情報処理学会論文誌　データベース　Vol. 2　No. 1　104–115 (Mar. 2009)

Remote Proxy を利用した分散 XQuery 問合せ処理

油井　誠[†1,†2]　宮崎　純[†1]
植村俊亮[†3]　加藤博一[†1]

Remote proxy 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，提案手法が値渡しをする既存手法に対し最大 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．

Distributed XQuery Processing Using Remote Proxy

Makoto Yui,[†1,†2] Jun Miyazaki,[†1] Shunsuke Uemura[†3] and Hirokazu Kato[†1]

We propose a scheme for distributed and parallel query processing which employs a *pass-by-reference* semantics by using *remote proxy*. Previously proposed methods that 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.

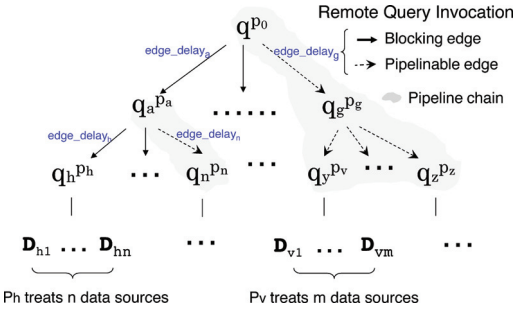
1. はじめに

XML データの増加にともない，大規模 XML データを分散処理することが重要な課題となっている．これまで XML データの分散処理の研究は，データ量とデータ構造，アクセス頻度を考慮に入れたデータ分割方式[1] や構造概要を利用した分散問合せ処理方式[2),3)] などが行われてきたが，計算機間のデータ交換方式については，値渡しを利用するのがつねであり，他の選択肢について十分な検討がなされていない．本論文では，特に問合せが階層的に実行される場合において，分散 XML データベースシステムが参照渡しを利用したデータ交換方式を考慮に入れることの重要性を述べる．なお，本論文ではデータ交換を主眼とし，データの配置方式は問わない．

単一の計算機上の 1 つの XML 問合せ処理器で取り扱うことができない数千の 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 1 つの問合せを複数の問合せに分解し，**図 1** にあるように分割する．そして，分割された問合せを複数の計算機ノードで実行する．XML データ処理に，このような階層的に分散したシステム（たとえば，複数段の *MapReduce* [4] 処理）を適用する場合，次にあ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，XML 問合せの中間結果[5] のデータ交換は多くの CPU 時間を消費する．なぜならば，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 6) によれば，XML 文書のパース処理には，1 K バイトごとに 175 K CPU インストラクションが必要であり，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（30 K〜200 K インストラクション）と同じオーダである．こうしたことから，分散 XML 問合せ処理では，2 つのオペレータ間の RPC（Remote Procedure Call）が処理時間の高い比重を占める．
- 値渡しを利用する現在の XML 問合せ処理器は，オペレータ間並列性を十分に享受で

†1 奈良先端科学技術大学院大学情報科学研究科
Graduate School of Information Science, Nara Institute of Science and Technology
†2 日本学術振興会特別研究員 DC-2
JSPS Research Fellow DC-2
†3 奈良産業大学情報学部情報学科
Department of Informatics, Faculty of Informatics, Nara Sangyo University



ユーザ問合せ q は，子問合せ $q_a, \cdots, q_z$ に再帰的に分解される．$p_0, p_a, \cdots, p_h, \cdots, p_z$ 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
Fig. 1　Divide-and-conquer and pipeline parallelism.

きない．オペレータ間並列性には，パイプライン並列性（pipeline parallelism）とオペレータ独立並列性（independent-operator parallelism）がある[7]．パイプライン並列を利用した問合せ実行では，生産者・消費者の関係にある複数のオペレータが並列に実行される．たとえば，図 1 における $q_y, \cdots, q_z$ と後続する q_g が，パイプライン処理により並列に実行される．パイプライン実行方式では，q_g が入力データである $q_y, \cdots, q_z$ の実行がすべてが完了するのを待つことなく，その実行を進めることができる．オペレータ独立並列は，並列に実行される複数のオペレータに依存関係がないときに成立する．オペレータ独立並列は計算機ノード間の干渉がないため，高い並列実行性能が期待できるが，その適用は *bushy* 型の実行に限られ，また，多くの実行リソースをバースト的に消費することが問題点として知られている[7),8),⋆1]．

オペレータ独立並列性については，図 1 においても，1 つの問合せ q_a は，その子問合せ（$q_h, \cdots, q_n$）を並列にアウト・オブ・オーダ実行することができる．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 *edge_delay* に要する時間を $T(edge\_delay_a)$ とする．本論文で *edge_delay* とは，2 つのオペレータ間の RPC に関するオーバヘッドを指す．*edge_delay* の主な構成要素は，通信オーバヘッドと計算結果やパラメータの符号化・複合化に要する計算オーバヘッドである．$T(q_a)$ 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．

$$T(q_a) = \mathbf{max}((T(q_h) + T(edge\_delay_h)), \cdots, (T(q_n) + T(edge\_delay_n))) + LT(q_a)$$

この式に基づけば，1 つのノードに要する計算時間は最も時間を要する *edge_delay* 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．さらに，たとえば $LT(q_a)$ 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[10)] 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．上記の式による制限を超えて，並列計算機環境の計算能力を活かすためにはパイプライン処理が鍵となる．

上記の側面を考慮に入れ，本論文では，過去の研究では十分に議論されてこなかった点，分散 XML 問合せ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[11)] を利用した効率的なデータ交換手法を提案する．Remote proxy は，プロキシデザインパターンを遠隔オブジェクトの操作に応用したソフトウェアパターンである．提案システムでは，遠隔サイトの計算結果に対して remote proxy を適用する．Remote proxy を利用したとき，実体である結果シーケンス[⋆2]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（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 3 点の効果がある．

- 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で行われる．実体シーケンスの新しいエントリ[⋆3]の生産は，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．3.3.2 項で述べる *remote blocking-queue* による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．
- 3.3.3 項で述べる *direct result forwarding* により，要求されたデータを供給ノードと

⋆1 問合せの並列実行の最適化では，総仕事量とレスポンス時間，メモリ消費を考慮に入れる必要があるが，総仕事量とレスポンス時間にはトレードオフの関係がある[9)]．

⋆2 関係データベースが集合を扱うのに対して，XQuery[12)] で扱うデータはシーケンス（順序付き Bag）である．

⋆3 XDM[5)] では，結果は 0 以上の Item インスタンスを含むシーケンスであり，その Item はオペレータによって FIFO 順に消費される．Item インスタンスは，XML のノードか原子型（アトム）からなるオブジェクトである．

消費ノードの間で 1 対 1 で交換することで，従来の値渡しを利用した手法[13)–15)]につきもの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とどまらず，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がある．

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*[16)] 上に実装した．実験結果により，値渡しを利用する MonetDB/XRPC[14)] 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．

本論文の構成は次のとおりである．2 章では，分散 XML 問合せ処理に値渡しを利用した場合の問題点を示す．3 章では，提案手法である remote proxy を利用した分散 XML 問合せ処理手法を説明する．4 章で提案手法の評価を行う．5 章で関連研究を紹介し，6 章でまとめる．

2. 値渡しを利用した場合の問題点

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[13)–15)]．本章では，値渡しの問題点をあげて我々の解決策との対応付けを行う．**図 2** に示す値渡しを利用した典型的なリモート問合せ処理の流れを用い，値渡し戦略を利用した場合の問題をあげる．

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．図 1 を用いてすでに議論したように，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．

予備実験として，入力文書を表す$doc 変数に束縛される XML 文書を代表的な XML ベンチマークツールの 1 つである XMark[17)] のスケールファクタ（SF）5 および 10 で生成するものとし，**図 3** 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，問合せ処理のうち，どの処理にどの程度の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 2,164 KB であり，SF が 10 のとき 4,307 KB である．**図 4** 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 *edge_delay* が，潜在的なボトルネックになりうるという我々の主張を裏付けるものである．

この問題への解決策として，提案手法では，3.3.1 項に示す Volcano イテレータモデルに

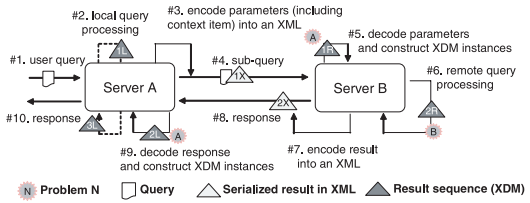
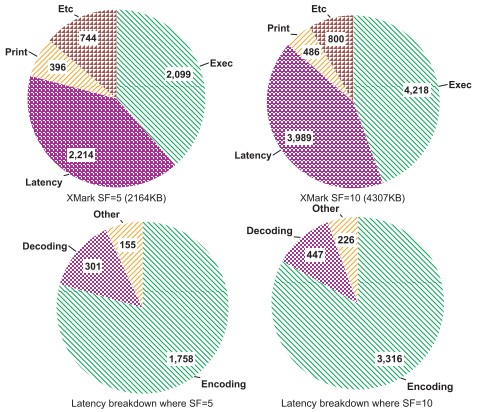


図 2　値渡しを利用した典型的なリモート問合せ実行の流れ
Fig. 2　A typical remote query execution flow with pass-by-value semantics.

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

図 3　XMark 問合せ Q5
Fig. 3　XMark query Q5.



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 4　リモート問合せ処理に有した時間の詳細（単位はミリ秒）
Fig. 4　Breakdown of remote query processing time (in msec.).

基づくパイプライン並列性を remote proxy に取り入れた．このことにより，図 1 の個々の *edge_delay* をパイプライン処理可能とすることができる．さらに，3.3.2 項に示す非同期式パイプライン（asynchronous pipeline）処理によって，オペレータ独立並列性を確保する．1 章で述べたように，オペレータ間並列性 ∋{ パイプライン並列性，オペレータ独立並列性 } であるが，上記の 2 つの取り組みにより，図 1 でいうパイプライン処理可能な範囲（*pipeline chain*）を広げ，高いオペレータ間並列性を実現できる．

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 およびメモリ資源を問合せごとに割り当てることが重要となる．たとえば，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，1 つの問合せに過多なリソースを消費し，結果としてリソース利用の競合を招く．現在の値渡し戦略を利用した場合，たとえば図 2 の A と B の箇所などで，リソース利用の競合が頻繁に発生する．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 章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，3.3.2 項で *remote blocking-queue* を利用した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[13)–15)]．しかしながら，論文 6) で言及されているように，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 18) で Bayardo らは，view-source 原則を諦めることになる以外は，バイナリ形式の XML 符号化はほとん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．

図 4 に示したように，XML 問合せ結果の出力には子孫ノードの直列化をともなうため，符号化のコストはバイナリ形式においても，依然として顕著である．そこで，3.3.3 項で，中間ノードにおける冗長な符号化・復号化を削減する参照渡しのための *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[12)] 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 5** 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 $\mathcal{P}$ で実行される．$\mathcal{P}$ には，*fn:string* (*fn:exactly-one* ($Expr_1$)) をとり，その形式には *//host:port/name* という URL をとる．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記名であり，サービスレジストリは，*host* と *port* により識別される．

3.2 分散問合せ処理のスキーム

XBird/D における分散問合せ処理の概要を，**図 6** に示す．問合せ処理の流れは，次のとおりである．

(1)　パーザは問合せを字句解析，構文解析，意味解析し，論理実行プランを生成する．

(2)　コンパイラは論理実行プランから，物理実行プランを生成する．ここで，問合せ木の型解決などの静的解析（static analysis）[12)] が行われる．生成される物理実行プラン

$$
\begin{array}{rcl}
BDQExpr & ::= & \text{“}execute\ at\text{”}\ Expr_1\ \text{“\{”}\ Expr_2\ \text{“\}”} \\
PrimaryExpr & ::= & Literal \mid .. \mid BDQExpr
\end{array}
$$

図 5　BDQ による文法拡張
Fig. 5　BDQ grammar extensions.

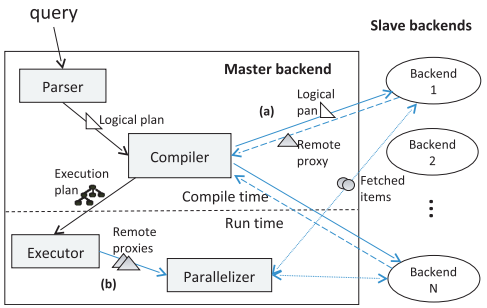


図 6　XBird/D の分散問合せ処理スキーム
Fig. 6　Distributed query processing scheme in XBird/D.

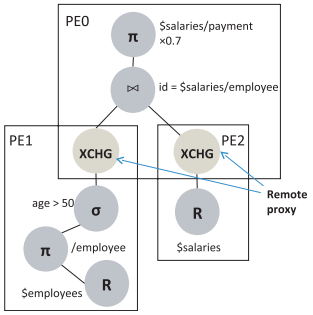


図 7　物理実行プランの概略図
Fig. 7　A brief sketch of a physical execution plan.

は Volcano イテレータモデル[19] に基づく．

図 6 (a)：論理実行プランにおける遠隔実行部分は，遠隔サイトに送信されてコンパイルされる．このとき，遠隔サイトにおける最適化に必要な情報も送信される．論理実行プランの該当箇所は，遠隔サイトが解決した物理実行プランに置換される．遠隔サイトが解決した実行プランは，remote proxy に内包されるプロキシオブジェクトである．この remote proxy もまた Volcano イテレータモデルに基づく．Volcano イテレータモデルとその実行過程については，3.3.1 項で述べる．

(3)　実行器（Executor）は．物理実行プランである Volcano イテレータを評価する．

図 6 (b)：実行器は，まず，プラン木に含まれる remote proxy を並列化器（Parallelizer）に渡す．ここで，それぞれの remote proxy は遠隔計算機の物理実行プランを代理するものである．並列化器は，すべての remote proxy を並列に評価する．Remote proxy の評価方法を 3.3 節に示し，3.3.2 項で，並列化器が個々の remote proxy を処理する仕組みを示す．

ステップ (2) で生成される物理実行プランの一例を，関係代数を用い，**図 7** に示す．この例では，計算ノード PE1 で実行されるパス問合せ$employees/employee[age>50] の選択結果と計算ノード PE2 で実行される$salaries が，計算ノード PE0 において employee 識別子によって結合され，その結合結果に含まれる employee の賃金を 0.7 倍した数値が列挙される．図 7 の XCHG は，Volcano イテレータモデルにおける Exchange オペレータ[19),20)] を表す．Exchange オペレータはメタオペレータであり，下位の実行をラップしてイテレータとしての機能を提供するプロキシとして機能する．提案手法においては，remote proxy

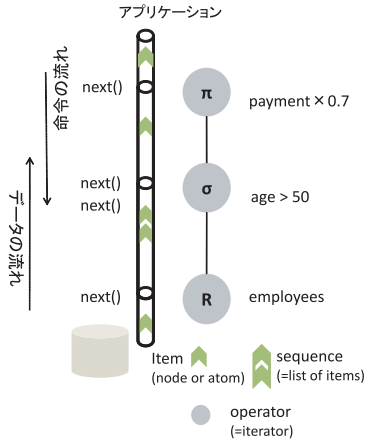


図 8　Valcano イテレータモデルにおけるパイプライン処理・遅延評価のしくみ
Fig. 8　Mechanism of pipeline processing/lazy evaluation in the Volcano iterator model.

が Exchange オペレータの役割を演じ，遠隔サイトの実行プランとローカル実行プランを結び付ける糊として機能する．この実行プランは，Volcano イテレータモデルに基づいてパイプライン処理される．3.3 節で，その実行過程を説明する．

3.3　Remote Proxy

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，Volcano イテレータモデルに基づく remote proxy を利用する．*XBird* と表記するベースラインの XQuery プロセッサもまた，Volcano イテレータモデルに基づき，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たどる処理を遂行する．

3.3.1　Volcano イテレータモデルに基づく問合せ処理

図 8 に Volcano イテレータモデルにおける問合せの評価の仕組みの概要を示す．評価は上位のオペレータから下位のオペレータに伝播され，データは下層から上層へ流れる．1 度にオペレータへのすべての入力が渡される *set-at-a-time* の問合せ評価戦略と異なり，上位層（アプリケーション）の要求に応じてデータがオペレータに供給される．このため，top-k 検索やカーソルを利用するようなユーザアプリケーションでの利用に有効に機能する．Volcano イテレータモデルは *set-at-a-time* 手法との対比で，*tuple-at-a-time* と呼ばれることもある．複数のタプルを同時にオペレータに供給する（つまり，*tuples-at-a-time* とする）ことで，

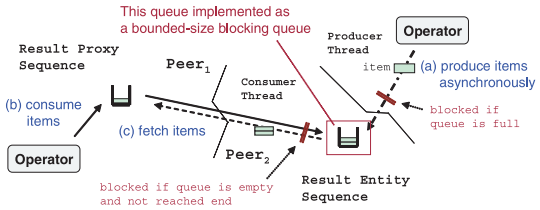


図 9　Remote proxy を利用した場合のプロセッサエレメント間の相互作用
Fig. 9　Server/Client interaction between processor elements using a remote proxy.

CPU キャッシュの利用効率に優れる手法が近年提案されている[21]．

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．提案手法では，Volcano イテレータによる実行モデルを分散するノード間の取引に応用する．遅延評価により，ユーザアプリケーションや上位のオペレータが必要とする量のタプル（XQuery では Item インスタンス）を供給することにより，メモリの最大消費量を抑え不要な通信オーバヘッドを省くことが可能となる．同時に，*tuple-at-a-time* 方式で問題となるノード間の RPC 回数を軽減するために，*tuples-at-a-time* 手法を採用する．

3.3.2 項で，remote proxy の評価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のための代理（Proxy）デザインパターンの拡張[11] 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はオペレータ間並列性を十分に享受することができない．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の組合せを開発した．

3.3.2　Remote Proxy を用い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 9** 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（Peer_1）に返される．そのとき，リモートのオペレータは *result entity sequence* のアイテムを，キューが一杯になるまで，非同期に生産する（図 9 (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 9 (c)）．Remote proxy は，ローカルのキューが空になった段階（図 9 (b)）で，リモートのエントリ群を取得する．リモートの

```
declare function bdq:select1() {
  execute at $PE3 {
    fn:collection($col)/site/closed_auctions/closed_auction(a)
  }
};
declare function bdq:select2() {
  execute at $PE4 {
    fn:collection($col)/site/open_auctions/open_auction(b)
  }
};
declare function bdq:reduce() {
  execute at $PE2 {
    (  fn:subsequence(bdq:select1(), 1, $length)
     | fn:subsequence(bdq:select2(), 1, $length) )
  }
};
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 10　入れ子問合せ実行の例
Fig. 10　Nested remote query execution example.

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，そのパラメー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．

3.3.3　Direct Result Forwarding

2 章で言及したように，*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* 機能を有する．ここでは，**図 10** 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 10 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．本論文では計算機上で実行される XQuery プロセッサを PE と

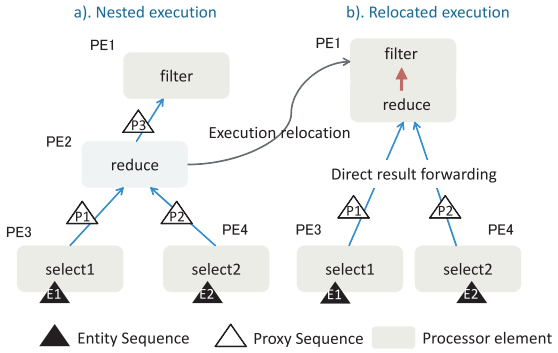


図 11　実行再配置と direct result forwarding
Fig. 11　Execution relocation and direct result forwarding.

表記する．**図 11** 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$length 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 関数か fn:collection 関数）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[⋆1]．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-name* 形式[⋆2]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 4 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．

DXQ [15] は，問合せの実行再配置に，論文 22) で提案された内包表現（intensional expression）を利用する．内包表現は部分評価（partial evaluation）の一形態[23] であり，呼び出された計算ノードは，その実行プランに必要に応じて部分結果に埋め込むかたちで，実行すべき仕事のいくつかを呼び出し元に委譲することができる．たとえば，内包表現は *reduce* 関数での計算を次のように変形し，譲渡する．

⋆1 動的に再配置を行うことの利点は，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．
⋆2 中間結果のサイズが大きいため，メモ化（Memoization）は利用していない．

```
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．なぜならば，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得ることができる．

4. 評 価 実 験

実験環境には，4 台の計算機を利用する．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 2.4 GHz の CPU を装備しているのを例外として，1 Gb/s の Ethernet でつながれ，CPU として Pentium D 2.8 GHz，2 GB 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．データセットには，XMark [17] 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 1.1 GB の XML 文書を利用し，生成した文書は図 10 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あらかじめ取り込んだ．

4.1 値渡しとの比較

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 [14] バージョン 4.18.1 との性能比較を行った．*XBird/D* の評価には，図 10 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．図 10 の$length 変数には，1,000 を設定した．

実験結果の概要が，**図 12** 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[14]（XRPC）の 4 つの評価戦略の比較を行った．

図 12 にあるように，我々の remote proxy を利用した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リモート

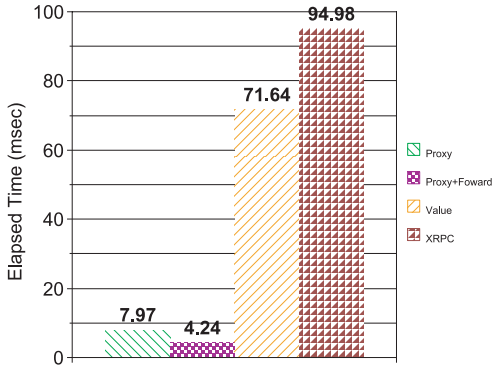
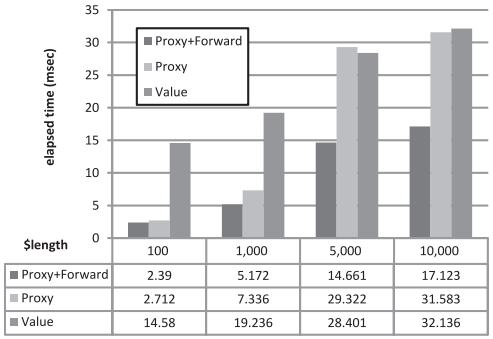


図 12　4 つの実行戦略の性能比較
Fig. 12　Comparison of four evaluation strategies.



$length	100	1,000	5,000	10,000
Proxy+Forward	2.39	5.172	14.661	17.123
Proxy	2.712	7.336	29.322	31.583
Value	14.58	19.236	28.401	32.136

図 13　レスポンス時間に影響されるパラメータを変化させた実験
Fig. 13　An experiment that changes influential parameters to the response time.

問合せの評価は，完全な結果を 1 度に計算・生産するが，それらのすべ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している．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ヘッドを削減した．その結果，提案システムは競合実装である 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（図 10）を 160 秒で処理することができた．このときの最大実行時間は 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では，頻繁なスワップイン/スワップアウトが発生する（Value および XRPC），900 秒以上の遅延が発生し実験遂行に支障が出る（XRPC）など，リソース活用に貧弱さを示した．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．

4.2　レスポンス時間に影響を与えるパラメータを変化させた実験

図 10 の$length 変数は，レスポンス時間の重要性を左右するパラメータである．$length 変数の値が小さいほど，短いレスポンス時間で結果を返す問合せプランが有効となる．本節では，$length 変数の値を 100 からすべての生産されるタプルを消費する値まで変化させることで，値渡しと参照渡しの有効な場面を検証する．

実験では$length 変数の値として，100〜10,000 の間で 4 つの値を利用し，提案システム上に実装した Proxy，Proxy+Forward，Value の 3 つの評価戦略を評価した．また，図 10 の (a) と (b) の箇所に，それぞれ closed_auction のプライスが 300 以上のもの（price/text() >= 300），open_auction の初期値が 300 以上のもの（initial/text() >= 300）を選択する述部を挿入した[⋆1]．これにより，97,500 個のうち 4,359 個の closed_auction と 12,000 個のうち 6,303 個の open_auction がそれぞれ選択される．

図 13 が，その実験結果である．$length 変数の値が 100 か 1,000 と小さいとき，すなわち，レスポンス時間が重要となる問合せにおいて remote proxy を利用した提案手法が有効に機能していることが分かる．$length 変数の値が 5,000 と 10,000 の実験では，それぞれタプル生産数と消費数が近い関係にあるか，まったく同じである．このような問合せにおいては，一般的にレスポンス時間よりも総計算時間が重要となるが，このようなケースにおいても remote proxy を利用した参照渡し戦略は値渡しと遜色なく機能している．個々のノードにおける計算量が大きい，$length 変数が 10,000 のケースでは，複数回の RPC が発生する remote proxy の方が，値渡しに勝るケースも見られた．これは，値渡しでは計算機ノードをまたいだ計算が逐次的となるのに対して，remote proxy によるパイプライン並列が有効に機能した結果である．参照渡しは$length=∞ とすると値渡しに近似するが，$length 変数が大きいほど逐次処理による待機が発生するためパイプライン処理が有効に機能する（たとえば，$length>=10,000 の場合）ことと，$length 変数が小さいほどレスポンス時間

⋆1 値渡しを利用した場合にメモリ不足が発生するための措置である．

が重要になることに留意されたい．図 13 の設定を例にとると，「生産されるタプルの消費率が高い」∩「生産されるタプル数が 10,000 未満」といった条件が成り立つ場合（たとえば，$length=5,000 の場合）に値渡しが有効に機能するが，参照渡しに対する値渡しの優越は小さい．次の 4.3 節で，*tuples-at-a-time* におけるデータ交換時のタプルフェッチ数とレスポンス時間の関係に考察を与える．

4.3　タプルフェッチ数とレスポンス時間

提案システムでは，問合せごとに計算結果に含まれるタプル数の取得方法を指定し，ネットワークの状況やメモリの空き状況，アプリケーションの形態に応じた計算結果の取得を行うことができる．

本節ではタプル転送量の指標を示すために，システムのデフォルトの設定を利用した場合のタプル交換数とレスポンス時間の関係を述べる．さらに，remote proxy を利用したデータ交換によって生じるオーバヘッドについて考察を与え，remote proxy のオーバヘッドが許容できるほど小さいことを示す．3.3.2 項で言及した計算結果タプルの初期取得サイズと取得サイズの伸張係数のデフォルト設定は，それぞれ，256 個，1.3 倍である．RPC が発生することに上限閾値までタプル取得サイズは増長する．

実験では，2 章ですでに利用した図 3 で示す問合せを遠隔サイトで実行し，問合せを発行したクライアントが remote proxy によってその計算結果である実体シーケンスから結果タプルを取得した．ここでは，1 回のタプル取得数の上限閾値を設けずに取得タプル数とレスポンス時間の関係を見ていく．なお，データセットには XMark の SF=10 で生成した XML 文書を利用した．

図 14 に，RPC により取得されたタプル取得数と実行時間の関係を示す．1 度目の RPC では，256 個のタプルが取得され，そのレスポンス時間として 0.143 秒がかかっている．累積時間がすでにあるのは，3.2 節におけるステップ (3) 以前のコンパイル処理などに要した時間が加算されているためである．初回の RPC については，Volcano イテレータモデルの初期化処理によって多少のオーバヘッドが存在する．このケースでは，remote proxy によるパイプライン処理によって，計 17 回の RPC によるデータ交換が発生するが，問合せは 6 秒弱で終わる．なお，図 3 の問合せをローカル計算機上で実行したときの計算時間は，5.658 秒である．このことから，remote proxy によるオーバヘッドが十分に小さいことが分かる．タプル取得数とレスポンス時間の相関は，図 14 のフェッチごとのレスポンス時間に示すとおりである．このケースでは 13,016 タプルの取得を 1 度に行う場合に，1 秒近いレスポンス時間がかかることが分かる．提案システムを利用するユーザはこのような指標と

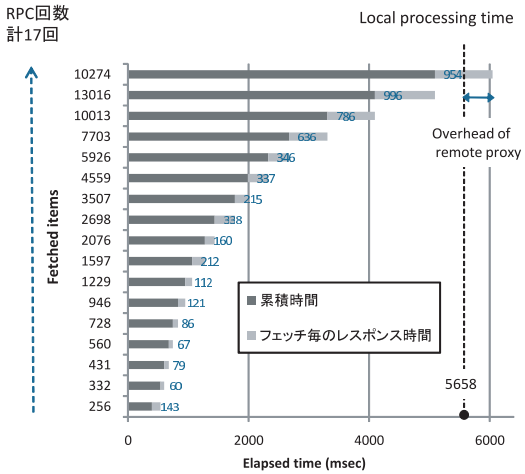


図 14　Remote proxy によるオーバヘッド
Fig. 14　Overhead of remote proxy.

アプリケーションのデータアクセス傾向を考慮したタプル取得戦略を決定することで，理想に近いレスポンス時間を実現することができる．

5. 関 連 研 究

論文 24) で，我々は remote proxy のプロトタイプシステムを現実的なアプリケーションである複数のバイオデータベースシステム群の統合に適用した．Abiteboul らは，XML データを自動的に分散・複製配置し，その分散した XML データを XQuery プロセッサを内部的に利用する Web サービスによって透過的に統合するシステムを提案している[25]．

DXQ[26] は，論理的な実行プランを遠隔サーバと交換することで分散コンパイル処理を可能とする分散 XQuery 処理系である．論理的な実行プランを分散コンパイル処理に利用するのは，提案手法をはじめ，既存の分散データベースシステムにおいても利用される．DXQ は，本研究と異なり，ノード間のデータ交換については XML 形式の値渡しのみがサポートされる．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実行基盤として，分散 XQuery 問合せ処理機 XRPC[27] を提案している．XRPC は複数のリモート関数呼び出しを 1 つにまとめて，*set-at-a-time* 形式で 1 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．XRPC

ではデータ交換はすべ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．レスポンス時間がそれほど重要でない値渡しが有効な場面で，BulkRPC は有効に機能するため，参照渡しを利用する提案手法とは相補的な関係にある．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する分散問合せ処理言語を提案している[28]．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[29]．我々の採用する remote proxy を問合せ処理は，これらの分散問合せ処理パターンでは議論されてこなかったパイプライン並列をサポートする．

オペレータ間のリソース割当てを効率的に行う研究の代表的なものに，RateMatch アルゴリズム[30] 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当てが行われる．オペレータ間のリソース割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おいても重要である．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検査し，コスト計算を行うことによって静的に計算資源の割当てを行う．RateMatch の静的な資源割当ては，コストの見積り精度に依存するものであり，急激な計算資源の状況変化に対して弱い．これに対し，提案手法は，3.3.2 項で述べたように *remote blocking-queue* に拠る受注生産方式を採用することで，動的な計算資源の割当てを実現した．

6. 結　　論

本論文では，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，提案手法が値渡しを利用する既存手法に対して最大 22 倍の性能向上を示すことがあることを明らかにし，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（たとえば，CPU の利用率やメモリの空き状況やスペック）を考慮に入れたリモートの問合せ処理器への動的な実行の割当てと実行戦略の選択モデルの開発があげられる．なお，本論文で示した提案手法を満たす処理系はコード公開する[⋆1]．

⋆1 http://code.google.com/p/xbird/

謝辞　MonetDB/XRPC の実験を遂行するうえで助言を与えてくれたオランダ国立数学計算機科学研究所（CWI）Ying Zhang 氏に感謝する．本研究の一部は，日本学術振興会特別研究員奨励費，科学技術振興機 CREST「情報社会を支える新しい高性能情報処理技術」，ならびに科研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」（課題番号：19024058），若手（B）（課題番号：19700094）による．ここに記して謝意を表す．

参 考 文 献

1) Kurita, H., Hatano, K., Miyazaki, J. and Uemura, S.: Efficient Query Processing for Large XML Data in Distributed Environments, *Proc. AINA*, Vol.0, pp.317–322, IEEE Computer Society (2007).
2) Bremer, J.-M. and Gertz, M.: On Distributing XML Repositorie, *Proc. WebDB*, pp.73–78 (2003).
3) 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評価，日本データベース学会 letters，Vol.5, No.2, pp.85–88 (2006).
4) Dean, J. and Ghemawat, S.: MapReduce: Simplified Data Processing on Large Clusters, *Proc. OSDI*, pp.137–150 (2004).
5) W3C: XQuery 1.0 and XPath 2.0 Data Model (XDM). http://www.w3.org/TR/xpath-datamodel/
6) Nicola, M. and John, J.: XML Parsing: A Threat to Database Performance, *Proc. CIKM*, pp.175–178 (2003).
7) Ozsu, T.M. and Valduriez, P.: *Principles of Distributed Database Systems*, *2nd ed.*, Prentice Hall (1999).
8) Shekita, E.J., Young, H.C. and Tan, K.L.: Multi-Join Optimization for Symmetric Multiprocessors, *Proc. VLDB*, pp.479–492 (1993).
9) Lanzelotte, R.S.G., Valduriez, P. and Zäit, M.: On the Effectiveness of Optimization Search Strategies for Parallel Execution Spaces, *Proc. VLDB*, pp.493–504 (1993).
10) Amdahl, G.M.: *Validity of the single processor approach to achieving large scale computing capabilities*, Morgan Kaufmann Publishers Inc. (2000).
11) Rohnert, H.: *The proxy design pattern revisited: Pattern languages of program design 2*, Addison-Wesley, Inc. (1996).
12) W3C: XQuery 1.0: An XML Query Language. http://www.w3.org/TR/xquery/
13) Ré, C., Brinkley, J., Hinshaw, K. and Suciu, D.: Distributed XQuery, *Proc. IIWeb*, pp.116–121 (2004).
14) Zhang, Y. and Boncz, P.A.: XRPC: Interoperable and Efficient Distributed

XQuery, *Proc. VLDB* (2007).
15) Fernandez, M.F., Jim, T., Morton, K., Onose, N. and Simeon, J.: DXQ: A Distributed XQuery Scripting Language, *Proc. XIME-P* (2007).
16) 油井　誠，宮崎　純，植村俊亮：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，Vol.48, pp.128–148 (2007).
17) Schmidt, A.R., Waas, F., Kersten, M.L., Florescu, D., Manolescu, I., Carey, M.J. and Busse, R.: The XML Benchmark Project, Technical Report INS-R0103, CWI (2001).
18) Bayardo, R.J., Gruhl, D., Josifovski, V. and Myllymaki, J.: An Evaluation of Binary XML Encoding Optimizations for Fast Stream Based XML Processing, *Proc. WWW*, pp.345–354 (2004).
19) Graefe, G.: Volcano—An Extensible and Parallel Query Evaluation System, *IEEE Trans. Knowledge and Data Engineering*, Vol.6, No.1, pp.120–135 (1994).
20) Graefe, G.: Encapsulation of Parallelism in the Volcano Query Processing System, *Proc. SIGMOD*, pp.102–111 (1990).
21) Boncz, P.A., Zukowski, M. and Nes, N.: MonetDB/X100: Hyper-Pipelining Query Execution, *Proc. CIDR*, pp.225–237 (2005).
22) Milo, T., Abiteboul, S., Amann, B., Benjelloun, O. and Ngoc, F.D.: Exchanging Intensional XML Data, *ACM Trans. Database Syst.*, Vol.30, No.1, pp.1–40 (2005).
23) Buneman, P., Cong, G., Fan, W. and Kementsietsidis, A.: Using Partial Evaluation in Distributed Query Evaluation, *Proc. VLDB*, pp.211–222 (2006).
24) 油井　誠，宮崎　純，植村俊亮，加藤博一：Remote Proxy を利用した並列分散 XML 問合せ処理手法の提案，電子情報通信学会技術研究報告，Vol.107, No.131, pp.217–222 (2007).
25) Abiteboul, S., Bonifati, A., Cobéna, G., Manolescu, I. and Milo, T.: Dynamic XML documents with distribution and replication, *Proc. SIGMOD*, pp.527–538 (2003).
26) Fernández, M., Jim, T., Morton, K., Onose, N. and Simeon, J.: Highly Distributed XQuery with DXQ, *Proc. SIGMOD* (2007).
27) Zhang, Y. and Boncz, P.A.: Integrating XQuery and P2P in MonetDB/XQuery*, *Proc. Workshop on Emerging Research Opportunities for Web Data Management* (*EROW*) (2007).
28) Sahuguet, A. and Tannen, V.: ubQL, a Language for Programming Distributed Query Systems, *Proc. WebDB*, pp.37–42 (2001).
29) Papadimos, V. and Maier, D.: Distributed Queries without Distributed State, *Proc. WebDB*, pp.95–100 (2002).
30) Mehta, M. and Dewitt, D.J.: Managing Intra-operator Parallelism in Parallel Database Systems, *Proc. VLDB*, pp.382–394 (1995).

(平成 20 年 9 月 21 日受付)
(平成 21 年 1 月 9 日採録)

（担当編集委員　鬼塚 真）

油井　誠（学生会員）

2003 年芝浦工業大学工学部工業経営学科卒業．同年（株）NEC 情報システムズ入社．グリッド・コンピューティングの研究開発に従事．2006 年奈良先端科学技術大学院大学情報科学研究科博士前期課程修了．同年 10 月より同大学情報科学研究科博士後期課程，現在に至る．日本学術振興会特別研究員 DC2．日本データベース学会会員．

宮崎　純（正会員）

奈良先端科学技術大学院大学情報科学研究科准教授．博士（情報科学）．1992 年東京工業大学工学部情報工学科卒業．1997 年北陸先端科学技術大学院大学情報科学研究科博士後期課程修了．同大学助手を経て，2003 年より現職．2003〜2007 年科学技術振興機構さきがけ研究員（兼務）．2000〜2001 年テキサス大学アーリントン校客員研究員．高性能・高機能データ工学システムならびに情報検索の研究に従事．電子情報通信学会，日本データベース学会，IEEE CS，ACM 各会員．

植村　俊亮（フェロー）

奈良産業大学情報学部情報学科教授．1966 年京都大学大学院工学研究科修士課程修了．同年電気試験所（現，産業技術総合研究所）．マサチューセッツ工科大学電子システム研究所客員研究員，東京農工大学教授，奈良先端科学技術大学院大学教授を経て，2007 年から現職．データ工学，データベースシステムの研究に従事．工学博士．IEEE Fellow，電子情報通信学会フェロー．

加藤　博一（正会員）

奈良先端科学技術大学院大学情報科学研究科教授．博士（工学）．1986 年大阪大学基礎工学部制御工学科卒業．1988 年同大学大学院修士課程修了．1989 年同大学基礎工学部助手．1996 年講師．1998 年ワシントン大学客員研究員．1999 年広島市立大学情報科学部助教授．2003 年大阪大学大学院基礎工学研究科助教授を経て，2007 年より現職．拡張現実感，ヒューマンインタフェースの研究に従事．ヒューマンインタフェース学会，日本 VR 学会，ACM 等各会員．