XQuery 問い合わせ処理の最適化に関する研究

2004MT016 古川 健太　　2004MT066 長瀬 安弘　　2004MT084 坂口 博紀
指導教員　蜂巣 吉成

1　はじめに

XML 文書に対する問い合わせ言語として XQuery に注目が集まっている．XML 文書の大規模化に伴い，問い合わせ処理において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．XML 文書全体の構文木を作成する SAXON[2] のような処理系においては，XML 文書が大規模になるにつれて構文木が大きくなり，構文木の構築に多大な処理時間とメモリ使用量がかかってしまう．これに対しストリーム指向処理 [3] では，構文木を構築しない処理なので，構文木の構築に時間がかからず高速である．しかし，XML データの中の全く異なる箇所からのデータの突き合わせをおこなう結合処理などの扱えない質問式がある．

本研究では，遅延パーサ [1] を用いて，結合処理のようなストリーム指向では扱えない質問式も扱え，高速かつ軽量な XQuery 処理系を提案することを目的とする．

本手法による処理系と SAXON を処理時間，メモリ使用量について実験し比較した．一重の FOR 節の問い合わせ処理においては，約 1.5 倍高速で，約 2/3 のメモリ使用量で処理できることを確認した．結合処理では，SAXON より処理時間を必要としていたが，その原因を明らかにし，高速化する方法を考察した．

古川は主に関連研究, 実験，長瀬は主に実験，考察，坂口は主に設計，実現を担当した．

2　関連研究

2.1　XQuery

XQuery とは W3C によって提案された問い合わせ言語で，XPath 式と FLWR 式の 2 つの形式で表される．

XPath 式はロケーションパスで表される式で，'/' で区切ってノードの階層を表現する．//(子孫軸)，*(ワイルドカード) を用いて，簡潔に複数のパスに該当する式を記述できる．例えば，/A//B はルート要素 A の子孫に 0 回以上要素が現れ，要素 B が出現することを表す．

FLWR 式は, FOR 節, LET 節, WHERE 節, RETURN 節で成り立つ式のことである．FOR 節，LET 節では要素の取得，WHERE 節では条件指定，RETURN 節では，結果の作成をおこなう．

2.2　遅延パーサ

遅延パーサは参照があったノードのみで木を作成するパーサである．アプリケーションの処理に必要なノードのみで木を構築するので処理が高速である．

2.2.1　概要

遅延パーサの処理過程を図 1 に示す．処理は前処理と木作成処理 (progressive parsing) から構成される．遅延パーサは XML 文書を入力とし，前処理で XML 文書の構文木を作成するのに必要な情報 (以下，内部情報) を保存する．アプリケーションが未作成のノードを参照すると，木作成処理がおこなわれ，内部情報をもとに参照があったノードが作成される．

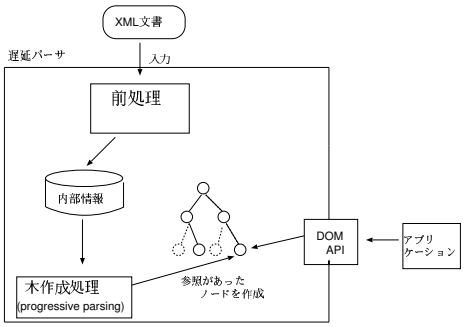

図 1　遅延パーサの処理過程

2.2.2　XQuery に応用した場合の問題点

遅延パーサを XQuery 処理系に用いた場合，//person のような展開されていないパス式が含まれているとき，パスの経路が省略されているので，person までの経路が分からない．したがって，person を探すために全てのノードを参照してしまうので，全体の木を構築してしまう．

パスを展開する方法として，パススキーマを用いる手法がある [3]．パススキーマとは XML 文書に現れる全てのパスの構造を示した木構造データで，特定の XML 文書から構築される．図 2 にパススキーマの例を示す．

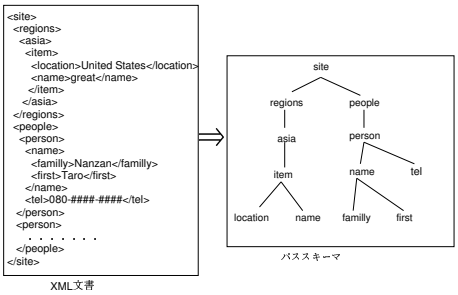

図 2　XML 文書から生成されるパススキーマ

パススキーマは XML データのタグ情報をもとに構築される．パススキーマの構築には，XML 文書を読み込む必要があるので，パススキーマの構築と問い合わせ処理で合わせて 2 回 XML 文書を読み込むことが必要となり処理に時間がかかる．

3 遅延パーサを用いた XQuery 処理系

3.1 概要

我々は XML 文書を 1 回読み込むだけで，パススキーマの構築と問い合わせ処理をおこなう XQuery 処理系を提案する．提案する手法の概略を図 3 に示す．

我々の手法では，XML 文書を読み込む際に XML 文書のパススキーマの構築と内部情報の保存を行う．ここで構築されたパススキーマを利用してパスの展開をおこない，問い合わせ式を書き換える．書き換えられた問い合わせ式は，構文木を参照して処理される．

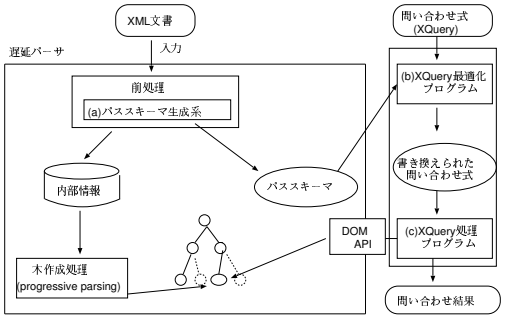

図 3　本手法の問い合わせの流れ

本研究で，遅延パーサに追加した処理を以下に示す．

(a) XML 文書からパススキーマの生成
(b) パススキーマから問い合わせ式の書き換え
(c) 問い合わせの処理

3.2 処理方法

パススキーマの生成

本手法では，遅延パーサの前処理の部分にパススキーマ生成系を追加した．XML 文書を読み込んでいる際に開始タグ，終了タグが出てきた時，パススキーマ生成系でパススキーマの構築を行う．これにより前処理で XML 文書の内部情報が保存されるとともに，パススキーマが生成される．

パススキーマの構築アルゴリズムを以下に示す．

1. ルートノードを作成し，カレントノードとする．
2. XML 文書を読み込んで解析して, 開始タグ，終了タグの列として扱い，XML 文書の解析が終了するまで以下の処理を繰り返す．
   - 2.1 開始タグが出現した場合，以下の処理をおこなう．
     - 2.1.1 カレントノードの子ノードに開始タグの要素名と同じノードがなければ作成する．
     - 2.1.2 カレントノードを開始タグの要素名と同じ子ノードにする．
   - 2.2 終了タグが出現した場合，カレントノードを現在のカレントノードの親ノードにする．

DTD からパスの展開を行う方法もあるが，DTD と比較して本研究で構築するパススキーマは実際の XML 文書から生成されるので，ある特定の XML 文書に対して特化した構造になっている．親が異なる同じ名前の要素は DTD では同じ要素として扱われるが，パススキーマでは異なる要素として扱われる．

例えば図 2 の XML 文書で，//familly は，DTD から展開すると，/site/regions/asia/item/name/familly, /site/people/person/name/familly の 2 通りに展開されるが，本研究で構築したパススキーマから展開すれば，/site/people/person/name/familly のみに展開される．

問い合わせ式の書き換え

パススキーマは XML 文書の構造を表しているので，パススキーマを 1 度走査することにより，パスの展開をするのに必要な情報が得られ，パスを展開することができる．問い合わせ式の書き換えの部分で，パスの展開をおこなう．

問い合わせの処理

展開されたパス式は必要な要素が階層順に並んでいるので，パス式に出てくる順に要素へアクセスをしていけば，必要な要素のみにアクセスすることができる．

一重の FOR 節を用いた処理は書き換えられた問い合わせ式から，XML 文書の構文木を 1 度走査することで問い合わせ処理をおこなうことができる．FOR 節の入れ子を用いた結合処理は，処理をおこなうときに XML 文書の構文木を複数回走査する必要がある．

3.3 設計と実現

Java, JavaCC を用いて処理系を実現した．JavaCC は，問い合わせ式の構文を解析する目的で用いた．解析した問い合わせ式は木構造であらわす．また，XML 文書の木構築には，DOM API を用いた．DOM API には，特定の子要素を取得するメソッドが存在しない．特定の名前の子孫要素や，全ての子要素を取得するメソッドはあるが，これらのメソッドでは必要の無い要素へアクセスしてしまい，必要の無い木を構築してしまう．DOM の Element クラスのサブクラスを作成し，特定の名前の子要素を取得するメソッドを追加した．

4 実験

遅延パーサを用いた本手法の処理時間，メモリ使用量の測定をおこない, 性能を評価した．実験環境は, PC(Vine Linux 3.2，celeron M，1.20GHz，メモリ 1.23GB) 上で，Java(1.5.0) を用いた．

実験では，XQuery 処理系を評価する標準的なベンチマークである XMark[4] を用いた．今回の実験では XMark の 20 個の質問式の内，代表的な問い合わせ式 (Q1，Q5，Q6，Q8，Q13，Q14，Q15，Q17) を扱った．表 1 に実験で扱った問い合わせ式の特徴を示す．

比較対象は，広く用いられている XQuery 処理系 SAXON とした．

表 1　XMark の問い合わせ式

No	特徴
1	値による絞りこみ
5	要素数の取得
6	必要要素が多い処理，要素数の取得
8	結合処理
13	要素の作成
14	文字列の検索
15	子要素の取得
17	子要素の有無

4.1　処理時間

図 4 に本手法と SAXON の XML 文書のサイズごとの処理時間を示す．一重の FOR 節の中で平均の処理時間であった Q1 と結合処理である Q8 の結果を示す．表 2 に 100MB の XML 文書を用いたときの，各問い合わせ式ごとの処理時間を示す．

実験の結果，本手法の Q8 以外の問い合わせ式の処理時間における，前処理部分の割合は約 95 %であった．

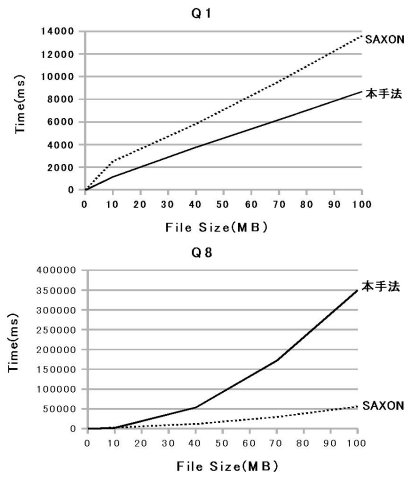

図 4　XML 文書のサイズごとの処理時間

表 2　100MB の場合の各問い合わせ式ごとの処理時間

	Q1	Q5	Q6	Q8	Q13	Q14	Q15	Q17
SAXON(ms)	13,617	13,708	13,552	56,211	13,865	14,053	13,556	13,668
本手法 (ms)	8,700	8,729	8,502	350,612	8,659	8,746	8,647	8,963

4.2　メモリ使用量

図 5 に本手法と SAXON における XML 文書のサイズごとのメモリ使用量を示す．測定した結果，処理時間は問い合わせ式によって違いがあったが，メモリ使用量は，問い合わせ式によらずほぼ一定であった．実験の結果，本手法のメモリ使用量における，前処理部分の割合は約 99 %であった．

4.3　評価

測定した処理時間，メモリ使用量をそれぞれ SAXON と比較する．

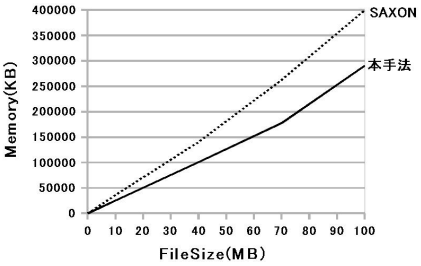

図 5　XML 文書のサイズごとのメモリ使用量

処理時間

SAXON と比較して，Q8 以外の問い合わせ式を高速に処理できることを確認した．高速に処理できた要因は，SAXON では，XML 文書全体の木を構築するのに対し，本手法では，問い合わせに必要なノードのみの木を構築するので，木構築ノード数を大きく削減できたことである．表 3 より SAXON と比べ本手法は，木構築ノード数が大きく減っていることが分かる．また，木構築ノード数を大きく削減できたことから，Q8 を除く本手法で測定した処理時間の内，前処理部分が占める割合が約 95 %となった．

図 4 より，処理時間は一重の FOR 節を用いた問い合わせ式では，XML 文書サイズに比例しているが，結合処理を用いた問い合わせ式では，XML 文書サイズの二乗に比例しており，大きく処理時間がかかっている．結合処理を高速に処理する方法については，5.2 節で考察する．

表 3　100MB の XML 文書での木構築ノード数

	Q1	Q5	Q6	Q8	Q13	Q14	Q15	Q17
SAXON	4186687	4186687	4186687	4186687	4186687	4186687	4186687	4186687
本手法	22699	26033	19365	68089	56301	19365	49272	56777

メモリ使用量

SAXON と比較して，全ての問い合わせ式において 2/3 程度のメモリ使用量で処理できることを確認した．その主な要因は，木構築ノード数を削減できたことである．木構築ノード数を大きく削減できたことから，本手法で測定したメモリ使用量の内，前処理部分が占める割合が約 99 %となった．

5　考察

実験結果をもとに以下のことを考察する．

- 一重の FOR 節，結合処理の計算量
- 結合処理の高速化

5.1　一重の FOR 節，結合処理の計算量に関する考察

一重の FOR 節を用いた問い合わせ式と，結合処理を用いた問い合わせ式について，それぞれ計算量を考察する．

5.1.1 一重の FOR 節

一重の FOR 節を用いた問い合わせ式では、FOR 節で処理するノードを取得するのに，パス指定に従って木を 1 度走査すればよく，最悪の場合でも木のノードを全て走査すればよいので，XML 文書のノード数を n とすると，最大で n 回ノードをアクセスすればよく，計算量は $O(n)$ である．

また FOR 節で取得するノード数は，XML 文書サイズに比例するので，FOR 節での処理の計算量は $O(n)$ となる．

本手法，SAXON ともに計算量は $O(n)$ であるが，本手法では SAXON と比べ，木構築時間，ノードを探索する範囲を削減することで, 処理の高速化をおこなっている．SAXON では FOR 節で a//e のような展開前のパス式があると，広範囲にわたってノードを探さなければならず, 結果的にパスに該当しないノードも探すことになる．本手法では，前処理部分でパスが展開されるので，必要なノードのみアクセスできる．

5.1.2 二重の FOR 節を用いた結合処理

外側の FOR 節の処理は，一重の FOR 節と同じように処理できるので，計算量は $O(n)$ である．また，内側の FOR 節の処理だけで考えた場合，一重の FOR 節と同様に，計算量は $O(n)$ である．内側の FOR 節は，外側の FOR 節で取得された各ノードについて繰り返す．したがって，外側の FOR 節により内側の FOR 節が最大で n 回繰り返されるので，計算量は $O(n^2)$ となる．

結合処理は，木の構築よりも木の走査に時間がかかるので，本手法では，処理時間の削減がおこなえなかった．

5.2 結合処理の高速化

二重の FOR 節を用いた結合処理では，5.1.2 節で述べたように，一重の FOR 節と比べ計算量が大きくなり，問い合わせ処理に時間がかかる．Q8 は外側の FOR 節で取得したデータに対し，内側の FOR 節で取得したデータを比較している．また, 外側の FOR 節に関わらず, 内側の FOR 節で毎回同じノードについて処理している．比較しているデータをキーとし，ハッシュ法を用いることで，Q8 の処理を高速化できる．

```
for  $p  in /site/people/person
let   $a :=
   for  $t  in /site/closed_auctions/closed_auction
   where  $t/buyer/@person = $p/@id
   return  $t
return  <item  person = "{$p/name/text()}"> {count($a)} </item>
```

Q8の質問式

この手法では，外側の FOR 節では，一重の FOR 節と同じように処理するので，計算量は $O(n)$ である．内側の FOR 節では，ハッシュ法を用いることで，平均して $O(1)$ の走査で処理できる．よって，計算量は $O(n)$ となる．この手法を用いることで二重の FOR 節を用いた結合処理の計算量が $O(n^2)$ から $O(n)$ となり，大きく処理時間を削減できる．

二重の FOR 節を用いた結合処理である Q8 の問い合わせ式で，ハッシュ法を用いた本手法と SAXON の処理時間の比較を図 6 に示す．ハッシュ法を用いた本手法が SAXON と比較して高速に処理できることを確認した．

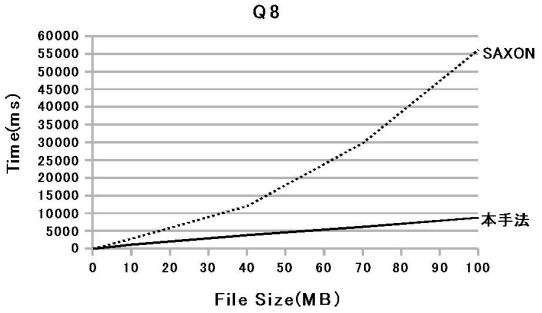

図 6　Q8 での SAXON との比較

6 おわりに

本研究では，遅延パーサを用いた XQuery 処理系を設計，実現し，XMark を用いて評価した．木の構築割合を削減することにより，既存の処理系 SAXON と比較し，処理時間，メモリ使用量ともに削減できた．

今後の課題を以下に示す．

1. 前処理部分にかかる処理時間，メモリ使用量の削減
2. 結合処理の高速化

今回の実験より，問い合わせ処理にかかる処理時間，メモリ使用量ともに前処理の占める割合が高いことから，前処理部分の処理時間の削減が求められる．

5.2 節で述べた Q8 のような結合処理において，ハッシュ法を用いることにより高速化をおこなった．今回，Q8 の高速化については，Q8 の処理に特化したプログラムを作成し処理系に追加している．ハッシュ法が使用できる問い合わせ式の定式化などをおこなう必要がある．

参考文献

[1] M. L. Noga, S. Schott and W. Lowe , “Lazy XML Processing, ” Proc. of the 2002 ACM symposium on Document engineering, pp.88-94, 2002

[2] Saxonica, “XSLT and XQuery Processing, ” http://saxonica.com/.

[3] 石野 明, 竹田 正幸, “パスプルーニングによる決定性有限オートマトンを用いた XQuery 処理の提案, ” DBSJ Letters, Vol.4, No.4, pp.17-20, 2006.

[4] A. Schmidt , F. Waas and M. Kersten M. Carey and I. Manolescu and R. Busse, “A benchmark for xml data management, ” Proc. of the 28th VLDB Conference, pp.974-985, 2002.