

ストリーム指向 XML データに対する効率的フィルタリング

高橋 翼 藤田 悟

法政大学情報科学部デジタルメディア学科

1. はじめに

近年、高速ネットワークの発達により大規模なストリームデータに対する効率的検索の要求が高まっている。ストリームデータにはオンラインニュースなどの更新の頻繁なウェブページ群や金融における取引記録、ネットワーク監視システムの通信記録などがあり、XML が積極的に活用されている。しかし、従来の XML 検索手法は、蓄積された XML データベースに対してクエリを最適化するものであって、ストリームデータからの検索には必ずしも適していない。そこで、2000 年頃からストリーム指向 XML に特化したフィルタリング技法の研究が行われ、XFilter[1]や YFilter[2]などの研究が進められた。XFilter は与えられた XPath クエリごとに有限状態マシンを生成して、入力 XML の受理を持って XML 検索を行う。これに対して YFilter は複数のクエリから、共通接頭辞パスをまとめた非決定性有限オートマトンの探索木を生成し、探索を効率化した。しかし、両システム共に検索のバックトラックコストが高く、更なる高速化が求められると考える。

そこで、本研究ではバックトラックコストの小さい高速フィルタリング技術について検討した。特に、検索クエリの葉要素に注目し、ボトムアップ的に探索を行う手法を提案する。

2. XPath[3]

XPath は XML の検索言語である。XPath は XML の木構造を利用し、パスや部分木に関する情報を組み合わせることによって、XML 内の必要な部分を指定することが可能である。以下に簡単な例を示す。

/a0/b0: a0 の子要素である b0 を選択。
/b0//d0: b0 の子孫要素である d0 を選択。
/d0/*/f0: d0 の孫要素である f0 を選択。

3. 提案するフィルタリング手法

3.1. 対象とする XML と XPath クエリ

今回提案する手法が対象とする XML と XPath クエリの例を図 1 に示す。本論文では XML の深さを Depth 呼び、第 2 レベルでの幅を Width と呼ぶ。XML は図のようにレベルの深い部分で様々な要素が出現する。対象とする XPath クエリは対象の XML の葉要素を含み、'/'、'*' の出現を許容する。

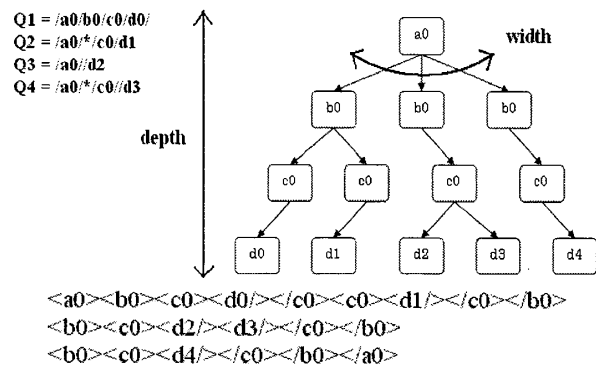


図 1 対象とする XML と XPath

3.2. 基本アルゴリズム

クエリの葉要素を最初に検出し、そこまでのノード探索履歴からクエリとの照合を行うというボトムアップ型の検索手法を採用する。

まず、クエリの葉要素を、出現可能な深さレベルと共に事前インデクシングする。続いて、入力 XML を走査するが、このとき走査した道筋(根から葉までのノード列)を Path と呼ばれるリストに保持しておく。そして、クエリの葉要素と一致する要素を検出したとき、Path とクエリの照合を行う。照合に成功した場合、そのクエリは受理される。根ノードに近いレベルでの状態遷移操作が不要なこと、クエリの照合時には XML の対象ノード列が確定しているために照合コストが小さいことなどから、高速な検索を行うことができる。探索イメージを図 2 に示す。

3.3. キャッシュによる探索の高速化

探索効率化を行うために、2 つのキャッシュ機能を付加した。1 つめは成功時のキャッシュであ

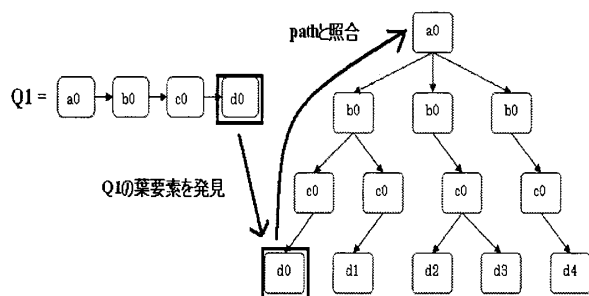


図2 提案する手法の探索イメージ

り、path のそれぞれの深さに成功したクエリを保存することによって、クエリの照合コストを節約する。2 つめは探索失敗時のキャッシュであり、同一のクエリに対しては照合で失敗した深さに戻るまでは葉要素の探索を行わないことにより高速化する。以上のキャッシュ機能は、キャッシュ自体による負荷が増加することからオプション機能として提供した。

4. 性能評価

実験には Microsoft Windows XP Professional Version 2002 Service Pack 2, CPU Intel Pentium M 1.48GB RAM を用いた。Width や Depth をパラメータとして指定できる XML・XPath generator を実装し、様々な大きさの XML と XPath クエリを自動生成し、探索性能を調べた。入力に用いる XML は図 1 で示したような深いレベルで複雑に要素が出現するタイプと、すべてのレベルで一様に要素が出現し、葉要素を同一要素に限定するタイプの 2 つを用意した。前者の XML を使った実験を第 1 実験、後者の XML を使った実験を第 2 実験と呼ぶ。計測した実行時間はすべて XML の探索時間でありインデックス作成などの前処理時間は含まない。Width、Depth、'/'、'*' の生起率の変化による XFilter、YFilter、提案手法の実行時間の違いを計測した。

図 3 に、第 1 実験の Width の変化による実行時間の違いを示す。XFilter、YFilter は葉要素にたどり着くまでに状態遷移を繰り返すことで性能の低下を起こす。しかも深いレベルで複雑に要素が出現するため、途中まで状態遷移を行っても最終状態までたどり着かずバックトラックが多発する。これに対して提案手法は葉要素から検索を行うため、無駄な状態遷移が行われず高速に動作する。Depth や '/'、'*' の生起率の変化に対しても、提案手法は安定して動作し、従来手法よりも実行性能が高い。

第 2 実験ではクエリの葉要素はただ 1 つであり深いレベルでの検索失敗が少ないため、第 1 実験と比べて、従来手法の実行時間が大幅に短

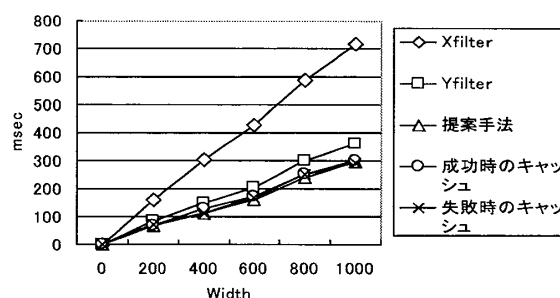


図3 Width の変化と実行時間(第 1 実験)

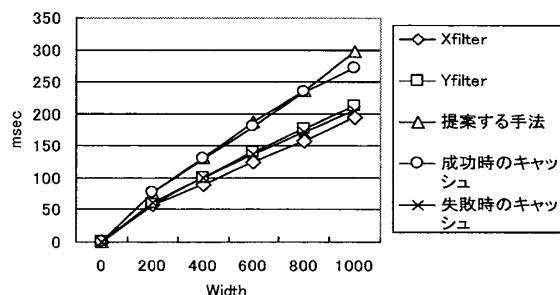


図4 Width の変化と実行時間(第 2 実験)

縮される。一方、提案手法の実行時間は大きな差がないが、従来手法と比べて処理速度は劣る。しかし、キャッシュ機能を利用することで、従来手法と同等の性能に改善できることが示された。

5. 終わりに

本論文ではボトムアップ式検索を利用したフィルタリング手法を提案した。実験評価によって、深いレベルで複雑な XML に対して、提案手法が XFilter、YFilter と比較して高速に動作することを示した。一方、要素の出現が一様な XML に対しては従来手法の方が高速に動作するが、キャッシュなどの付加機能で対応できることを示した。

今後の課題として、提案手法の一般化、クエリに対する規模耐性の追加が挙げられる。XML スキーマから XML の形を判断し、最速に動作するアルゴリズムを選択する手法などについて、研究を進める予定である。

参考文献

- [1] M. Altinel and M. Franklin, Efficient Filtering of XML Documents for Selective Dissemination of Information, In Proc. of VLDB, pages 53-64, September 2000.
- [2] Y. Diao, H. Altinel, M. Franklin, H. Zhang, and P. Fischer, Path Sharing and Predicate Evaluation for High Performance, ACM TOD, 2003.
- [3] W3C, XML Path Language (XPath) 1.0, <http://www.w3.org/TR/xpath>, November 1999.