XQuery 問い合わせ処理の最適化に関する研究
－ 結合処理の最適化 －

2005MT009 江崎 圭介　　2005MT100 関谷 剛史　　2005MT111 高橋 知宏
指導教員　野呂 昌満

1　はじめに

XML 文書に対する問い合わせ言語として XQuery が提案されている．既存の XQuery 処理系の多くは，XML 文書を木構造データに変換した後で木に対して問い合わせ処理を行っているので，XML 文書が大規模であるほど多量な処理時間とメモリ使用量がかかることが問題となっている．

その解決策として我々の研究室で提案された xqgen[2] では，XML 文書の構文解析によって問い合わせに必要な要素のみの木を作成することで質問式の大部分の処理時間が削減された．しかし，結合処理においては，異なる木の要素どうしを全て照合してしまうので，その処理時間がかかることが問題となる．その問題に対し我々の研究室では，結合処理をハッシュ法を用いて高速化する方法を示した [3]．2 重の for 節の内側と外側の for 節で取得した要素間の同値比較を行う質問式 (XMarkQ8) について実験を行い，ハッシュ法が適用できることが確認されたが，ハッシュ法が適用可能な結合処理の質問式について系統的な整理がされていない．また，3 重以上の for 節の質問式に対するハッシュ法の適用可能性については考察されておらず，大小比較を行う質問式については，最適化の提案がされていない．

本研究では既存の XQuery 処理系における結合処理の高速化を目的とする．我々は結合処理に対する高速化手法を系統的に考察し，処理手順を確立する．具体的には結合処理を比較方法の観点から同値比較と大小比較に分類し，入れ子構造の観点から 2 重の for と 3 重以上の for に分類する．同値比較にはハッシュ法，大小比較には二分探索を用いて高速化する方法を提案する．3 重以上の for 節に関しては，2 重の for 節のアルゴリズムを組み合わせるで処理可能であることを示した．本稿ではハッシュ法および二分探索が適用できる条件を整理し，それぞれのプログラムを作成して実験を行った．その結果，従来平均 $O(n^2)$ の時間計算量であった結合処理が同値比較では平均 $O(n)$ の時間計算量で，大小比較では平均 $O(n \log n)$ の時間計算量で処理が出来たことを確認した．

なお，江崎は主に関連研究，関谷は主に設計と実装，高橋は主に実験を担当した．

2　XQuery 問い合わせにおける問題

2.1　概要

XQuery とは, W3C において定められた XML 文書に対する問い合わせ言語である．XQuery では主に XPath 式，FLWR 式の 2 つの形式で問い合わせを記述する．

2.2　結合処理

結合処理とは XML 文書の異なる箇所の要素群を照合し，条件にあった要素を抽出する処理である．XQuery では入れ子構造の for 節のそれぞれで要素を取得し，where 節で両者を照合する．

一般に結合処理で照合の対象となるデータはファイルサイズに比例する．ファイルサイズを n とすると，外側の for 節で取得される要素数と，内側の for 節で取得される要素数は n に比例する．結合処理ではそれらの要素の組合せを照合するのでその回数は n^2 に比例する．したがって時間計算量は平均 $O(n^2)$ となる．

3　既存の XQuery 処理系

3.1　XQuery 問い合わせ処理生成系

従来は，XML 文書全体の木を構築していたので，XML 文書が大規模になると，木の構築に多量の処理時間とメモリ領域が必要となっていた．文献 [2] ではその問題を解決するために高速かつ軽量な XQuery 処理系として xqgen を提案した．xqgen は XQuery 問い合わせ処理を構文解析問題ととらえ，XML 文書を構文解析し，必要要素木を作成し，問い合わせ処理を行う．XML 文書全体の構文木を作成するのではなく，処理に必要な要素のみの木を作成することで問い合わせ処理の処理時間の削減やメモリの削減を行うことができる．

xqgen により 1 重の for 節などの質問式で処理時間の削減が行われた．しかし，xqgen は構文解析後の木の処理については考察されていない．したがって結合処理のような質問式では，木の走査時間がかかるので処理時間の削減を行うことができなかった．

3.2　遅延パーサを用いた処理系

文献 [3] では遅延パーサを用いた XQuery 処理系を提案し，結合処理をハッシュ法を用いて高速化する方法が示されている．文献 [3] では 2 重の for 節において外側と内側の for 節で取得した要素の同値比較をする XMark の Q8 について，内側の for 節で使用される要素をハッシュテーブルに格納し，探索の際に外側の for 節を比較することにより高速化できることが確認されている．しかし，文献 [3] ではハッシュ法が適用できる質問式の条件が明確ではなく，大小比較により照合を行う結合処理の高速化手法は提案されていない．また，3 重以上の for 節の入れ子の質問式に対する適用可能性も考察されていない．

4 ハッシュ法および二分探索を用いた結合処理の最適化手法

4.1 概要

本研究では，結合処理の高速化手法を提案する．まず結合処理を比較方法の観点から同値比較と大小比較に分類し，入れ子構造の観点から 2 重の for 節と 3 重以上の for 節に分類する．そして，同値比較にはハッシュ法を，大小比較には二分探索を用いた高速化手法を提案する．

4.2 結合処理の分類

4.2.1 同値比較と大小比較

XMark[1] やその他のベンチマークを調べた結果，結合処理を次の 2 つに分類することができた．そしてそれぞれについての高速化手法を考察する．

1．2 つの異なる箇所の要素群を同値比較により照合する結合処理

2．2 つの異なる箇所の要素群を大小比較により照合する結合処理

XMark の Q8 は 1 に，Q11，Q12 は 2 に該当する．3 重の for 節を用いた Q9 は，1 の処理を組み合わせたものである．

また，XMark 以外のどのベンチマークでも結合処理は存在していた．よって結合処理は一般的に存在するので本研究の一般性も高いと考えられる．

4.3 質問式の定式化

次の条件を満たした質問式について本手法が適用可能である．

1.for 節の入れ子を用いた問い合わせである

1 つの for 節で指定できる要素群は 1 つである．結合処理は，2.2 節で説明したように異なる箇所の要素群を照合するので，2 つの for 節の入れ子を必要とする．

2.where 節の左辺と右辺がそれぞれ，内側と外側のどちらかの for 文の Path 式の末端要素またはその子孫と一致する

where 節の左辺の値 (要素または属性値) が 2 つの for 節のどちらかのパス内またはその子孫にあり，それと同時に右辺の値 (要素または属性値) が左辺ではない方の for 節のパス内またはその子孫にある場合のみ適用できる．

3.where 節が等号式または不等号式である

条件によって処理方法の判別を行い，最適な処理方法を適用する．

4.3.1 3 重以上の for 節

3 重以上の for 節の結合処理への適用可能性を考察する．4.3 節の条件は入れ子の親子関係にある 2 つの for 節の関係についての条件である．したがって，3 重以上の for 節において，親子関係の for 節のそれぞれが 4.3 節の条件を満たせば適用可能である．

3 重の for 節の処理では，まず 1 番外側と 2 重目の for 節の結合処理の時間計算量は，2.2 節と同様に考えると平均 $O(n^2)$ で，このとき 2 重目の for 節の where 節に合致する要素数はファイルサイズ n に比例する．2.2 節より照合の対称がファイルサイズ n に比例するので合致した要素も for 節と同じ処理を行い，1 番内側の for 節との結合処理の時間計算量も平均 $O(n^2)$ になる．よって 3 重の for 節の処理は 2 重の for 節の組合せと考えられ，時間計算量は平均 $O(n^2)$ となる．

4.4 結合処理の高速化手法の提案

我々は結合処理にハッシュ法と二分探索を用いて高速化する手法を提案する．同値比較にはハッシュ法を，大小比較には二分探索を用いる．

4.4.1 同値比較の結合処理の高速化

同値比較の高速化手法として文献 [3] で用いられたハッシュ法と二分探索の処理の 2 つの方法が考えられる．ハッシュ法の方が高速で最適化を行えるので，ハッシュ法を用いる．ハッシュ法は対象のデータをキーとしてハッシュ関数でハッシュ値を割り出し，ハッシュ値に対応した配列番号のハッシュテーブルにデータを格納する．探索の際はまたハッシュ値を割り出し，作成したハッシュテーブルを参照することでそのデータを探索することができるので全ての要素の参照を行わずに済み，処理時間が削減できる．ハッシュ法を用いた結合処理のアルゴリズムを以下に示す．

1. キーは内側の for 節の Path 式の末端要素または子孫の where 節の値とする．
2. キーから計算されたハッシュ値をもとに内側の for 節の Path 式の末端要素をテーブルに格納する．
3. 探索の際にはもう一方の where 節の値をキーとして作成したハッシュテーブルを参照する．

4.4.2 ハッシュ法

ハッシュ法にはチェイン法とオープンアドレス法がある．チェイン法では衝突が起きた場合，リストを用いてデータを格納する．オープンアドレス法では格納要素数に制限があり，配列への格納数が多くなると探索に時間がかかるので本手法には適していない．よって我々はチェイン法を用いる．しかし，チェイン法では最悪の場合は，同じ番地に全ての要素が格納されるので 1 回の探索に $O(n)$ の時間計算量となる．衝突回数は，ハッシュ関数の性能によって変化する．ハッシュ関数の性能評価をすることにより，探索の時間計算量を求める．

今回用いた XML 文書でキーとなる値は文字型であり，文字型を処理できる方法として，ホーナー法と普遍ハッシュ法がある．この 2 つをハッシュ関数に用い，それぞれの平均と分散を求めて，性能評価をすることにより，より良いハッシュ関数を使用するとともに，予測されている時間計算量になることを確認した．

ホーナー法は，$P(x)=a_0+x(a_1+x(a_2+\cdots x(a_{n-1}+a_nx)\cdots))$ の式で表現し，本手法では文字列を左から右へ進みながら，足し込まれた数を 128 倍して，次の符号化された値を足すということを繰り返し行う．

普遍ハッシュ法の処理は，掛ける値の係数にランダム値を用い，キーの各桁に対して異なるランダムな値を使う処理である．

平均は 1 に近いほど関数の性能がよく，分散は 0 に近いほどばらつきが少ない．表 1 がそれぞれの配列に格納さ

れているデータ数の平均と分散を表にしたものである．この表から，平均，分散ともに普遍ハッシュの法が低い値だったので本研究では普遍ハッシュを使用することとした．そして平均の値から探索の時間計算量が O(1) で行えていることが確認できた．

表 1　ホーナー法と普遍ハッシュ法の格納データ数の平均と分散

ハッシュサイズ	1193	597
ホーナー法の平均	1.32	1.63
ホーナー法の分散	1.07	1.03
普遍ハッシュ法の平均	1.12	1.55
普遍ハッシュ法の分散	0.62	0.97

4.4.3　大小比較の結合処理の高速化

大小比較の高速化手法として，二分探索を用いる．これは，大小比較の結合処理においては，ハッシュ法を用いることができないからである．大小比較では調べたい要素の特定ではなく，要素群の特定を行う．ハッシュ法では要素を昇順または降順にソートしてテーブルに格納するわけではないので，大小比較の処理では異なる要素群どうしで逐次探索を行い要素群を特定する．これは 2.2 節の処理と変わらないので，ハッシュ法を用いても高速に処理が行えない．

二分探索では，ソートされた配列を用いて処理を行う．対象のデータがソートデータの中央値と一致するかを調べ，一致するならそのデータを出力する．一致しない場合は対象のデータと中央値の大小を比較し，中央値よりも大きいならば中央値以上のデータのみで同じ探索を行う．不等号の向きや等号の有無によって探索する範囲は変わる．探索結果をもとに，それよりも配列番号が大きいまたは小さい配列の要素を探索結果として出力する．

二分探索を用いた結合処理のアルゴリズムを以下に示す．

1. where 節の左辺または右辺のうち内側の for 節の末端要素および，その子孫要素を配列に格納する．
2. 手順 1 で作成した配列をマージソートする．

where 節にある変数のうち，手順 1 で保存した値をキーとしてソートする．

3. 外側の for 節と対応する where 節の値で二分探索を行う．探索する値と定数との演算がある場合は，二分探索のマッチングの際に定数との演算を行う．

手順 2 でマージソートを選択したのは，時間計算量が平均 $0(n\log n)$ であるクイックソート，マージソートとヒープソートのうち安定なソートであり，既存の処理系と結果の出力が同じ値を示すという理由からである．

4.5　時間計算量

ハッシュ法の時間計算量は，要素数 $N \times M(N,\ M$ はファイルサイズ n に比例する) の場合，片方の要素をハッシュテーブルに格納することで，探索の際には平均 O(1) の時間計算量で済み，全体として平均 O(n) の時間計算量になる．

二分探索の時間計算量は，要素数 $N \times M$ の場合，要素をソートするのに平均 O($n\log n$) かかり，N 個の要素をそれぞれ二分探索するのに平均 O($n\log n$) の時間計算量がかかる．よって全体で平均 O($n\log n$) の時間計算量である．

5　実験

ハッシュ法および二分探索を用いた結合処理の有効性を示すために実験を行った．ここでは，xqgen を用いて XMark の質問式を処理するプログラムを生成し，生成されたプログラムの for 節の処理部分を質問式に応じてハッシュ法および二分探索を用いるように書き換えた．実験では次のことを確認する．

1. 一般的に用いられている SAXON や既存の処理系である xqgen と本手法を比較しどれだけ処理時間が削減されているか．
2. 予測した時間計算量となっているか．
3. ハッシュ法のオーバーヘッドは無視できるか．

1 を確認するために，同値比較は XMark の Q8 を用いて比較を行い，大小比較は XMark の Q11 を用いて比較を行う．2 では，同値比較に対しては，1 重の for 節の処理時間との比較を行い，O(n) の時間計算量となることを確認する．大小比較に関しては，$n\log n$ のグラフと二分探索を適用した結合処理との比較を行い，O($n\log n$) の時間計算量となることを確認する．3 では，XmarkQ8 に関して本手法と xqgen で小さいファイルサイズで比較を行い，オーバーヘッドが無視できるかを確認する．実験環境は，PC(Vine Linux 3.1, Kernel 2.4.27-0v17.3, CPU Celeron M 370 1.5GHz, メモリ 1.256GB，C コンパイラ Gcc 3.3.2，Java version 1.5.0_11) で行った．

5.1　予測

実験を行う前に結果の予測をした．4.5 節で示したように XMark の Q8 はハッシュ法を用いることにより平均 O(n) の時間計算量に，XMark の Q11 は二分探索を用いることにより平均 O($n\log n$) の時間計算量になると考えられる．

3 重の for 節の質問式の場合，ハッシュ法を用いた場合は，1 番外側と 2 重目の for 節の時間計算量は平均 O(n)，合致した要素と 1 番内側の for 節の時間計算量は平均 O(n) となり，全体で平均 O(n) である．二分探索を用いた場合は，同様に考えると平均 O($n\log n$) である．

本手法の処理時間におけるコストについて考察する．外側の for 節の要素数を n，内側の for 節の要素数を m とする．このとき m，n は共にファイルサイズに比例する．本手法を用いた結合処理では，要素を格納するのにかかるコストを α とし，要素を探索するのに β とおくことで本手法の全体の処理は $\alpha m+\beta n$ となる．

本手法を用いない結合処理ではマッチングにかかるコストを γ とおくことで γmn となる．

本手法を用いる場合は，$\alpha m+\beta n > \gamma mn$ の条件を満たすことにより本手法を用いることができる．しかし，m，n が小さい値のときはこの条件はなりたたない．

5.2 実験結果

表 2 に XMark の Q8 の処理時間を示す．この結果から本手法は既存の処理系よりも処理時間が大幅に削減できていることが確認できた．

表 2　本手法と xqgen，SAXON の処理時間 (秒) の比較 (XMark の Q8)

サイズ (MB)	5	10	15	20	25
本手法	0.23	0.48	0.68	0.92	1.19
xqgen	2.02	8.05	17.65	31.11	48.39
SAXON	4.95	14.38	28.84	48.9	74.61

表 3 に XMark の Q1(1 重の for 節) の処理時間とハッシュ法を用いた XMark の Q8(2 重の for 節) と 3 重の for 節の処理時間の結果を示す．1 重の for 節は平均 $O(n)$ の計算量であるが，Q8 と 3 重の for 節も 4.5 節で示した予測通りに平均 $O(n)$ の時間計算量になっていることが確認できた．

表 3　ハッシュ法を適用した 3 重の for 節と 2 重の for 節 (Q8) と 1 重の for 節 (Q1) の処理時間 (秒) の比較

サイズ (MB)	10	20	30	40	50
3 重の for 節	0.45	0.91	1.36	1.81	2.31
Q8	0.48	0.92	1.37	1.85	2.30
Q1	0.41	0.80	1.19	1.59	1.99

また図 1 は Xmark の Q8 と 3 重の for 節のファイルサイズごとの処理時間である．このグラフからも処理時間がファイルサイズつまり要素数に比例しているといえる．

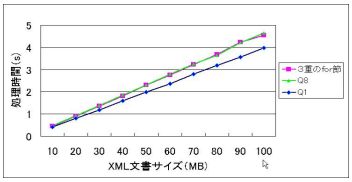

図 1　XMark の Q8 と 3 重の for 節のファイルサイズごとの処理時間 (秒)

表 4 に XMark の Q11 を本手法と既存の XQuery 処理系である xqgen と SAXON で比較した．ハッシュ法同様，本手法は既存の処理系よりも処理時間が大幅に削減できていることが確認できた．

表 4　本手法と SAXON，xqgen の処理時間 (秒) の比較 (XMark の Q11)

サイズ (MB)	10	20	30	40	50
本手法	0.51	1.12	1.82	2.60	3.40
xqgen	3.08	12.54	27.98	48.50	74.17
SAXON	14.6	50.8	112.3	191.9	296.1

また，図 2 は Xmark の Q11 のファイルサイズごとの処理時間と $n \log n$ のグラフである．このグラフからも予測どおり処理時間が平均 $O(n \log n)$ といえる．
表 5 より，ファイルサイズの小さい XML 文書においても，本手法と xqgen は同等または高速に処理できたことが確認でき，本手法のオーバーヘッドは小さく無視できるものと考えられる．

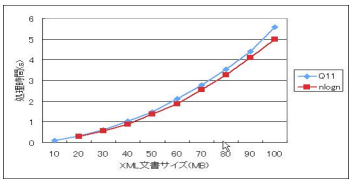

図 2　XMark の Q11 ファイルサイズごとの処理時間 (秒) と $n \log n$ のグラフ

表 5　本手法と xqgen との処理時間 (秒) の比較 (XMark の Q8)

サイズ (MB)	0.1	0.5	1	2	3
本手法	0.00	0.01	0.01	0.01	0.02
xqgen	0.00	0.01	0.03	0.12	0.35

6 考察

6.1 関連研究との比較

今回の実験により，3 章で示した既存の XQuery 処理系と比べ，ハッシュ法および二分探索を用いた結合処理は処理時間を削減できたことから本研究の有効性が確認できた．またどちらも予測した時間計算量どおりに処理が行えたことも確認できた．

6.2 他の処理系への適用

本研究では，xqgen を用いて結合処理の最適化を行った．本研究は xqgen 以外の XQuery 処理系に対しても適用できると考えられる．他の処理系へ適用するために 4.3 節で示した条件の判別方法，内側の for 節で取得する要素の格納方法の 2 点を考察する必要がある．

7 おわりに

本研究では結合処理の最適化として，既存の処理系である xqgen にハッシュ法と二分探索を組み込む方法を提案し，それぞれが使用できる問い合わせ式の定式化を行った．そして，同値比較の結合処理は平均 $O(n)$ で，大小比較の結合処理は平均 $O(n \log n)$ で処理できることを確認した．
今後の課題として，他の処理系への適用のための本手法の整理が必要である．

参考文献

[1] A. R. Schmidt, F. Waas, M. L. Kersten, D. Florescu, I. Manolescu, M. J. Carey, and R. Busse, “XMark：A Benchmark for XML Data Management,” *Proc. of the 28th VLDB Conference*, 2002, pp. 974-985.

[2] 水野耕太, “XQuery 問い合わせプログラムの生成に関する研究,” 南山大学大学院数理情報研究科 2006 年度修士論文, 2007.

[3] 古川健太, 長瀬安弘, 坂口博紀, “XQuery 問い合わせ処理の最適化に関する研究,” 南山大学数理情報学部情報通信学科 2007 年度卒業論文, 2008.