

遅延評価を利用した並列分散 XQuery 問合せ処理

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 RemoteProxy を利用した参照渡しによる並列分散 XQuery 問合せ処理手法を提案する。これまで、分散 XML 問合せ処理には値渡しによるデータ交換が行われてきたが、値渡しによるデータ交換にはシーケンスの一部の要素のみが利用される場合に不要な通信が発生する問題や、オペレータ間の並列性が阻害されるという問題がある。提案手法ではこの問題への解決策として、遅延評価を活かした分散問合せ処理手法を開発した。評価実験において、提案手法が競合手法に対し最大 22 倍の性能向上を得られることを明らかにし、分散 XML データベースシステムが参照渡し戦略を考慮に入れることの重要性を実証する。

キーワード 並列分散データベース, XML 問合せ処理, 遅延評価

Distributed and Parallel XQuery Processing using Lazy Evaluation

Makoto YUI[†], Jun MIYAZAKI[†], Shunsuke UEMURA^{††}, and Hirokazu KATO[†]

[†] Graduate School of Information Science, Nara Institute of Science and Technology
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 We propose a scheme for distributed and parallel query processing which employs a *pass-by-reference* semantics by using *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. To cope with this problem, we developed a distributed XML query processing scheme which leverages the benefit of *lazy evaluation*. Our experimental results show up to 22x speedups compared with competitive methods, and demonstrated the importance for distributed XML database systems to take *pass-by-reference* semantics into consideration.

Key words Distributed and parallel database, XML query processing, Lazy evaluation

1. はじめに

XML の利用拡大に伴い、XML データは計算機網に拡散している。計算機間に分散して配置され、刻々と内容が変化することがある動的な XML データ群を効率的に統合する技術は重要である。例えば、XML フィードリーダの利用者は、購読した数千のサイトの更新 (例えば、スポーツの試合結果や最新の災害情報) を更新時間の新しい順に遅延なしに把握したいと要求するかもしれないが、こうしたリアルタイム性が要求される問合せに、現在の XML 問合せ処理技術で応えるのには困難が予想される。その他の例には、バイオ情報データベース群の統合がある。今日のバイオ情報データベースは、異種情報源の統合のためにデータを XML として出版する機能を有すが、個々

のデータベースには世界中の研究室から頻繁なデータの更新や訂正が行われている。そのため、バイオ情報データベース群の統合システムはデータの高い鮮度を保証しなければならない。

こうした状況を考慮したとき、数千の XML データを実時間に XML 問合せ処理することは、現在の XML 問合せ処理技術では不可能な領域である。何故ならば、我々の知る限り、オペレータ間並列性 (Inter-operator parallelism [1]) と分散問合せ処理を共に取り組んだ研究が存在しないからである。これらの二点を同時に取り組むことは、上記した問題設定を克服する上で不可欠である。

単一の計算機上の一つの XQuery 処理器で取り扱うことができないう数多くの XML 文書に対する実時間 XML 問合せ処理を実現するために、我々は一つの問い合わせを複数の副問合せに分

解し、図 1 にあるように分割を行う。そして、分解された副問合せを複数の計算機ノードで実行する。しかしながら、このような階層的に分散したシステムは次に挙げる課題を克服する必要がある。

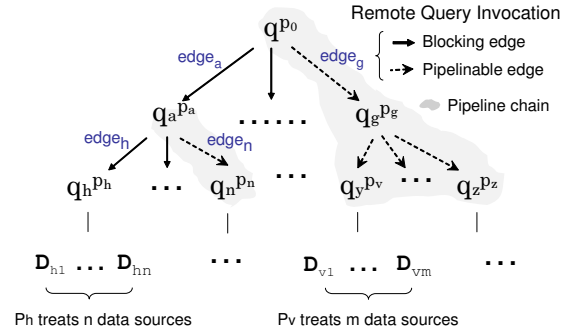
- 協調計算では最終的な結果を出すために、問合せ処理ノード間で中間結果を交換することとなる。この中間結果を交換するのに要する時間は無視できない。特に、XQuery Data Model (XDM) [2] インスタンスのデータ交換は多くの CPU 時間を消費する。何故ならば、関係データベースでは基本的にスカラー値だけを扱えばよかったのに対して、XML データベースでは XML 木などの非スカラー値を交換する必要があるため、符号化/復号化により多くのコストがかかるからである。論文 [3] によれば、XML 文書のパース処理は、1K バイトごとに 175K CPU インストラクションを必要であり、関係データベースにおいて 1 行を関係表に追加するコスト (30K ~ 200K インストラクション) と同じオーダーである。こうしたことから、分散 XML 問合せ処理では、二つのオペレータ間の枝がボトルネックになる可能性がある。

- 現在の XML 問合せ処理器は、分割統治並列性 (Divide-and-conquer parallelism) に加えてオペレータ間の並列性を利用することができない。分割統治並列性については、一つの問合せ q_a は、その副問合せ ($q_h \dots q_n$) を並列にアウト・オブ・オーダー実行することができる (図 1 参照)。ここで、計算機ノード p_a において実行される問合せ q_a の実行時間の表記として $T(q_a)$, $T(q_a)$ のうちローカル計算に要する実行時間を $LT(q_a)$, $T(q_a)$ のうち枝に要する時間を $T(edge_a)$ とする。ここで、枝はリモートサイトで問合せを実行して結果を取得する際に必要な計算を表現する。 $T(q_a)$ は、パイプライン並列性を考慮しない場合、次のように再帰的に定義される。

$$T(q_a) = \underbrace{\max((T(q_h) + T(edge_h)), \dots, (T(q_n) + T(edge_n)))}_{\text{最も時間の要する枝にかかる実行時間}} + LT(q_a)$$

この式に基づけば、一つのノードに要する計算時間は最も時間を要する枝に依存する。図 1 を例にとれば、 q^{p_0} のローカル計算は最後の中間結果が返るまでブロックされ、その間、 p_0 の CPU リソースはアイドル状態となる傾向がある。更に、例えば $LT(q_a)$ のような問合せを並列化できない箇所が、並列化による理論的な性能向上の上限を制限する。アムダールの法則 [4] に基づけば、問合せの並列実行により期待できる性能向上は、しばしば、並列不可能な部分により制限される。こうしたことから、上記の式による制限を越えて、今日のマルチコアプロセッサを含むマルチプロセッサ環境の計算能力を活かすためには、パイプライン化が不可欠となる。

上記の側面を考慮に入れ、本稿では、過去の研究では十分に議論されてこなかった点、分散 XQuery 処理におけるプロセッサノード間のデータ交換に焦点を当てる。そして、過去の手法に対する代替案として、*remote proxy* を利用した効率的なデータ交換手法を提案する。*remote proxy* を利用したとき、実体で



ユーザ問合せ q は、副問合せ q_a, \dots, q_z に再帰的に分解される。 $p_0, p_a, \dots, p_h, \dots, p_z$ の記号は、問合せが実行される計算機ノードを表す。 D_{hn} のような記号 D は、データソースを表す。

図 1 分割統治とパイプライン並列

ある結果シーケンスは、代理 (Proxy) シーケンスに置換され、代理シーケンスのみがリモートの計算ノードに返される。実際の結果 (時に部分結果も可能) は、代理シーケンスにアクセスがあった際に、データ供給ノードから取得される。*remote proxy* の利用には、次の三点の効果がある。

- 我々の手法により、オペレータ独立並列性 (independent-operator parallelism) やパイプライン並列性を用いた問合せの並列実行が可能である。

- 実体シーケンスの計算が要求駆動に行われる。実体シーケンスの新しい FIFO エントリ^[注1]は、エントリが消費されてエントリ量が下限閾値 (low watermark level) に落ち込んだ場合に行われる。この組織化された受注生産方式により、メモリや CPU といったサーバのリソースを効率的に利用することができる。

- 我々の手法では、要求されたデータを供給ノードと消費ノードの間で一対一で交換することにより、従来の値渡しを利用した手法 [5] ~ [7] につき物であったネットワークのトラヒックとバッファの冗長な消費を抑えることができる。中間ノードのデータ交換を省略する効果には、ネットワークのトラヒックとネットワークのレイテンシの削減に留まらず、中間ノードにおける冗長な符号化・復号化の計算を削減できることにある。

我々は提案手法を、*XBird/D* として、これまでに我々が提案してきたネイティブ XML データベース *XBird* [8] 上に実装した。実験結果により、競合実装に対して最大 22 倍の性能向上が得られたことを示し、分散 XML データベースシステムが参照渡しによるデータ交換を採用することの妥当性を実証する。

2. 関連研究と未解決問題

本節では、関連研究を挙げると共に、これまでの分散 XQuery 処理における問題点を挙げ、我々の解決作を述べる。

図 2 に示すのは、典型的な値渡しを利用したりリモート問合せ

(注1): XDM [2] では、結果は 0 以上のアイテムを含むシーケンスであり、そのアイテムはオペレータによって FIFO 順に消費される。本稿では、シーケンスに残存するアイテムを FIFO エントリを呼ぶ。

処理の流れである。値渡しは、過去の分散 XML 問合せ処理器で一般的に用いられてきた [5] ~ [7]。次に、この戦略の根本的な問題を挙げる。

- 低いオペレータ間並列性

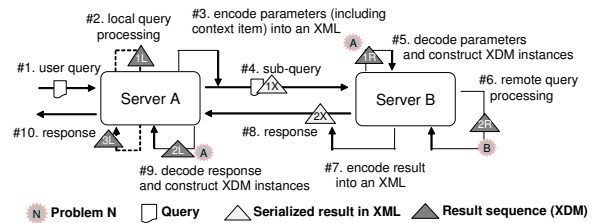
値渡しを実行戦略に用いた場合、リモート問合せは図 2 に示すように逐次的に実行される。そのため、サーバ A で問合せを処理している間、サーバ B はアイドル状態となる。対照的に、サーバ B が問合せを処理している間、サーバ A がアイドル状態になる傾向がある。また、図 1 を用いて既に議論したように、値渡しを利用した並行 XML 問合せ処理はパイプライン並列を扱うことができないために、入れ子問合せ処理時にオペレータ間並列性を享受できない。我々は、\$doc が XMark [9] のスケールファクタ (SF) 5 及び 10 で生成した XML 文書を扱う次の問合せをリモートの計算ノードで実行して結果を得るのに、どの処理がどの程度問合せ処理時間を占めるのかを計測した。その際、結果 XDM インスタンスの送受信されるデータ量は、SF が 5 のとき 2164 KB であり、SF10 のとき 4307 KB である。

```
for $a in $doc/site/closed_auctions/closed_auction
where $a/price/text() >= 40 return $a/price
```

図 3 に示した結果は、符号化・復号化を含むレイテンシに要した時間が、リモートの計算ノードにおける問合せ実行時間とほぼ同等であったことを示している。このことは、図 1 の個々の枝が、潜在的なボトルネックになり得るとする我々の主張を裏付けるものである。さらに、それぞれの枝がブロックされる枝 (図 1 を参照) となるため、値渡しではオペレータ間並列性が制限される問題がある。この問題への解決策として、我々はパイプライン並列性とオペレータ間並列性を我々の *remote proxy* に取り入れた。*remote proxy* により、我々のシステムは図 1 の個々の枝をパイプライン可能とすることができ、それが故に、計算ノードは原理的に並列に計算を行うことができる。

- リソース活用の貧弱さ

マルチユーザ環境では、複数の並行した問合せが多くのシステムリソースを消費する。そのため、適切に CPU 及びメモリ資源を問合せ毎に割り当てることが、特にマルチプロセッサ・マルチコア環境において、重要となる。例えば、理想より低いオペレータの並列性を選択すると、システムの利用効率の低下を招き、スループットが下がる。一方で、理想より高いオペレータの並列性は、一つの問合せに過剰なリソースを消費し、結果としてリソース利用の競争を招く。現在の値渡し戦略を利用した場合、例えば図 2 の A と B の箇所など、リソース利用の競争が頻繁に発生する。ここで、完全なパラメータシーケンスを A において符号化する場合を想定する。サーバ A は多くの CPU サイクルとメモリ領域を消費し、その間、サーバ B におけるリモート問合せ実行は符号化が終わるまでブロックされる。サーバ B でパラメータを受信し復号化する際には、サーバ B は利用可能なメモリが不足する可能性がある。このような状況は、4. 節での我々の評価実験で現実のものとして発生した。この問題に対処するために、我々は *remote blocking-queue* を利用した



サーバ A は (3.1 節で解説する) *BDQExpr* を受け取り¹、ローカルの問合せ処理器がその問合せを評価する²。*BDQExpr* を評価する際、その計算されたパラメータ $1L$ とコンパイルされた式はリモートサーバ B に送信 (符号化³ され移送⁴) される。すると、サーバ B は符号化されたパラメータ $1X$ を復号化し、XDM インスタンス $1R$ を生成する⁵。その後、サーバ B は受信したコンパイル済みの式を評価する⁶。そして、サーバ B の中間結果 $2R$ がサーバ A に返される^{7,8}。サーバ A はサーバ B からの応答 $2X$ を復号化し、XDM インスタンス $2L$ を生成する⁹。最後に、サーバ A は式全体としての評価を行い²、その結果 $3L$ をユーザプログラムに返却する¹⁰。

図 2 値渡しを利用した典型的なリモート問合せ実行の流れ

効率的なリソース活用手法を提案する。*remote blocking-queue* により、オペレータの実行レートが調整される。

- 符号化と復号化のオーバーヘッド

過去の研究では、プロセッサノード間のデータ交換に XML 形式を利用してきた [5] ~ [7]。しかしながら、論文 [3] で言及されるように、入力 XML データの復号化には多くの CPU サイクルが消費される。論文 [10] で Bayardo らは、*view-source* 原則を諦めることになる以外には、バイナリ形式の XML 符号化は殆どのアプリケーションにおいて有効であると述べている。一方で、ナイーブな (ブロックされる) バイナリ符号化は、パイプライン化された XML ストリーム処理を阻害する可能性がある。そこで我々は、XDM インスタンスを SAX 風にバイナリ形式のイベントストリームに漸増的に符号化・復号化する手法を採用した。さらに、3. 節で参照渡しのための効率的な *direct result forwarding* メカニズムを提案する。

3. XBird/D の実装

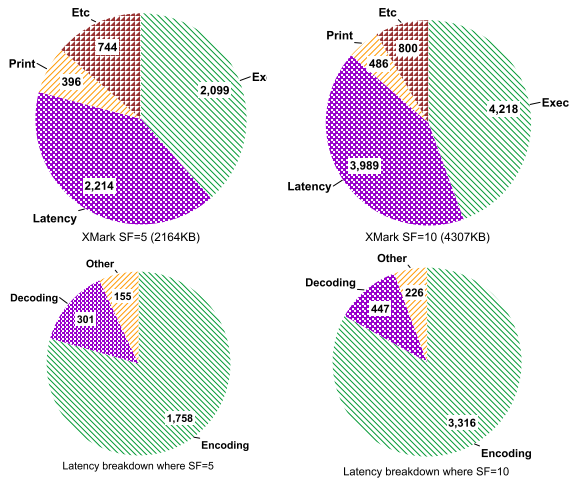
3.1 言語拡張: BDQ

我々は、XQuery [11] に対してリモート問合せ実行の言語拡張を行う。この拡張を *XBird Distributed Query* (略して *BDQ*) と呼ぶ。図 4 が、XQuery の *PrimaryExpr* に対する文法上の拡張である。*BDQExpr* では、 $Expr_2$ が $Expr_1$ から導かれるリモートの計算ノード P で実行される。 P には、*fn:string(fn:exactlone(Expr₁))* を取り、その形式には *//host:port/name* という URL を取る。ここで、*name* はサービスレジストリに登録されたリモートサービスの登録名であり、サービスレジストリは、*host* と *port* により識別される。

3.2 Remote Proxy

2. 節で言及したように、我々の分散 XQuery プロセッサ *XBird/D* は、オペレータ間並列を実現するために *remote proxy* を利用する。

XBird と表記するベースラインの XQuery プロセッサは、



Exec: リモート計算ノードにおける問合せ実行時間
 Latency: 符号化・復号化とネットワークの遅延を含むリモート問合せ実行のレイテンシ
 Print: XDM インスタンスをファイルに直列化するのに有した時間
 ETC: 問合せのコンパイルとその他のローカル計算のフットプリントを含む実行時間

図 3 リモート問合せ処理に有した時間の詳細 (単位はミリ秒)

```
BDQExpr ::= "execute at" Expr1 "{" Expr2 "
```

```
PrimaryExpr ::= Literal | .. | Constructor | BDQExpr
```

図 4 BDQ による文法拡張

Volcano イテレータモデルに基づき、BEA/XQRL XQuery プロセッサ [12] と同様のイテレータ木による反復問合せ処理を行う。パイプライン化された問合せ処理により、オペレータはループ処理や軸を辿る処理を遂行する。このイテレータに基づく実行モデルは、問合せ式の遅延評価を可能とするが、XBird/D の分散処理設計においても重要な役割を担う。

ここで、remote proxy を利用した pass-by-reference 実行戦略がどのように実現されるかを示す。Remote proxy は、分散オブジェクト通信 [13] のための代理 (Proxy) デザインパターンの拡張であり、XBird/D 特有の機能ではないが、これまでに分散 XML 問合せ処理に用いられた例はなく、また remote proxy を利用することだけでオペレータ間並列を実現することできない。そこで、次に示す remote proxy と非同期に行う中間結果の実体生産を組合せを試みた。

非同期生産とキュー管理

シーケンスに含まれるアイテム群は FIFO キューに格納され、各々のアイテムは消費ノードで処理される別のオペレータによって消費されると仮定する。図 5 に、我々の remote proxy 実装の概要を示す。リモート問合せを実行すると、中間結果 (result entity sequence) が、代理オブジェクトに包まれ、その代理オブジェクト (result proxy sequence) が呼び出し元 (Peer₁) に返される。その時、リモートのオペレータは result entity sequence のアイテムを、キュー一杯になるまで、非同期に生産する (図 5(a))。一方で、アイテムの受信はキューが空でなくなるまでブロックされる (図 5(c))。remote proxy は、ローカルのキューが空になった段階 (図 5(b)) で、リモートのエントリ群

を取得する。リモートのキューから取得するアイテム数は、それぞれの問合せごとに、初期取得サイズと取得サイズの伸張係数を指定することによって設定することが可能である。取得サイズは、そのパラメタの設定値に基づき、最大取得数の閾値に達するまで自動的に成長する。この機能は、クライアント・サーバ間の通信回数を削減することを狙ったものである。この remote blocking-queue によるシンプルで効果的なキュー管理により、我々のシステムは供給過多・供給不足を避け、システムのリソースを活かすことができる。

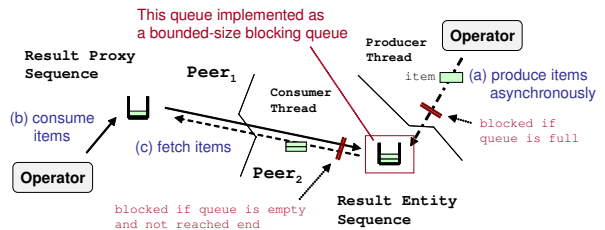


図 5 remote proxy を利用した場合のプロセッサエレメント間の相互作用

Direct Result Forwarding

2. 節で言及したように、XBird/D は、分散問合せ実行におけるレイテンシ (符号化・復号化やネットワーク遅延) を減らすために、direct result forwarding 機能を有す。ここでは、図 6 を用いて、分散し入れ子になった問合せの実行がどのように処理されるかを示す。図 6 では、filter、reduce、select₁、そして select₂ 関数が、それぞれ、PE1、PE2、PE3、そして PE4 で実行される。図 7 の左半分が、その入れ子実行木を表現する。reduce 関数は、closed_auction と open_auction を集め、それぞれの先頭から 1000 個のアイテムを返す。reduce 関数は、ローカルリソースへのアクセス (fn.doc か fn.collection) を含まないため、その実行は再配置可能である。この再配置による最適化は、コンパイル時ではなく実行時に動的に行われる^(注2)。我々の XQuery プロセッサは Volcano イテレータモデルを採用しているため、その結果イテレータの計算は call-by-need 形式の遅延評価によって導出される。動的再配置のために提案手法が行うのは、結果イテレータである P1 と P2 を呼び出し元のオペレータ (このケースでは PE1) に転送するだけである。P1 と P2 は PE1 で実行され、中間結果は PE3 と PE4 より直接取得される。この最適化は、PE2 における符号化・復号化を無視できるという点で効果的である。図 3 で示したように、リモート問合せ実行において、符号化・復号化に要する時間は全体の実行時間の大半を占めるからである。

既存手法の DXQ [7] は、この問題に対して、論文 [14] で提案された内包表現 (intensional expression) を利用している。内包表現を用いることで、ある計算ノードは、その実行結果に中間結果に埋め込むことで、実行すべき仕事の幾つかを呼び出し元に委譲することができる。例えば、内包表現は reduce 関数

(注2): 動的に再配置を行うことの利点としては、実行時情報を動的再配置の戦略決定に用いることができる点にある。

での計算を次のように変形し、譲渡する。

```
fn:sequence( (<closed_auction> ... </closed_auction>,
            <closed_auction> ... </closed_auction>), 1, 1000),
| (<open_auction> ... </open_auction>,
   <open_auction> ... </open_auction>), 1, 1000 )
```

しかしながら、我々の経験に基づけば、内包表現の利点は実行結果が小さい場合に制限される。何故ならば、内包表現は内包結果のために負荷の高い符号化・復号化を必要とし、符号化・復号化は、問合せの実行よりも計算機のリソースを消費することがあるからである。一方で、我々のイテレータを転送するアプローチは、そのような欠点なしに遅延評価の効果を達成することができる。

```
declare function bdq:select1() {
  execute at $PE3 {
    fn:collection($col)/site/closed_auctions/closed_auction
  }
};
declare function bdq:select2() {
  execute at $PE4 {
    fn:collection($col)/site/open_auctions/open_auction
  }
};
declare function bdq:reduce() {
  execute at $PE2 {
    ( fn:subsequence(bdq:select1(), 1, 1000)
      | fn:subsequence(bdq:select2(), 1, 1000) )
  }
};
declare function local:filter() {
  for $a in bdq:reduce()
  where $a/seller/@person >= "person10000"
    or $a/buyer/@person >= "person10000"
  return $a
};
local:filter() (: execute at PE1 :)
```

図 6 入れ子問合せ実行の例

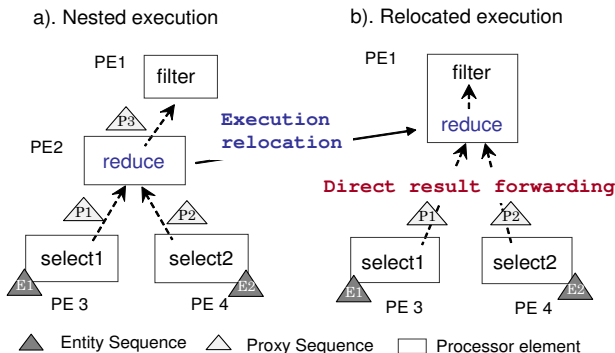


図 7 実行再配置と direct result forwarding

4. 評価実験

提案手法による機能強化である *remote proxy* と *direct result forwarding* の効果を測るために、最先端の分散 XQuery プロセッサである MonetDB/XRPC [6] バージョン 4.18.1 との性能比較を行った。

実験環境には、4 台の計算機を利用した。本節では、それぞれの計算機で実行される XQuery プロセッサを PE1 ... PE4 として表記する。それぞれの計算機は、PE2 が Athlon 64 X2 2.4GHz の CPU を装備しているのを例外として、1Gb/s の Ethernet で繋がり、CPU として Pentium D 2.8GHz、2GB のメモリ、SuSE Linux 10.2 を OS、JDK 1.6 を実行環境として装備する。XBird/D の評価には、図 6 に示す問合せを用い、同等の問合せを MonetDB/XRPC の評価に用いた。データセットには、XMark [9] のデータ生成器にスケールファクタ 10 を設定して生成した 1.1GB の XML 文書を利用し、生成した文書は図 6 の \$col 変数に束縛した。生成した XML 文書は、PE3、PE4 で実行中の MonetDB と XBird のデータベースインスタンスに予め取り込んだ。

実験結果の概要が、図 8 である。実験では、*remote proxy* (Proxy)、*direct result forwarding* を有効とした *remote proxy* (Proxy+Forward)、我々の値渡し戦略の実装 (Value)、MonetDB/XRPC の値渡し戦略 [6] (XRPC) の四つの評価戦略の比較を行った。

図 8 にあるように、我々の *remote proxy* を利用したりリモート参照渡しの実装は実行時間に大きな改善を示した。これは、PE3 と PE4 における不必要な計算を除去した結果であり、分散 XQuery 処理に遅延評価を適用した結果である。値渡し戦略を利用したりリモート問合せの評価は、完全な結果を一度に計算・生産するが、それらの全てが後の計算で利用されていない。これは明らかに非効率であり、我々の *remote proxy* 評価戦略 (Proxy) が 9 倍、値渡しに比べて短い実行時間を示したことを説明する。さらに、*direct result forwarding* は PE2 における冗長な符号化/復号化を除去し、仲介された通信のオーバーヘッドを削減した。その結果、我々の Java で実装したプロトタイプシステムは Java 仮想マシン上で動くシステムであるのに係わらず、競合実装である C++ で実装された XRPC に比べて 22 倍の性能向上を示すに至った。

また、我々の *remote proxy* を利用したシステムのみが、マルチユーザ環境をシミュレートした 30 スレッドからの総計 160 の並行した問合せを 160 秒で処理することができた。この時の最大実行時間は 53.75 秒であり、平均実行時間は 36.75 秒である。値渡しを利用した実装 (Value と XRPC) では、頻繁なスワップイン/スワップアウトが発生し、そのリソース活用の貧弱さを露呈した^(注3)。この結果から、上記の設定における我々の提案手法の確かな優越性を確認した。

(注3): XRPC では 10 分しても 5 つの並行した問合せの最初の結果を返すことはなかった。

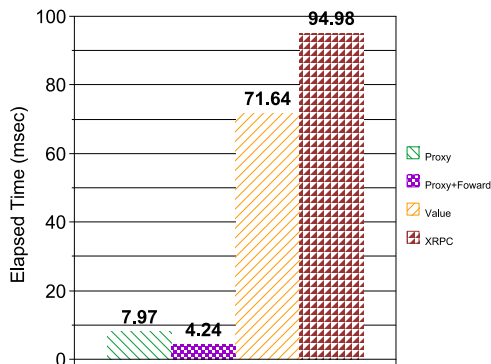


図 8 四つ実行戦略の性能比較

5. 結 論

本稿では、我々は *remote proxy* を利用した効率的な分散 XML 問合せ処理戦略を提案した。実験結果から、提案手法は競合手法に対して最大 22 倍の性能向上を示すことが明らかになり、分散 XML データベースシステムが、値渡しの代替として参照渡しを利用した *call-by-need* 戦略を考慮に入れることの重要性を実証した。さらに、我々の改善 (*remote blocking-queue* によって管理される非同期生産) により、分散 XML 問合せシステムは、オペレータ間並列性をサポートしつつ、システムのリソースを効率的に活用することができる。

今後の課題には、分散システムに参加するノード群のシステムリソースとその活用状況 (例えば、CPU の利用率やメモリの空き状況やスベック) を考慮に入れたりリモートの問合せ処理器への動的な実行の割り当てと実行戦略の選択モデルの開発が挙げられる。

謝 辞

本研究の一部は、日本学術振興会科学研究費補助金若手 (B) (課題番号 19700094)、科学技術振興機構戦略的創造研究推進事業 (CREST) ならびに文部科学省科学研究費補助金特定領域研究 (課題番号 19024058) による。ここに記して謝意を表します。

文 献

- [1] T. M. Oszu and P. Valduriez: “Principles of Distributed Database Systems (2nd ed.)”, Prentice Hall (1999).
- [2] W3C: “XQuery 1.0 and XPath 2.0 Data Model (XDM)”, <http://www.w3.org/TR/xpath-datamodel/>.
- [3] M. Nicola and J. John: “XML Parsing: A Threat to Database Performance” (2003).
- [4] G. M. Amdahl: “Validity of the single processor approach to achieving large scale computing capabilities”, Morgan Kaufmann Publishers Inc. (2000).
- [5] C. Ré, J. Brinkley, K. Hinshaw and D. Suci: “Distributed XQuery”, Proc. IIWeb, pp. 116–121 (2004).
- [6] Y. Zhang and P. A. Boncz: “XRPC: Interoperable and Efficient Distributed XQuery”, Proc. VLDB (2007).
- [7] M. F. Fernandez, T. Jim, K. Morton, N. Onose and J. Simeon: “DXQ: A Distributed XQuery Scripting Language”, Proc. XIME-P (2007).
- [8] 油井誠 宮崎純: “効率的な xquery 処理のための dtm に基づく xml ストレージ”, 情報処理学会論文誌: データベース, 48, pp. 128–148 (2007).

- [9] A. R. Schmidt, F. Waas, M. L. Kersten, D. Florescu, I. Manolescu, M. J. Carey and R. Busse: “The XML Benchmark Project”, Technical Report INS-R0103, CWI (2001).
- [10] R. J. Bayardo, D. Gruhl, V. Josifovski and J. Myllymaki: “An Evaluation of Binary XML Encoding Optimizations for Fast Stream Based XML Processing”, Proc. WWW, pp. 345–354 (2004).
- [11] W3C: “XQuery 1.0: An XML Query Language”, <http://www.w3.org/TR/xquery/>.
- [12] D. Florescu, C. Hillery, D. Kossmann, P. Lucas, F. Riccardi, T. Westmann, M. J. Carey and A. Sundararajan: “The BEA streaming XQuery processor”, VLDB Journal, 13, 3, pp. 294–315 (2004).
- [13] H. Rohnert: “The proxy design pattern revisited: Pattern languages of program design 2”, Addison-Wesley, Inc. (1996).
- [14] T. Milo, S. Abiteboul, B. Amann, O. Benjelloun and F. D. Ngoc: “Exchanging Intensional XML Data”, Vol. 30, pp. 1–40 (2005).