問合せを取りまとめて一度のリモート関数呼び出しとする手法 *BulkRPC* を提唱している．*XRPC* では値渡しによってノード間のデータ交換が行われる．

我々は，*Dependant Joins* の問題を認識しているが，現在のところ，この問題への対処はユーザの手による問合せの等価変換に頼っている．その理由として，これまでに提案されてきた手法 [7] には，オペレータ間の並列性 (*Inter-operator parallelism*) の問題が存在することがある．ここでいうオペレータ間の並列性の問題とは，一括したリモート関数呼び出しを行う場合，呼び出し元で値渡しするデータの計算を全て行ってからリモート呼び出しを行うこととなり，ローカルサイトでの計算の間，リモート呼び出しがブロックされるという問題を指す．図 4 に，その一例として，値渡しを利用したリモート問合せの処理シー

(注1)：XRPC では，同一のモジュールを予め呼び出し先と呼び出し元に用意する．

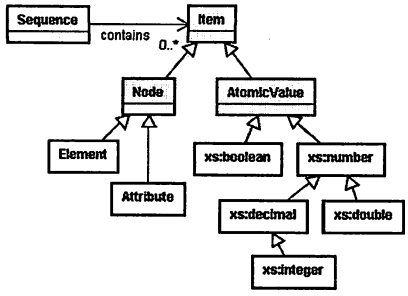


図 2 XQuery のデータモデル (Node と AtomicValue の子要素は一部省略)

```
for each left in leftOperand do
  for each right in rightOperand do
    if( left θ right ) then
      return true
return false
```

図 3 Generic Comparator のセマンティクス

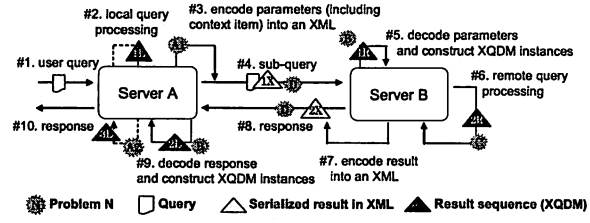


図 4 XML 形式による値渡しを利用したリモート問合せ

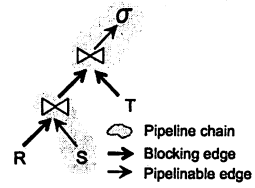


図 5 3 つのサイト間の Join

ケンスを示す．図 4 では，#4 でリモート問合せを発行しているが，そのためには，予め#2 の計算でパラメタの値 1L を具体化する必要がある．ここで，リモート問合せの発行がブロックされる (問題 A1 の箇所)．さらに，#2 の計算は Server B でのリモート問合せ結果が全て実行し終えるまでブロックされる (問題 A2 の箇所)．

こうした問題に対処するため，本稿では，オペレータ間の並列性の問題に焦点を当て，これを改善する提案を行う．

ii)．**冗長なノード間通信**

これまで，分散 XQuery 問合せ処理の研究は値渡しによるデータ交換のみが行われてきた [6] [7] [8]．値渡しによるデータ交換は，リモートの処理結果シーケンスの一部分だけが利用される場合に必ずしも効率的でない．XQuery では，シーケンス (シーケンスは複数の Item を含む (図 2) 参照) の一部の Item しか利用されない状況が生じる．リモート問合せが Generic Comparator を含み，その rightOperand がローカルサイトの変数という状況を考える (後述する図 10(α) の箇所がその実例)．Generic Comparator のセマンティクス (図 3) では，rightOperand に含まれる Item は必ずしも全て利用されるわけではない．Zhang らは論文 [7] で，参照渡しではなく値渡しを利用する理由として，parent::*の呼び出しが，二次的に細粒度のサーバ – クライアント間の細粒度の通信を引き起こす可能性があるという例を挙げて，参照渡しを実装することの複雑性を挙げている．参照渡しの通信コストは高レベルのノード間通信を行うことによって下げることができるため，この指摘は，常に正しいとは限らない．リモートオブジェクトにアクセスする際に，高レベルの通信を行うというのは，分散オブジェクト技術で培われたベストプラクティスの一つである．値渡しによるデータ交換は，シーケンスの一部分だけが利用される場合等で必ずしも効率的ではない．我々の提案手法では値渡しに加えて，Remote Proxy を用いた参照渡しによるデータ交換をサポートし，参照渡しによるデータ交換を標準として採用している．

iii)．**ノード間の情報伝送形式**

既存の分散 XQuery 問合せ処理において，計算ノード間でのデータ交換には，内部データモデルを XML として符号化して，その XML データを伝送形式に用いていた [7] [8]．図 4 に，XML 形式による値渡しを利用したリモート問合せの処理シーケンスの一例を示す．ここで，#5 と#9 の箇所で，値渡しされた XML データのデコード処理 (XML のパース処理と XQuery Data Model(*XQDM*) [9] インスタンスの生成) が行われる．XML のパース処理 (文字列解析と解析イベントの生成) は軽い処理ではない．そのため，特に XML データが大きい場合に，XML のパース処理 (文字列解析と解析イベントの生成) が性能劣化 (図 4 の問題 B) を引き起こすことが予想される．

提案システムでは，SAX イベントに *XQDM* 特有のイベント (AtomicValue やノード情報の開始終了イベントなど (図 2 参照) を追加し，そのイベント列をノード間の情報伝送形式に用いることで，従来，文字列解析にかかっていたコストを抑えている．(XML) テキスト形式ではなくバイナリ形式を伝送形式に用いることは，ノード間通信量の削減，つまり問題 *ii* の改善にも繋がる．

iv)．**問合せの並列処理**

分散 XML 問合せ処理の研究は幾つかあるが，多くはオペレータ中の並列性 (*Intra-operator parallelism*) を議論したものであり，オペレータ間の並列性 (*Inter-operator parallelism*) を深く論じた研究は存在しない．*XRPC*[7] は，リモート問合せを並列に発行することで，リモートの計算ノード間の並列実行を行うとしている．しかし，問合せを統合するノードでの計算は，リモートノードでの処理結果が終了されるまでブロックされる (図 4 においては問題 A1，A2 の箇所)．

図 5 を用いて，3 つの計算ノード間でデータの統合を行う場合を考える．ここで，R と S はリモートノードの計算結果，T はローカルノードでの中間計算結果とし，ローカルノードで R と S，T を統合するものとする．図 5 は，*Blocking Edge* がもたらすパイプライン処理可能な連鎖への影響を示している．既存の XML 問合せ処理の研究では，ノード間での計算結果を値渡しで交換する．そのため，リモートノードの計算から統合ノードへの枝は全て *Blocking Edge* となり，このことがオペレータ間の並列性を下げていた．

計算結果のノード間での交換に，*Remote Proxy* を我々のアプローチでは，リモートノードの計算から統合ノードへの枝を全て，パイプライン可能な枝 *Pipelinable Edge* とすることで，オペレータ間の並列性を高めている．

v)．**特定の問合せによる計算資源の占有**

問合せの並列実行では，同一のリソースへの同時アクセスが相互干渉し，性能の劣化が起こる．複数のクライアントからの問合せ処理要求が同時に発生する場合など，複数の問合せ処理を処理する上では，それぞれの問合せ処理要求によって生じる計

算資源 (プロセッサ，記憶域，帯域幅など) の占有を抑えることが重要となる．オペレータ間のロードバランスを行う手法は，プロセッサの割り当てを効率的に行うことを目的として，並列データベースの分野で研究されてきた．現在では，廉価な PC にも 2 コア CPU が搭載されることが一般的となってきており，更に多いコアを搭載した CPU の登場が予定されていることを鑑みると，マルチプロセッサを搭載した並列データベース分野だけでなく，PC を利用したデータベース処理においても，豊富な CPU リソースを有効に活用するためにはオペレータレベルでの効率的なプロセッサを割り当てが重要となる可能性がある．

オペレータ間のリソース割り当てを効率的に行う研究と代表的なものに，RateMatch アルゴリズム [10] がある．RateMatch では，消費されるタプルと生産されるタプル数に応じてプロセッサの割り当てが行われる．例えば，生産数が消費数に比べて多ければ，過剰な生産によってリソース (CPU やメモリ) が過度に消費されるし，あるオペレータによって消費されるタプル数が割り当てられた CPU リソースによる消費可能なタプル数に比べて少なければ，その CPU リソースは idle 状態となり無駄となる．そこで，生産数と消費数を考慮した動的なリソース割り当て手法を提案している．

オペレータ間のリソース割り当てを効率的に行うことは，マルチプロセッサ環境における問合せの並列実行に限らず，分散問合せ処理の並列実行にも重要である．値渡しによる生産を行った場合，一度に大量のデータが送受信される (図 4 における D の箇所) ことで，デコード処理 (図 4 における B の箇所) で多くのリソース (CPU やメモリ) が消費されるため，他の計算 (例えば，図 4 における C の箇所や他の問合せ処理) に必要なリソースを圧迫する可能性がある．

RateMatch アルゴリズムは，問合せプラン作成時に各オペレータの並列可用性を意識し，コスト計算を行うことによって計算資源の割り当てを行う手法である．我々の提案手法では，*Blocking Queue* [11] を利用した消費数に応じた受注生産方式を採用することで，計算資源の割り当てをよりシンプルに実現している．*Blocking Queue* とは，要素の取得時にキューが空でなくなるまで待機したり，要素の格納時にキュー内に空きが生じるまで待機するキューである．図 6 を用いて，一例を挙げて説明する．まず，指定の要素数に達しない場合 (#1) は，通常のキュー操作と同一である．指定の要素数に達した場合 (#2)，enqueue 処理が dequeue 処理が行われるまでブロックされる (#3)．この dequeue 操作によって enqueue 処理が再開され，ブロックされていた要素がキューに投入される (#4)．

2.3 RemoteProxy を利用した分散問合せ処理

図 7 に *RemoteProxy* を利用した分散問合せの処理シーケンスを示す．我々のシステムは，クライアントへの問合せ結果の応答方式として，Response，Callback，Polling の 3 つをサポートするが，ここでは応答方式が Response の場合を議論する．尚，非同期のメッセージに依存する部分の実行順序は不定であるため，ここでは実行形態の一例を用いて説明する．

a) クライアントノードは問合せをリモートノードに発行する．

b) 問合せを受けたリモートノードは，スレッドプール中のクエリ処理スレッドを割り当てる．

c) 応答方式が Response であれば，クエリ処理スレッドからの返答があるまでスレッドの実行を停止する．

d) 問合せを実行し，結果シーケンスを得る．問合せの実行では，Volcano イテレータ実行モデル [12] に基づく．Volcano イテレータモデルでは，個々のオペレータは標準的なイテレータのインタフェースを備える．結果シーケンスは Volcano イテレータモデルにおけるオペレータ木であり，結果シーケンスの取得は即座に行われる．

e) 結果シーケンスを，RemoteProxy [13] で包む．ここで，

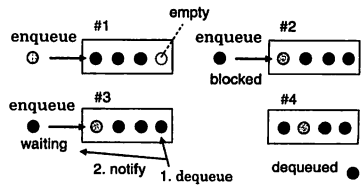


図 6 要素数 4 の BlockingQueue

RemoteProxy とは実体である結果シーケンスへのプロキシとして動作するオブジェクトである．通常のプロキシと異なるのは，リモートの計算機にそのオブジェクトを移しても，同様に実体にアクセス可能なことである．

f) RemotePeer スレッドを再開し，リモートプロキシを返す．

g) リモートプロキシを直列化し，クライアントに問合せの実行結果として返す．ここで，問合せの実行結果の実体が全て計算が終えているとは限らない．

h) 問合せの実行結果の実体を計算するスレッド (Parallel Producer) を起動したら，Query Processor スレッドをスレッドプールに戻す．

i) 結果シーケンスの実体たる Item(図 2 参照) を生産する．ここで，Parallel Producer によって生産される Item の量は，*Blocking Queue* の容量 (デフォルトでは 2048 個の Item) までとする．容量を超える Item の生産を行った場合，そのスレッドは Item が消費されて *Blocking Queue* に空きスペースが生じるまでブロックされる．この仕組みにより，過剰な生産を抑え，計算資源が無駄に消費されることを回避している．尚，生産が終わった際には番兵をキューに入れる．

j) クライアントが，g で受け取ったリモートシーケンスへのプロキシ (*RemoteProxySequence*) にアクセスし，そこにローカルノードの *RemoteProxySequence* に未消費の Item が存在しない場合，k の高レベルの通信を行う．ここで高レベルの通信とは，複数回のリモートオブジェクトへのアクセスを一度の RPC で行うことを指す．ノード間通信の詳細は，図 8 を用いて別途説明する．

k) プロキシを通じてリモートの実体から一定の Item を取得する．ここで，一定のアイテムとは，ユーザが問合せ発行時に指定する一度に取得する Item 数と fetchSize の伸張係数に依存する．特に指定しない場合，取得 Item 数は 256 で伸張係数は 3 割増である．つまり，k の処理が行われるごとに，取得する Item 数を 256, 332, 431... と増加させていく．

l) 結果シーケンスの実体にアクセスして，Item を取得する．ここで，これ以上の生産がないことは番兵により検知する．番兵に達していないが，*Blocking Queue* にアイテムがない場合は，Item が i により生産されるまで待機する．

ここで鍵となるのは，i) と l) 間で生産量の調整が行われることと，リモートプロキシを利用することでリモートの実体シーケンスへの透過的なアクセス j~l が行われることである．

リモートプロキシを介した実体シーケンスへのアクセス

リモートプロキシを利用したリモートの実体シーケンスへのアクセスの仕組みを示したのが，図 8 である．オブジェクト通信のレイヤは，機能層 (Functional layer)，論理層 (Logical layer)，物理層 (Physical layer) の三層から構成される．機能層は，通常のオブジェクトへのメッセージ通信をエミュレートする．論理層では，プロキシオブジェクトを介したサーバ・クライアント間での高レベルの通信 (L2) を扱う．論理層を設けているのは，RPC による潜在的な性能劣化を防ぐ目的で，機能層の細粒度の呼び出し (L1) をまとめ，疎粒度の呼び出し (P1)

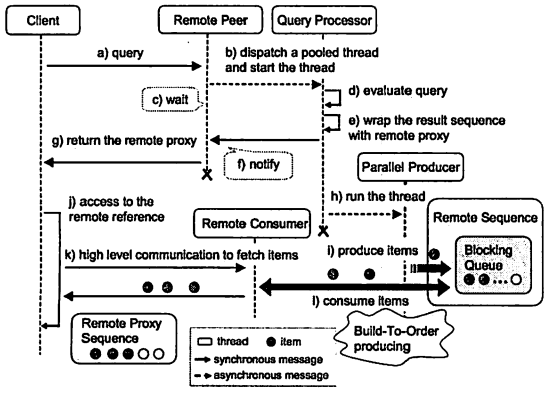


図 7 リモート問合せ処理の実行シーケンス

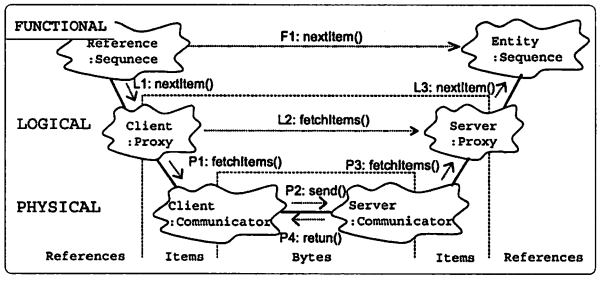


図 8 ノード間通信のレイヤ

とするためである．物理層は，下位の通信メカニズムに基づいてノード間での通信を行う．F1 は機能レベルでみたリモートの実体シーケンスへのアクセスを示し，L2 は論理レベルでみたリモートの実体シーケンスへのアクセスを示し，P2, P4 は物理レベルでのノード間通信を示す．

3. 応 用 例

XML による情報統合が必要とされている分野の一つにバイオインフォマティクス分野がある [14]．本節では，図 9 に我々が取り組んでいるバイオデータベースの情報統合システムの概要と，図 10 にその問合せの一例を示す．

システムを利用する流れとしては次の通りである．1). データベース管理者が，予め，Uniprot と GenBank というデータベースインスタンスに，それぞれ Uniprot [2] と GenBank [1] より取得した XML データを取り込む．2). ユーザが Blast 検索の結果を XML として出力し，Blast-DB データベースインスタンスに登録する．3). ユーザはその Blast 検索結果を用いて，データベース統合を行う XQuery 問合せを Blast-DB インスタンスに対して発行する．Blast-DB インスタンスが，複数のデータベース (Uniprot, Genbank, Blast-DB) の情報統合を行うユーザ問合せを実行する．

現在，複数の生物学データベースの統合のために，生物学の研究者は Web サービスや汎用プログラミング言語のライブラリを利用して手続き的にプログラムを組んでいるが，柔軟性のあるデータの利活用やデータの効率的な利用のためには複数のライブラリやネットワーク・プログラミングの知識が必要とされる．我々のプロジェクトでは，ユーザが Blast の検索結果の XML 構造と Uniprot, Genbank が出力する XML 構造を認識すれば，柔軟な情報統合を統一的なデータアクセス手段で実現

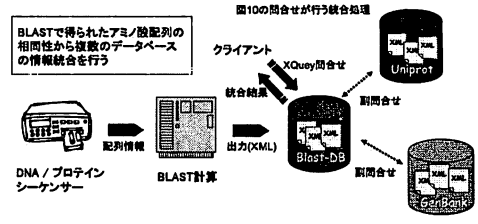


図 9 複数のバイオデータベースの情報統合

```
(: local database :)
declare variable $blast-gb := fn:collection("/crest/blast-gb.xml")/result;
declare variable $blast-uniprot := fn:collection("/crest/blast-uniprot.xml")/result;
(: remote basebase :)
declare variable $remote-endpoint := "//genbank:1099/xbird/srv-01";

let $gb_entries :=
  (for $e in $blast-gb/BlastOutput
   let $hit := $e/BlastOutput_iterations/Iteration/Iteration_hits/Hit
   where $hit/Hit_num = 1 and $hit/Hit_hsps/Hsp/Hsp_evalue/text() <= 1.0e-8
   return $e),
   $uniprot_entries :=
  (for $e in $blast-uniprot/BlastOutput
   let $hit := $e/BlastOutput_iterations/Iteration/Iteration_hits/Hit
   where $hit/Hit_num = 1 and $hit/Hit_hsps/Hsp/Hsp_evalue/text() <= 1.0e-8
   return $e)
where $gb_entries/BlastOutput_query-def/text()
    = $uniprot_entries/BlastOutput_query-def/text()
return
  <result>
  {
   let $gb-hit := $gb_entries/BlastOutput_iterations/Iteration/Iteration_hits/Hit,
       $gb-key := $gb-hit/Hit_gb/text()
   return
    let $genbank-remote-entries := xbird:remote-eval(
      $remote-endpoint,
        'fn:collection("/crest/genbank.xml")/genbank/INSDSeq
                        [INSDSeq_accession-version/text() = $gb-key(α)]',
      $gb-key)
    return
     for $remote_entry at $pos in $genbank-remote-entries
     let $quals := $remote_entry/INSDSeq_feature-table/INSDFeature/INSDFeature_quals
     where $quals/INSDQualifier/INSDQualifier_name/text() = "gene"
     return <genbank num="{ $pos }">{ $quals } </genbank>
  }
  </result>
```

図 10 Blast 検索結果と GenBank の該当エントリの統合

できる機能を提供している．

3.1 性能評価

提案手法の有効性を検証する目的の予備実験として，データ交換手法に提案手法である *RemoteProxy* を用いた参照渡しを用いた場合と値渡しを用いた場合の比較を行った．参照渡しを用いる場合については，2.3 節で解説した *BlockingQueue* を用いる非同期方式の他に，リモートの実体シーケンスの計算を図 7 の 1 の操作に応じて同期的に行う同期方式を検討材料として用いた．

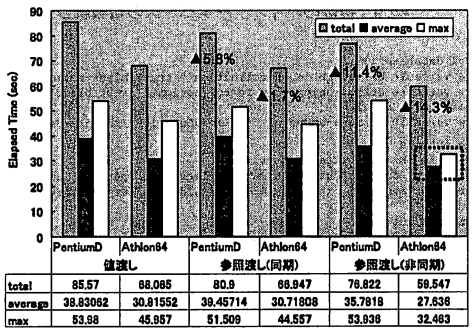
実験には図 10 の問合せを利用した．図 10 で参照するデータファイルのサイズは，genbank.xml が約 1M バイト，blast-gb.xml が約 44M バイト，blast-uniprot.xml が約 39M バイトである．実験環境は表 1 の通りで，図 10 における GenBank インスタンスに PentiumD を用いた．Blast-DB インスタンスには，PentiumD を用いた場合と比較的高性能な 64 ビットマシンである Athlon64 を用いた場合の 2 つのケースをテストした．実験方法としては，上記した組合せについて，図 10 の問合せを連続 50 回発行し，クライアント (Blast-DB インスタンス) 上で同時実行される問合せ処理のスレッド数を 25 として性能差を測定した．

実験結果が，図 11 の通りである．図 11 の total, average, max は，50 個の問合せを処理するのに要した処理時間，個々の問合せの実行時間の平均処理時間，最も時間を有した問合せの処理時間を示す．

今回の実験では，非同期方式の参照渡しを用いた場合が最も良い性能を示し，値渡しを用いた場合に比べて 1 割以上の性能

表 1 実験環境

マシン名	PentiumD	Athlon64
CPU	PentiumD 2.8GHz	Athlon64 X2 2.4GHz
コア数	2	2
メモリ	2G バイト	2G バイト
ネットワーク	1000Base-T	1000Base-T
OS	Linux Kernel 2.6.18	Linux Kernel 2.6.18
JavaVM	Sun JDK 1.6	Sun JDK 1.6



	PentiumD 値渡し	Athlon64 値渡し	PentiumD 参照渡し(同期)	Athlon64 参照渡し(同期)	PentiumD 参照渡し(非同期)	Athlon64 参照渡し(非同期)
total	85.57	68.085	80.9	66.947	76.822	59.547
average	38.83062	30.81552	39.45714	30.71808	35.7818	27.638
max	53.98	45.957	51.509	44.557	53.936	32.463

図 11 複数のデータ交換方式による性能差

向上を示した (▲が値渡しを利用した場合に対する総処理時間の向上を示す). 特筆すべき点としては, Athlon64 を用いた比較的 CPU リソースが充実した環境において, 非同期の参照渡しによる性能の向上が顕著に見られたことと, 非同期の参照渡しでは問合せ間のスループットの差が最も小さかった (標準偏差 2.25 秒) ことがある. この理由としては, 分散ノード間でオペレータ間の並列性を保てたことで, CPU リソースが有効が活用されたことが理由として考えられる. 高性能でコアを複数有するプロセッサを利用した場合に, オペレータ間の並列性の高い提案手法が有効に働くことを確認した.

4. 関連研究

Ré らは, XQuery への分散問合せ処理の拡張である *XQueryD* [6] を提案している. そして, *Dependant Join* に対処するために, 問合せの文字列書き換えによる等価変換を提案している. Zhang らは, P2P を利用した XML データベース環境の実現のための土台として, 分散 XQuery 問合せ処理機 *XRPC*[7] を提案している. *XRPC* は複数のリモート関数呼び出し (RPC) を一つにまとめて, *set-at-a-time* 形式で一度の呼び出しとすることによる, RPC の潜在的な通信オーバヘッドを削減する手法 (*BulkRPC*) を提案している. これらの手法では, データ交換は全て値渡しで行われるが, 我々は値渡しに加えて参照渡しをサポートしている.

Sahuguet らは, 連結方式, 訪問方式, query/data/hybrid shipping などの複数の分散問合せ処理パターンを紹介し, XQuery プロセッサを下位コンポーネントとして利用した分散問合せ処理言語を提案している [15]. Papadimos らは, *Mutant query plans*(MQPs) という訪問方式の一種に分類される問合せ処理手法を提案している. 我々の採用する *RemoteProxy* を問合せ処理は, 既存の分散問合せ処理パターンとは異なり, これまでには実現されてこなかったオペレータ間の並列実行をサポートする.

5. まとめ

本稿では, 分散 XML 問合せにおけるノード間のデータ交換に着目し, RemoteProxy を利用した分散 XML 問合せ処理手法を提案した. 提案手法の利点として, 実体シーケンスの Item を受注生産することによって, マルチクライアント環境における計算資源の占有を防ぐことがある. また, リモート問合せとローカル問合せのオペレータ間のパイプライン処理が可能であることから, 分散問合せ処理におけるオペレータ間の高い並列性を容易に実現しうることを示した.

今後の課題として, 分散 XML 問合せ処理の性能評価のためのベンチマークの作成とその実施, リモートサイトへの演算プッシュダウンなどの分散問合せ最適化の実装が挙げられる.

謝辞

本研究の一部は, 科学技術振興機構戦略的創造研究推進事業 (CREST)「情報社会を支える新しい高性能情報処理技術」ならびに, 文部科学省科学研究費補助金特定領域研究「情報爆発時代に向けた新しい IT 基盤技術の研究」, 公募研究「偏在する大規模構造化文書からの高度情報抽出」(研究代表者：宮崎純, A01-34, 課題番号 19024058) による. ここに記して謝意を表します.

文献

[1] National Center for Biotechnology Information (NCBI). Genbank. http://www.ncbi.nlm.nih.gov/Genbank/.

[2] The Universal Protein Resource (UniProt). http://www.uniprot.org/.

[3] National Center for Biotechnology Information (NCBI). BLAST: Basic Local Alignment and Search Tool. http://www.ncbi.nlm.nih.gov/BLAST/.

[4] Tamer M. Ozsu and Patrick Valduriez. *Principles of Distributed Database Systems (2nd Edition)*. Prentice Hall, 1999.

[5] World Wide Web Consortium. XQuery 1.0: An XML Query Language. http://www.w3.org/TR/xquery/.

[6] Christopher Ré, Jim Brinkley, Kevin Hinshaw, and Dan Suciu. Distributed xquery. In *Proc. Workshop on Information Integration on the Web (IIWeb)*, pages 116–121, 2004.

[7] Y. Zhang and P. A. Boncz. Integrating XQuery and P2P in MonetDB/XQuery*. In *Proc. Workshop on Emerging Research Opportunities for Web Data Management (EROW)*, 2007.

[8] Mary Fernández, Trevor Jim, Kristi Morton, Nicola Onose, and Jerome Simeon. Highly Distributed XQuery with DXQ. In *Proc. SIGMOD conference*, 2007.

[9] World Wide Web Consortium. XQuery 1.0 and XPath 2.0 Data Model (XDM). http://www.w3.org/TR/xpath-datamodel/.

[10] Manish Mehta and David J. Dewitt. Managing intra-operator parallelism in parallel database systems. In *Proc. VLDB*, pages 382–394, 1995.

[11] Doug Lea. The java.util.concurrent synchronizer framework. *Sci. Comput. Program.*, 58(3):293–309, December 2005.

[12] G. Graefe. Volcano-an extensible and parallel query evaluation system. In *IEEE Trans. Knowledge and Data Engineering*, volume 6, pages 120–135, 1994.

[13] Hans Rohnert. *The proxy design pattern revisited: Pattern languages of program design 2*. Addison-Wesley, Inc., 1996.

[14] F. Achard, G. Vaysseix, and E. Barillot. Xml, bioinformatics and data integration. *Bioinformatics*, 17(2):115–125, February 2001.

[15] Arnaud Sahuguet and Val Tannen. ubql, a language for programming distributed query systems. In *Proc. WebDB*, pages 37–42, 2001.