ストリーム指向による XQuery 問い合わせ処理の効率化

2003MT055 宮田 裕則　　2003MT106 牛田 匠
指導教員　張 漢明

1 はじめに

XML は柔軟な構造による記述の容易さ, 問い合わせ言語 XPath[1], XQuery[2] などの関連技術の豊富さから広く普及している. XQuery は XML データを木構造とみなし, パスを用いて特定のノード集合を抽出し, 処理する問い合わせ言語である. XML データの大規模化に伴い, XQuery 処理系の高速化が望まれている. 高速処理を行う XQuery 処理系として, ストリーム指向 XQuery 処理系が研究されている. XML データから木を構築することなく, 走査しながら問い合わせ処理ができる.

しかし, 扱える XQuery に制限がある. ストリーム指向 XQuery 処理手法 [3] では, ストリーム走査で過去に検出された情報が保存されないので, パスの途中に条件が含まれている質問式を扱えない.

本研究では, パスの途中に条件が含まれた質問式を扱えるストリーム指向 XQuery 処理手法を提案する. 本手法では, 文献 [3] のプッシュダウンオートマトン (以下, PDA) を用いた処理に加え, リストを用いて過去の情報を保留する.

実験では, 本手法を用いた問い合わせ処理プログラム (a program processing a query in XQuery using PDA and List, 以下, XQPL) と広く用いられている XQuery 処理系 SAXON[4] とを比較した. XQPL が SAXON と比べて, 約 1/560 のメモリ使用量にもかかわらず, 半分の時間で処理できることを確認した.

研究手順を以下に示す.

1. ストリーム指向 XQuery 処理手法 [3] の拡張
2. XQPL の設計と実現
3. 実行時間, メモリ使用量に関する実験, 考察

宮田は拡張と実現, 牛田は実験を担当した.

2 関連研究

2.1 XQuery

XQuery とは, W3C によって提案された XML データに対する問い合わせ言語である. XQuery の質問式は, 主に以下の 2 点からなる.

- XPath によるノード指定
- FLWOR 形式により, 選択されたノードに対して操作を指定

XPath は, XML データの木構造を利用し, ロケーションパス (以下, パス) でノードを選択する. パスは, '/' で区切ってノード階層を表現する. ノードを絞り込む条件は, '[ ]' 内で指定する. 本稿では, パスの最後で指定されるノードを最終ノードと呼ぶことにする.

FLWOR 形式には, For, Let, Where, Order, Return 節がある. For, Let では, XPath によって指定されたノードを変数に代入する. Where では条件指定, Order では並べ替え, Return では結果の作成を行う.

本研究で扱う XQuery

本研究では, For, Where, Return を扱う. XML 文書を一度走査するだけで処理可能な質問式を扱っているので, For の入れ子は扱っていない. For の入れ子は, XML データ中の全く異なる箇所からのデータの突き合わせを行う結合処理となる場合があるので, 一度の走査では処理できない. Let は, For で表現可能であるので扱わない. Order も扱っていないが, 保留したリスト内の情報をソートすることによって対応可能である.

以上の制限は, 文献 [3] と同じであるが, 本手法では, [3] の手法で扱えなかったパスの途中に条件を含んだ質問式が扱える. 図 1 に, パスの途中に条件を含んだ質問式の例を示す. (a) は, 条件のパスに親軸'..' を含んだ (b) と本質的に同じである. (a) は, where 節を用いることで (c) にも書き換えられる. 本研究では, 条件は全て (c) のように where 節で扱う. 条件を where 節だけに記述することで, XQPL 生成の際に必要となる質問式の構文解析が容易となる.

```
(a)  for $a in /people/person[address/pref = "Aichi"]/name
     return $a

(b)  for $a in /people/person/name[.. /address/pref = "Aichi"]
     return $a

(c)  for $a in /people/person
     where $a/address/pref = "Aichi"
     return $a/name
```

図 1　パスの途中に条件を含んだ質問式の例

条件のパスに親軸を含んだ質問式は, 過去に検出された情報が必要となる場合がある. where の最終ノードより先に, return の最終ノードが検出された場合である.

質問式で使用する関数は, 検索の条件としてよく用いられる統計関数 (count, sum), 位置指定関数 (position), 文字列関数 (contains) を扱った. 全ての XQuery を扱うことはできないが, データの各部を抜き出すことや統計処理を扱うことができ, 十分に実用的な質問式を扱える.

2.2 ストリーム指向 XQuery 処理系

文献 [3] では, PDA を用いたストリーム指向 XQuery 処理系を実現している. パスの最後に条件があり, 条件のパスは, 子軸, 子孫軸の質問式だけを扱っている.

概要

[3] の手法は, 前処理と本処理に分れている. 前処理は, パススキーマを用いて, 子孫軸'//', ワイルドカード文字'∗' を含んだパスを出現可能な要素名だけのパスに展

開する. パススキーマとは, XML 文書に含まれる全てのパスの種類を表したものである.

本処理では, PDA を用いて問い合わせ処理をする. PDA は, XML データのストリーム走査で検出される開始タグ, 値, 終了タグをイベントとして処理する. それぞれが検出されたときの処理を以下に示す.

- 開始タグ, 値
  - スタック: 開始タグ名と値をプッシュ
  - DFA: 状態遷移
- 終了タグ
  - スタック: ポップ
  - DFA: 状態遷移
  - 条件判定
  - 出力処理

条件判定は, 条件における最終ノードの終了タグ検出時に行う. スタックの一番上のデータが条件に該当したら, return の最終ノードが格納されているスタックデータの該当フラグ (条件判定の真偽値) に true を設定する.

出力処理は, return における最終ノードの終了タグ検出時に行う. スタックデータの該当フラグに true が設定されていれば出力する.

図 2 の XML 文書に対して, 次の質問式
for $a in /people/person/address[pref = "Aichi"]
return $a
を用いて本処理を述べる. 図 3 にスタックの変化, 図 4 に DFA を示す. スタックデータで, 値がなく, 該当フラグが false のものは, タグ名だけで示した.

ストリーム走査器によって, <people> が検出され, スタックにプッシュ (図 3(A)), DFA の状態が S_0 から S_people に遷移する. 同様の処理を <person>, <name>, <first> 検出時に行う (図 3(B)〜(D)). </first> 検出時には, スタックデータ (first, Jiro, F) をポップし (図 3(E)), 状態 S_name に遷移する.

以上の処理を, 条件における最終ノードの終了タグ </pref> が検出されるまで繰り返す. </pref> 検出時に, スタックの一番上のデータが (pref, Aichi) であれば, 条件に該当したので, スタックデータ address の該当フラグに T(true) を設定し, ポップする (図 3(H)). return における最終ノードの終了タグ </address> 検出時に, スタックデータ address の該当フラグに T(true) が設定されていれば, 出力する (図 3(I)).

問題点

[3] の手法では, パスの最後に条件があり, かつ, 条件のパスは子軸, 子孫軸の質問式だけを扱っている. 図 1 のような, パスの途中に条件を含んだ質問式は扱えない. 図 4 を用いて, 図 1 の質問式が扱えない理由を以下に示す.

- 出力処理がされる </name> の時点では, </pref> がまだ検出されていないので, 条件判定がされない.
- 条件判定がされる </pref> の時点では, name の情報がスタックに残っていない.

図 1 の質問式の出力を得るためには, 出力処理のタイミングと過去の情報 name の扱いが重要となる. 本研究では, このようにパスの途中に条件が含まれた質問式を扱える手法を提案する.

```
<people>
 <person>
  <name>
   <first>Jiro</first><family>Ota</family>
  </name>
  <tel>0584212194</tel>
  <address>
   <pref>Aichi</pref><city>Nagoya 1-1-1</city>
  </address>
  <dateOfBirth>
   <year>1954</year><month>10</month><day>12</day>
  </dateOfBirth>
 </person>
 <person>
  .....
</people>
```

図 2　XML 文書

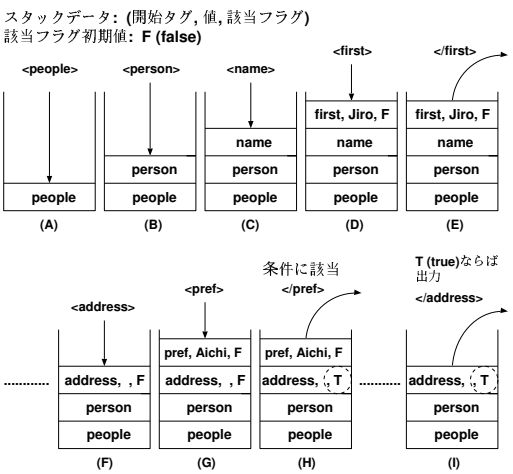

図 3　スタックの変化

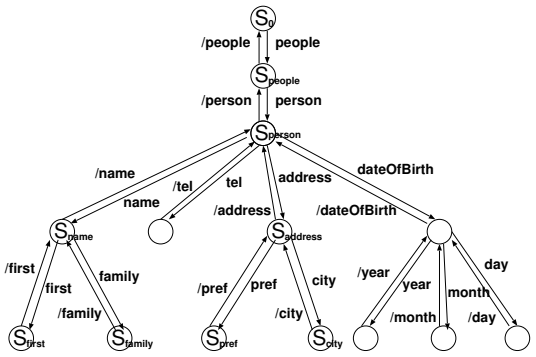

図 4　DFA

3　パスの途中に条件が含まれている質問式を考慮したストリーム指向 XQuery 処理

文献 [3] の手法を拡張し, パスの途中に条件を含んだ質問式も扱える手法とそれを用いた問い合わせ処理プログラムの設計と実現について述べる.

3.1　提案手法

前処理は, パススキーマではなく, XML 文書構造を定義した DTD を用いておこなう. DTD を用いれば, あらか

じめ XML 文書を走査し, パススキーマを構築する必要はない. 本研究では, 問い合わせ処理の手法を提案しているので, 前処理は行われているものとする.

本処理の手順は [3] の手法と同様であるが, 以下の 2 点が異なる.

- 出力処理の遅延
  出力処理を for における最終ノードの終了タグが検出されるまで遅らせる.
- リストによる過去情報の保留
  出力に必要な要素の開始タグが検出されてから, 終了タグが検出されるまでの情報を保存する. その情報は, 出力処理の時点まで保留する.

図 1 の質問式を用いて本手法を述べる.

出力処理の遅延

出力処理を for の最終ノード </person> が検出されるまで遅らせる. where のノードは person の子や子孫であるので, その時点まで遅らせれば, 条件判定ができる.

リストによる過去情報の保留

出力に必要なのは, name に関する情報である. name の開始タグ検出時から, 終了タグ検出時まで, スタックの処理と並行してリストに情報を保存する. その情報は, 出力処理をする </person> 検出時まで保留する.

本手法を用いたスタック, リストの変化を図 5 に示す.

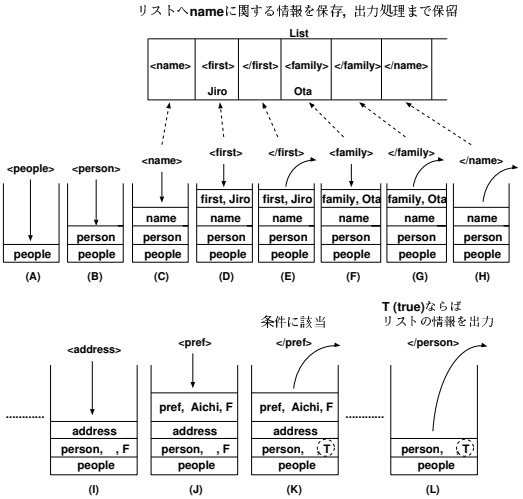

図 5 スタック, リストの変化

where における最終ノードの </pref> が検出されるまで, PDA の処理を繰り返す (図 5(A)〜). name に関する情報が検出されたら, リストに保存する (図 5(C)〜(H)). </pref> 検出時に, スタックの一番上のデータが (pref, Aichi) であれば, スタックデータ person の該当フラグに T(true) を設定し, ポップする (図 5(K)). </person> 検出時に, スタックデータ person の該当フラグに T(true) が設定されていれば, リストに保留していた情報を出力する (図 5(L)). F(false) であれば, リストを初期化する.

本手法を用いれば, 条件がパスの最後だけに制限されず, 途中に含まれている質問式も扱える.

3.2 設計と実現

本手法を用いたストリーム指向問い合わせ処理プログラム (XQPL) の生成概略図を図 6 に示す.

XQPL 生成系は, 以下の手順で生成をおこなう.

1. DTD から PDA を生成.
2. XQuery を構文解析し, PDA の状態遷移時のアクションを生成.

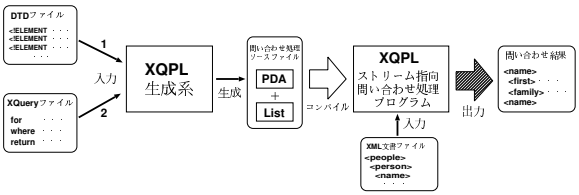

図 6 XQPL 生成の概略図

生成される XQPL は質問式にもよるが, 約 600 行〜1200 行の Java コードで生成される. 質問式からコードを生成するので, 同じ質問式であれば, 再利用できる.

XQPL では, スタックデータに属性名, 属性値を加えたことによって, 属性に関する質問式も扱える. where に複数の条件が指定された場合は, 各条件に対応した該当フラグを用いて処理できる. return 節では, 直接タグ記述ができ, 複数のパス指定も可能であるので, リスト内はパスごとに格納している.

4 実験

XQPL の実行時間, メモリ使用量の実験をおこなった. 実験環境は, PC(Vine Linux 4.0, Pentium 4, 3.2GHz, メモリ 1GB) 上で, 言語は Java(J2SE 1.5.0) を用いた.

実験では, XQuery 処理系を評価する標準的な XMark ベンチマーク [5] を用いた. 本手法では, 結合処理と並べ替え処理について扱っていないので, それらの質問式の評価は行っていない.

比較対象は, 広く用いられている XQuery 処理系 SAXON である. SAXON は, SAX パーサから報告されるイベントをもとに主記憶上に木を構築する. SAXON 同様, XQPL もストリーム走査に SAX パーサを用いた.

4.1 実行時間

扱った質問式 (Q1, 2, 5, 6, 7, 13, 14, 15, 16, 17) のうち, 最も高速な実行時間であった Q15, 平均的な実行時間であった Q13, 最も実行時間がかかった Q14 で SAXON との比較を行った (図 7). Q15 は, パスの深さが 12 の質問式である. Q13 は, return 節で複数のパス指定がされた質問式である. Q14 は, 文字列の照合を行う contains 関数を含んだ質問式である.

XQPL のコード生成に要する時間は, 約 0.3 秒であった. XQPL の問い合わせ処理時間を測るために SAX のストリーム走査時間も載せた. 前処理の時間は, 実行時間に対して非常に小さいので, 含まれていない.

XQPL は SAXON の約半分の時間で処理できることを確認した. XQPL において, SAX のストリーム走査は

実行時間の約 8 割を占めているので, さらに高速なストリーム走査器を用いれば, 一層高速化を図れる.

4.2 メモリ使用量

XQPL, SAXON の各質問式におけるメモリ使用量を測定した (図 8). XQPL は Q13 以外の質問式で, 約 870KB のメモリ使用量であった. PDA だけの処理に使用するメモリ使用量は, 866KB であるので, ほとんど変わらないことがわかる. Q13 については, リストに保存する情報量が他の質問式と比べて非常に多いので, メモリ使用量も多い. SAXON は, どの質問式に対しても木を構築するので, 一定のメモリ使用量 (約 485MB) であった. XQPL は保存する情報量が少なく, SAXON の約 1/560 のメモリ使用量で処理できることを確認した.

以上の実験から, XQPL は SAXON と比較し, 半分の実行時間で, 約 1/560 のメモリ使用量であった.

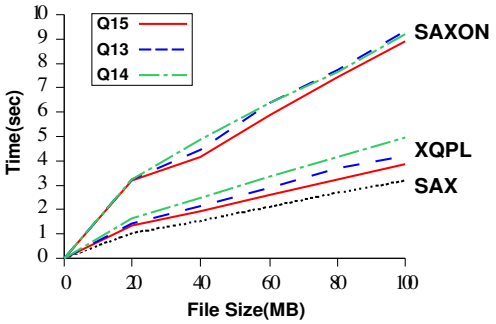

図 7 SAXON との比較

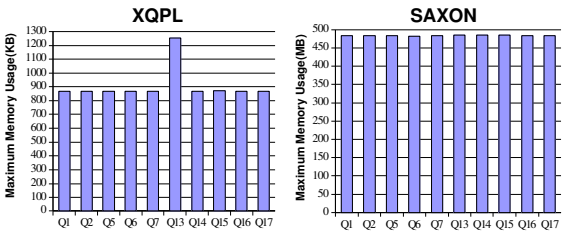

図 8 各質問式における最大メモリ使用量 (100MB 時)

5 考察

5.1 リスト処理

リストに保留する情報は部分木に相当することから, 保留する要素の格納回数と実行時間の関係を考察した. 実験は, Q15 のパスの深さを変えておこなった (図 9). 約 10 万回のリスト処理に 1 秒かかることを確認した.

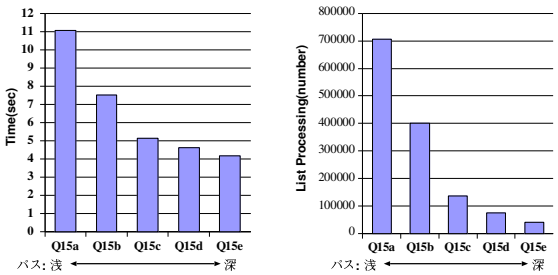

図 9 実行時間とリスト格納回数 (100MB 時)

以上のことから, 4.1 節の Q15 と Q13 の実行時間の差は, リストの格納回数にあることがわかった. Q14 は, contains 関数処理に時間がかかることを確認している.

リストに保留する要素数が実行時間, 文字数がメモリ使用量に依存することがわかった. 本手法を適用する必要がない質問式, つまり, パスの途中に条件が含まれていない質問式については, XQPL 生成の段階で PDA だけの処理にすれば, さらなる高速化が図れる.

5.2 適用可能性

XQPL のアプリケーションへの適用可能性について考察する. XQPL を XML 文書の検索エンジンとしてアプリケーションを構築した場合, 検索に多くのメモリを使用せず, 高速に処理できるので, 本来のアプリケーション処理に時間とメモリを多く割り当てることができる. XQPL は, 実行環境に制限がある組込み機器などに有効である.

6 おわりに

本研究では, パスの途中に条件が含まれている質問式を考慮したストリーム指向 XQuery 処理手法を提案した. 手法を用いた XQPL は, 出力処理を遅らせ, リストによって過去の情報を保留することで実現できた. 実験では, XQPL が SAXON より高速で, メモリ使用量が少ないことを確認した.

以下の 3 点を今後の課題とする.

- 複数 XQuery 処理
- 高速なストリーム走査器の実現
- XQPL 生成の最適化

複数の XQuery 処理も, PDA の各状態にそれぞれの質問式の処理を設定することで処理可能である. 高速性を一層追求するためにも, SAX より高速なストリーム走査器を実現する必要がある. リスト処理の考察から, 本手法を適用する必要がない質問式に関しては, XQPL 生成の段階で PDA だけの処理にする.

謝辞

本研究を進めるにあたり, 二年間御指導いただいた野呂昌満教授, 張漢明助教授, 蜂巣吉成講師, 有益なアドバイスをしていただいた水野耕太さんをはじめ, 大学院生の皆さんに深く感謝いたします.

参考文献

[1] W3C, XML Path Language(XPath) http://www.w3.org/TR/xpath/, Jul. 1999.
[2] W3C, XML Query Language(XQuery) http://www.w3.org/TR/Xquery/, Feb. 2001.
[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, Mar. 2006.
[4] SAXONICA.com XSLT and XQuery Processing http://www.saxonica.com/, Feb. 2005.
[5] An XML Benchmark Project(XMark) http://www.xml-benchmark.org/, Aug. 2002.