インクリメンタルに更新可能な状態遷移表を用いたXPushマシン

XPush Machines Using an Incrementally Updatable State Transition Table

武川 肇[†1] 片山 薫[†2] 石川 博[†3]

Hajime TAKEKAWA Kaoru KATAYAMA
Hiroshi ISHIKAWA

XPathフィルタエンジンはフィルタ条件を数多く評価する必要がある．そのためGuptaらは複数のフィルタをボトムアップに結合し，フィルタ条件の数に影響しないオートマトン，XPush マシンを提案した．XPushマシンは状態数の増加を防ぐため，XMLデータの評価時にそのデータ構造に基づき状態遷移表を遅延構築する．しかしデータの構造が変化しやすい環境ではオートマトンの状態数が増え続けるためにメモリコストが増大する．この問題を解決するために，問合せごとに構築したサブ XPush マシンの集合から全体のオートマトンを構築する方式を提案する．提案する本方式では部分フィルタの結合と分離に必要な状態遷移表の更新処理がインクリメンタルに計算できる．

XPath filter engines need to evaluate many filter conditions. Therefore, Gupta et. al. proposed the automaton called XPush machine which does not influence the number of filter conditions by combining two or more filters in a bottom up fashion. An XPush machine carries out lazy construction of the state transition table based on the data structure at the time of evaluation of XML data in order to prevent the increase in the number of states. However, the required memory space increases because the number of an automaton increases in the environment where the structure of data changes rapidly. In order to solve this problem, we proposed a method which constructs the whole automaton from a set of sub-XPush machines for each query. In the system the updating process of a state transition table can be incrementally calculated for addition and separation of partial filters.

1. はじめに

XMLデータをコンテンツベースに処理するフィルタ型配信システムやXMLルータは，フィルタ条件を数多く評価する必要がある[7]．これらシステムのスループット向上のため，Guptaらは遅延型の単純決定性プッシュダウンオートマトンに基づき，複数のフィルタをボトムアップに結合したXMLフィルタエンジン用オートマトン，XPushマシンを提案した[8]．

†1 正会員 職業能力開発総合大学校情報工学科
takekawa@uitec.ac.jp
†2 正会員 首都大学東京システムデザイン学部，東京都立大学大学院・工学研究科 katayama@eei.metro-u.ac.jp
†3 正会員 首都大学東京システムデザイン学部，東京都立大学大学院・工学研究科 ishikawa@eei.metro-u.ac.jp

XPushマシンではフィルタ条件をXPath[5]で記述し，この条件をXPathフィルタという．XPathフィルタはナビゲーション部とプレディケート部で構成されている．例1にXPushマシンで利用するXPathフィルタの例を示す．

例1 [Running example]
P1 : //a[@b<20]
P2 : //a[@b>=10 and @b<20]

“[ ]”で囲まれた部分がプレディケート部であり，そこには条件を記述する．プレディケート部より前の部分をナビゲーション部という．“//a”はナビゲーション部であり，上記2式ともナビゲーション部は同じである．“//”はワイルドカードであり，“//a”はaタグがXMLデータのどの位置に現れてもよいことを表す．“@”は属性を意味し，“@b”は属性bを表す．数値の10と20は引用符で囲まれていないため，XMLデータの属性bの値は数値型のデータとして評価される．

XPathの評価にはDOM型のXMLプロセッサがよく利用される．ここでDOM型とはXMLデータ全体を一時的にメモリに読込み処理する方式を意味する．XMLプロセッサによるXPath評価は，XMLデータサイズとXPathフィルタ数の双方に比例する．一方XPushマシンの処理速度はデータサイズに比例するだけであり，フィルタ数にはほとんど影響を受けない．

例えば例1でXPushマシンを利用すると，XPushマシンは2つのフィルタ条件を予めボトムアップに結合し1つの状態遷移表に変換するため，文書ごとに1回の逐次的なアクセスだけで2つ全てのフィルタ条件評価をカバーできる．

逐次的に処理するためにXPushマシンはSAX型のXMLパーサを利用する．SAX型のパーサはXMLデータをストリームとして扱い，XMLデータのタグや値ごとにイベントをアプリケーションに通知する．アプリケーションは各イベント時に渡される断片的なXMLデータによって処理を行う．一般にSAX型はXMLデータサイズに制限されない利点を持つ．実際XPushマシンのスタック制御部はXMLデータの深さ分のスタックしか利用しない．

しかしGuptaらが提案するXPushマシンは初期化ができるが，状態遷移表の部分削除の機能が実現されていない．XPushマシンの更新手段が未解決である．そのために評価データの構造が変化しやすい環境ではオートマトンの状態数が増え続け，システムのメモリを使い果たす問題がある．

ここでメモリコストをコントロールする必要があるシステム，例えば利用者が多いデータベースシステム上の問い合わせエンジンとしての応用を考える．メモリコストを下げるため，XPathフィルタ集合から使用頻度の低いものに対して削除処理が計画されたとする．XPushマシンは1つのフィルタ条件を削除することであっても，XPushマシン全体の再計算が必要となる．初期状態の非決定性有限オートマトンに戻ったXPushマシンは決定性有限オートマトンへの遅延変換を要求する．そのためメモリコスト削減のために状態遷移表をランダムに部分削除すると遅延変換が多発しスループットを低下させる恐れがある．仮にスループットが低下してもよいとしても，データベースなどの同じデータ構造を評価する環境では，状態遷移表を部分削除しても，次回のデータ評価時において状態遷移表から削除したはずの状態遷移情報が直ぐに再構築される．よってメモリコストの削減にならない．状態遷移表の部分削除を成功させるためには，不要な内部状態を特定し，削除する必要がある．

本稿ではこの未解決問題のために，XPathフィルタグループごとに構築した部分状態遷移表（サブXPushマシンと呼ぶ）の集合から，全体のオートマトンを構築する方法を提案する．

なおXPathフィルタグループには，問合せ[1][9][10]やセッション単位を想定している．以降では，このサブXPushマシン集合から構築した全体のオートマトンを統合型XPushマシンと呼ぶ．

統合型XPushマシンは，従来方式のXPushマシンにフィルタの結合と分離の2つの機能を追加する．その仕組みは次の通りである．

統合型XPushマシンはコントローラモジュールとのインターフェースが従来どおりであり，従来方式と同じスループット性能を持つ．そしてコンパイル後のオートマトンはグループごとに分割管理された状態遷移表（サブXPushマシン）となっている．そのためフィルタグループ単位で結合と分離が可能である．サブXPushマシンのコンパイラには，GuptaらのXPushマシンのコンパイラをほぼそのまま利用する．このコンパイラは状態ごとにキーを追加する修正が施されるだけである．このキーは状態どうしのさらなる結合に利用される．

XMLストリームを利用したXMLフィルタの研究の多くは，ナビゲーション部を利用する[3][6]．それらはXML構造が木構造であることに基づき，XPathフィルタ間の同じナビゲーション部評価の削減計画を行い，効率化が図られる．しかしこの手法では，XMLデータの断片がプレディケート部について未評価の状態で抽出される．

一方，本方式はGuptaらのXPushマシンと同様に，ナビゲーション部だけではなくプレディケート部評価の効率化を行っている．

本稿の構成は以下の通りである．この節の残りでは関連する研究について述べる．続く2節で，提案方式について述べ，3節ではフィルタの部分交換に関する実験概要とその結果を示す．

2. 提案方式

この節では本提案方式の統合型XPushマシンについて述べる．

2.1 統合型 XPush マシンの概要

ここでは統合型XPushマシンの概要について述べる．

図1に統合型XPushマシンを用いたXPathフィルタエンジンのアーキテクチャを示す．エンジンは従来方式と同じくオートマトンモジュールとコントローラモジュールで構成されている．

オートマトンモジュールではXPathフィルタから問合せ木（構文木）への変換をXPathパーサによって処理した後に，それをさらにコンパイラによってサブXPushマシンへと変換する．図1でM1～M5はサブXPushマシンを表す．サブXPushマシンはXPathフィルタグループ単位に作成される．現在フィルタグループには問合せやセッション単位を想定している．コンパイラはGuptaらのXPushマシンのコンパイラとほぼ同じであり，AFA（Alternating Finite Automaton）[4]に基づいている．サブXPushマシンは従来方式のXPushマシンの定義[8]と同じである．ただし本提案手法はXPushマシンのオプション機能のトップダウンプルーニングを考慮していない．

コンパイラとスタック制御部の間には統合処理部が存在する．このことはGuptaらの手法と大きく異なる点である．統合処理部はコンパイル結果の統合型XPushマシンへの結合と，さらに外部記憶を利用したサブXPushマシン単位の分離と再結合を処理する．例えば図1ではXPathフィルタP3をM3にコンパイルした後，統合処理部によって，統合型XPushマシンがインクリメンタルに更新される．そしてM4のidを利用し

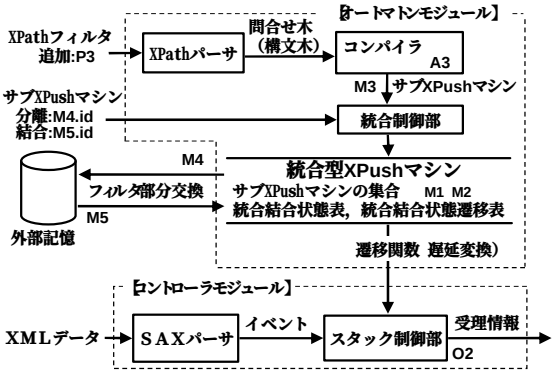

図 1 統合型 XPush マシンを用いた XPath フィルタエンジン
Fig.1 An XPath filter engine using an integrated XPush Machine.

て分離命令を統合処理部に伝えると，M4を統合型XPushマシンから分離後，外部記憶に保存される．さらにM5の結合命令によってサブXPushマシンM5を復元し，統合型XPushマシンとの再結合が行われるという流れを示している．

コントローラモジュールは，コンパイラによって作成した状態遷移表に従ってXMLデータの評価を行う．状態遷移表の参照は遷移関数を利用する．このとき必要があれば遅延変換処理が発生する．

遷移関数は統合結合状態を参照する統合結合状態遷移表からの表引きに変更している．これによって，オートマトンモジュールとコントローラモジュールが従来方式と互換性を保つため，XPath フィルタエンジンのスループットは同じである．

2.2 結合状態キー

ここでは結合状態キーの作成方法について述べる．

図 2 に統合型 XPush マシンのデータ構造を示す．図 2a,図 2b は，それぞれ例 1 の P1,P2 の AFA をボトムアップ結合したサブ XPush マシン M1,M2 である．サブ XPush マシンの内部状態は，従来方式の XPush 状態と呼ばれる内部状態と異なり，状態ごとにキーがある．以降ではこのキーを結合状態キー，また結合状態キーを持つ XPush 状態を結合状態と呼ぶ．本方式では，統合制御部が結合状態表とサブ XPush マシンを結合する時に，この結合状態キーを利用する．

結合状態キーは，サブ XPush マシンの状態遷移表の構築処理に用いられた AFA 上の遷移に基づいて計算されている．なおサブ XPush マシンの状態遷移表の構築処理は，従来方式の XPush マシンの状態遷移表の構築処理と同じである．

例えば，ある結合状態 q1 の AFA 状態 s ごとに pop(s,σ) 遷移をさせた AFA 状態の集合が，結合状態 q2 として新規に作成されるとき，q2 の結合状態キーは「pop q1,σ」となる．

図 2c は，M1,M2 を結合した統合型 XPush マシンである．M3 には 15 個の統合結合状態が存在し，それら各々がサブ XPush マシンごとに 1 つの XPush 状態への参照が存在している．同様に統合結合状態表にもキーが存在し，これを統合結合状態キーという．

2.3 更新処理

ここでは統合型 XPush マシンの更新処理について述べる．

まず，分離処理の手順を以下に示す．

1) 統合結合状態表に対して，削除するサブ XPush マシンへの参照を除いた Q 参照のリストでグルーピングする．それらグループごとに ID 番号の最小値を持つ状態 xg を選び，各グループで xg 以外を削除対象 xd とする．

2) 全ての xd を X から削除する．そして統合結合状態遷移表から xd をキーとした情報を削除する．
3) 統合型 XPush マシンにある全ての X 参照 xr ごとに，もし xr が削除対象 xd を参照しているなら，xr を 1) で xd と同じグループであった xg への参照に修正する．
4) サブ XPush マシンをディスクに保存する．

例えば図 2c の統合型 XPush マシンから M2 を分離する場合，以下の xd から xg への参照修正リストが得られる．

x5 -> x0　　x8 -> x3　　x11 -> x4　　x14 -> x1
x6 -> x0　　x9 -> x4　　x12 -> x3
x7 -> x2　　x10 -> x3　　x13 -> x4

上記リストに従って，削除対象の状態を統合型 XPush マシンから切り離す．図 2d の統合型 XPush マシンが得られる．

また図 2c の統合型 XPush マシンから M1 を分離する場合，以下の xd から xg への参照修正リストが得られる．

x3 -> x0　　x4 -> x2　　x8 -> x6　　x9 -> x7

このとき手順 3)の参照修正を行わずに削除処理を終えると，pop(x2，a)に不正な参照 x3 が残る．そのため手順 3)は省略できない．

次に，結合処理の手順を以下に示す．

入力：結合する新規または分離したサブ XPush マシン

1) 結合アルゴリズム[2]を使い，関係データベースにおける表結合のように統合結合状態キーと世代関係を計算する．統合結合状態キーと世代関係から新規統合結合状態を X に追加する．
2) 世代関係を利用して，Tpop，Tadd を修正する．

ここで世代関係とは，新規統合結合状態を結合処理前の統合結合状態と結合中のサブ XPush マシンの結合状態で表した組をいう．また結合処理中において，結合前の統合型 XPush マシンを前世代という．

例えば図 2d の統合型 XPush マシンに M2 を追加する場合，結合アルゴリズムを利用して図 2e の M1+M2 の世代関係が得られる．この世代関係は例えば x9 に関して次のことを表す．

x9 は前世代の x4 と M2.q5 から状態遷移表 Tpop と Tadd がインクリメンタルに更新可能である．

実際図 2.c の Tpop(x9,a)は，次のように計算できる．

Tpop(x9,a) => (Tpop(前世代の x4,a),M2.Tpop(M2 の q5,a))
=> (前世代の x3,q6) => (q3,q6) => x10

3. 実験

ここでは統合型XPushマシンのフィルタ部分交換に関する実験概要と実験結果を示す．

3.1 実験概要

実験は提案方式のフィルタエンジンを Java で実装し，これを Red Had Linux 8.0 上で使用した．使用したコンピュータの CPU は PentiumⅣ3.3GHz，メモリ 2G である．XML パーサには xerces2.6.2（http://xml.apache.org）を利用した．また比較のために従来方式の XPush マシンも Java で実装し，同じ環境で測定した．

実験データには Gupta らの実験でも用いられた Protein（http://pir.georgetown.edu）を利用した．Protein は 700MB あるデータセットであり，これを 9.12MB のサイズに分割したものを用いた．XPath フィルタは XML データからランダムに作成するプログラムを作成し，ナビケーション部の長さが 1～3 でプレディケート部が 1～8 の XPath を 10,000 件生成し，各実験において必要数分だけ利用した．

予備実験1：XPath フィルタ 1,000 件から従来型の XPush

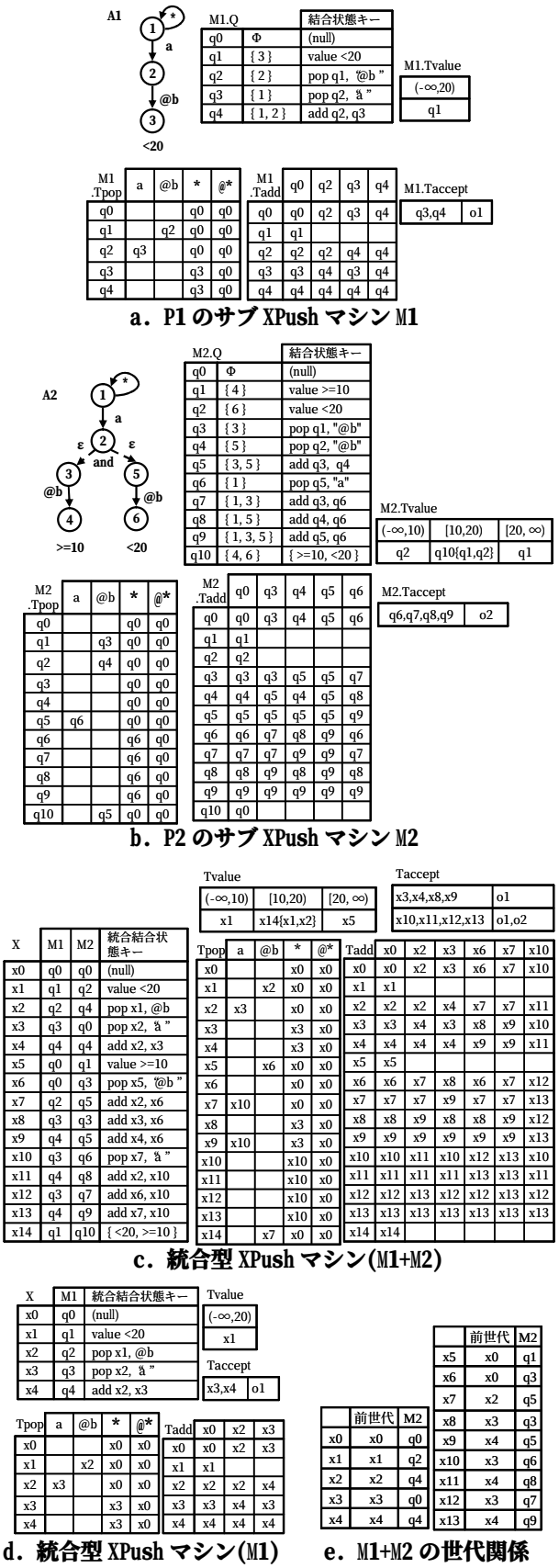

図 2　統合型 XPush マシンのデータ構造
Fig.2　Data structures of integrated XPush Machines.

マシンを生成し，データ 9.12MB にかかる評価時間を測定した．結果は再コンパイル後の 1 回目の評価時間が約 352 秒であり，2 回目は 8 秒であった．よって従来方式では総フィルタ数 1,000 件のとき，遅延変換時間が 344（=352－8）秒であった．(従来方式における遅延変換時間＝再コンパイル後 1 回目の評価時間-再コンパイル後 2 回目の評価時間)

予備実験 2：まず 1 フィルタグループあたりのフィルタ数（以降，グループサイズ）が 10 間隔で 10 から 200 までの XPath フィルタを各サイズ 1 つずつ計 20 組用意した．次にそれら XPath フィルタを結合して 1 つの統合型 XPush マシンを生成した．そして 1 回だけ予備実験 1 と同じデータを評価させた後に，すべてのフィルタグループを分離し，各フィルタグループ（サブ XPush マシン）をディスクに保存した．

実験 1：まず提案方式で予備実験 1 と同じ 1,000 件の XPath フィルタをグループサイズが 10（×100 組），20（×50 組），･･･，200（×5 組）でコンパイルした後，1 つに結合して統合型 XPush マシンを作成した．なおグループサイズが 30 のときのように 1,000 を割り切れない場合は，その余りを 1 つのフィルタグループとした．例えば 30 の場合，30×33 組＋10×1 組として用意した．次に 1 回だけ予備実験 1 と同じデータを評価させた．その後，予備実験 2 で保存したフィルタでグループサイズが同じものをディスクから読み込み，統合型 XPush マシンからランダムに 1 つのフィルタを部分交換した．そしてフィルタ部分交換後に交換前と同じデータを 2 回評価させ測定した．なお評価に利用したデータは予備実験 1 と同じものを使った．実験の評価は従来方式のフィルタ遅延変換時間（344 秒）を基準に比較をした．（統合型方式における遅延変換時間 = フィルタ部分交換後 1 回目の評価時間 - フィルタ部分交換後 2 回目の評価時間）

3.2 実験結果

実験 1 の結果を図 3 に示す．統合型 XPush マシンにおけるフィルタ部分交換後の 1 回目の評価時間はグループサイズが増えると，評価時間が増加する傾向があった．グループサイズが 10 のとき最小 10 秒であり，200 のとき最大 21 秒であった．2 回目の評価時間は，どのグループサイズであって従来方式と同じ約 8 秒であった．よって遅延変換時間は従来方式に比べ，グループサイズが 10 で 1%以下（=(10 秒-8 秒)/344 秒)であり，200 のときで 4%以下(=(21 秒-8 秒)/344 秒)であった．したがって統合型 XPush マシンの統合結合状態表および統合結合状態遷移表は更新されても，従来方式の全フィルタの再コンパイル（再計算）に比べ，遅延変換処理時間をほとんど要求せず，更新に非常に強い特性を持っているといえる．

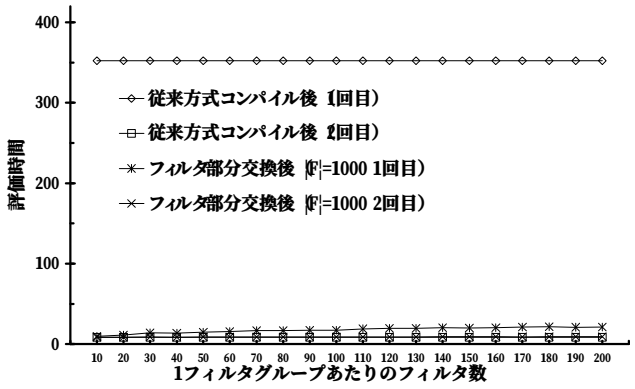

図 3 フィルタ部分交換後のデータ評価時間
fig3 Data evaluation time after filter partial exchange.

4. おわりに

本研究では，フィルタの部分交換ができる統合型 XPush マシンを提案した．さらに統合型 XPush マシンの更新の計算コストが従来の XPush マシンの再計算の計算コストより低いことを実験結果で示した．

今後の課題として，統合型 XPush マシンのトップダウンプルーニングへの対応を検討している．

[文献]

[1] 澤井里枝,塚本昌彦,寺田努,西尾章治朗 ： フィルタリングのためのユーザ要求記述言語 FilteringSQL について, DBS 2003.
[2] 武川肇,片山薫,石川博 ： インクリメンタルに更新可能な XPush マシン, DBWS2005.
[3] 森川裕章,浅井達哉,有村博紀 ： データストリーム処理のための効率良い XPath 問合せ機構, DBWS 2003.
[4] Chandra,A. Kozen,D. Stockmeyer,L.： Alternation. In journal of the ACM, pp. 115–133, January 1981.
[5] Clark,J. ： XML path language(XPath). 1999. http://www.w3.org/Tr/xpath.
[6] Diao, Y. Fischer, P. Franklin, M. To, R. ： Yfilter: Efficient and scalable filtering of XML documents. In Proc. ICDE, pp. 341?344, February 2002.
[7] Green,T.J. Miklau,G. Onizuka,M. Suciu.D. : Processing XML streams with deterministic automata. In Proc. ICDT, pp. 173–189, 2003.
[8] Gupta,A. Suciu,D. ： Stream Processing of XPath Queries with Predicates. In Proc. SIGMOD, pp. 419–430 , 2003.
[9] Peng,F. Chawathe,S.S. : XPath Queries on Streaming Data. In Proc. SIGMOD, pp431-442, 2003.
[10] XQuery 1.0: An XML Query Language. http://www.w3.org/TR/xquery.

武川 肇 Hajime TAKEKAWA
平成 3 年職業訓練大学校情報工学科卒業. 雇用促進事業団(現雇用・能力開発機構)入社．平成 15 年より職業能力開発総合大学校勤務．東京都立大学大学院研究生として XML データベースの研究に従事．情報処理学会，日本データベース学会各会員.

片山 薫 Kaoru KATAYAMA
東京都立大学大学院工学研究科助手．2000 年京都大学大学院情報学研究科社会情報学専攻博士後期課程了．博士(情報学)．データベースシステムに関する研究開発に従事．情報処理学会，日本データベース学会各会員.

石川 博 Hiroshi ISHIKAWA
東京都立大学大学院工学研究科教授．富士通研究所を経て 2000 年より現職．1979 年東大・理卒．博士（理学）．データベースシステムの研究開発に従事．情報処理学会，ACM，IEEE 各会員．International Journal on Very Large Data Bases Editorial Board，ACM Sigmod Japan 評議員，日本データベース学会理事．情報処理学会データベースシステム研究会主査．情報処理学会論文誌(データベース)編集委員長．ナント大学(フランス)エコールポリテクニック招聘教授．著書に「E－ビジネス技術入門教科書」「次世代データベースとデータマイニング」(CQ 出版)など.