漸増的なパストライ構築に基づく高速・軽量XML文書フィルタリング

Fast and Lightweight Filtering of Streaming XML Documents Using Incrementally Constructed Path-trie

萩尾 一仁* 御手洗 秀一† 石野 明‡ 竹田 正幸§

Kazuhito HAGIO Shuichi MITARAI Akira ISHINO Masayuki TAKEDA

XPath 式によるストリーム型 XML 文書フィルタ DXAXEN を開発したので報告する．DXAXEN は，パストライの動的構築に基づき，高速・省メモリかつ質問式の増大に頑健な性能を実現している．代表的なストリーム型 XML 文書フィルタである XMLTK と比較したところ，実行速度で 2〜5 倍高速であり，メモリ使用量も 5〜20%と圧倒的に少ないことを示した．

In this paper, we present a streaming XML document filter named DXAXEN which is based on incremental construction of path-trie. It runs very fast, and processes a large number of XPath queries efficiently. Experimental comparison with XMLTK, a well-known streaming XML document filter, shows that DXAXEN is 2–5 times faster and needs only 5–20 percent of memory.

1. はじめに

近年，XML の利用範囲が拡大し，センサデータや株式市場データなど，リアルタイム性の高い情報の伝達に XML データストリームが用いられる傾向にある．また，携帯電話や情報家電など，計算機としてのリソースが少ない環境における XML データ処理の需要も高まりつつある．XML データ処理の中でも，大量のデータから必要な部分を抽出するフィルタリングは，XML データ処理の基礎であり，非常に重要である．フィルタリングシステムの入力には，XPath が用いられ，XPath を高速に処理するための手法の研究が数多くなされている [2, 4, 3, 10].

1.1 先行研究

代表的な XML データストリーム処理器に YFilter [4] と XMLTK [3] がある．YFilter は，XFilter [2] のパス照合処理を改良したもので，質問式毎に非決定性有限オートマトン (NFA) を構築するのではなく，質問式全体の共通接頭辞パスを共通の状態として束ねる手法を用いている．これにより，XFilter と比べ，質問数に関する高い規模耐性を得ている．しかし，NFA は同時に複数の状態遷移を処理しなければならず，処理速度は遅い．一方で，NFA を等価な決定性有限オートマトン (DFA) に変換すると，一般に状態数が指数的に増大する．

そこで，XMLTK では，XML データ走査時に必要に応じて NFA を部分的に DFA に変換する lazy DFA [6] と呼ばれる手法を採用している．これにより，XMLTK は質問数に関して規模耐性が高く，高速なものとなっている．

*学生会員 九州大学大学院システム情報科学府 kazuhito.hagio@i.kyushu-u.ac.jp
†正会員 九州大学情報基盤研究開発センター mitarai@cc.kyushu-u.ac.jp
‡正会員 東北大学大学院情報科学研究科 ishino@ecei.tohoku.ac.jp
§九州大学大学院システム情報科学研究院 takeda@i.kyushu-u.ac.jp

著者らは [13] において，パストライとパスプルーニングを用いて DFA の状態数の増大を抑え，効率的にパス照合処理を行なう手法を提案した．パストライは，XML データ中の根から要素節点へ向かうパスに沿ったタグ名列の全体を表わすトライ構造である．この手法は，まず入力 XML データを一度走査してパストライを得ておき，これを用いて質問式に含まれる//や*を展開し，状態数の少ない DFA を直接構築するもので，フィルタ処理はこの DFA を用いた二度目の走査によって行なうことになる．パストライをいったん作成しておけば，DFA の状態数はパストライの節点数を超えないため，非常に高速なフィルタ処理が実現できる．

次に著者らは [12, 8] において，XML データを事前にパストライとバイナリ XML データに変換しておき，それを配信するという設定のもとで高速なフィルタ処理手法を実現し，XAXEN (eXtreamly-Accelerated XML filtering ENgine) と名付けた．変換の際，バイナリ XML データには，開始タグ相当部分に対応するパストライの節点へのポインタを埋め込んでおく．また，パス照合処理は，バイナリ XML データ走査時に実行するのではなく，パストライを対象にして質問式を得た段階で完了させておく．こうすることにより，バイナリ XML データ走査時に，埋め込まれたパストライの節点へのポインタを用いて，パス照合処理の結果を参照できる．なお，パストライに対するパス照合処理にはビット並列化手法 [9] を用いた NFA を採用している．

しかし XAXEN では，パストライとバイナリ XML データを作成する前処理が必要であり，このため，XML データをそのまま配信する設定でのフィルタ手法としては適さない．そこで，本論文では，漸増的にパストライを構築しながらパス照合処理を行なうことで，XAXEN の処理速度をそれほど低下させることなく，前処理を排することに成功した．本論文では，この手法を DXAXEN (dynamic XAXEN) と呼ぶことにする．

1.2 XPath 部分族

XAXEN および DXAXEN は，XPath 式の大きな部分族を処理できる．表 1 に示したのは，その例である．また，複雑な XPath 式は，基本となる単純な XPath 式の結合として表現できる．そこで，本論文では，基本的な XPath 式として線形パスパターン，パスパターン，XPattern に焦点を絞って，その効率的な処理手法について論じる．ここで，線形パスパターンとは，タグ名およびタグ名に関するドントケア記号 “*” からなる記号列を，親子関係を表わす “/” や先祖-子孫関係を表わす “//” で区切ったものをいう．また，パスパターンとは，二つの線形パスパターン π_1，π_2 を組み合わせた $\pi_1[\pi_2]$ という形式の XPath 式をいい，XPattern とは，これにキーワードの生起に関する論理式を付加したものである．

2. パストライを用いたフィルタ処理

2.1 パストライ

XML 木の根から要素節点へ向かうパスに沿ったタグ名の系列を**パス文字列**とよぶことにする．**パストライ**とは，ある XML 木におけるパス文字列全体の集合をトライ (trie) として表現したも

表 1. サポートする XPath 式の例

Table 1. Examples of XPath expressions DXAXEN supports.

/*/order//sender	線形パスパターン
//receiver[name]	パスパターン
/order//[contains(name, “mickey”)]	XPattern
//sender[count(//region) > 2]	集約
/order//address[street or region]	論理演算
//address[//region[//country and zipcode]]	入れ子
/order[//address]//zipcode	パス途中の述語

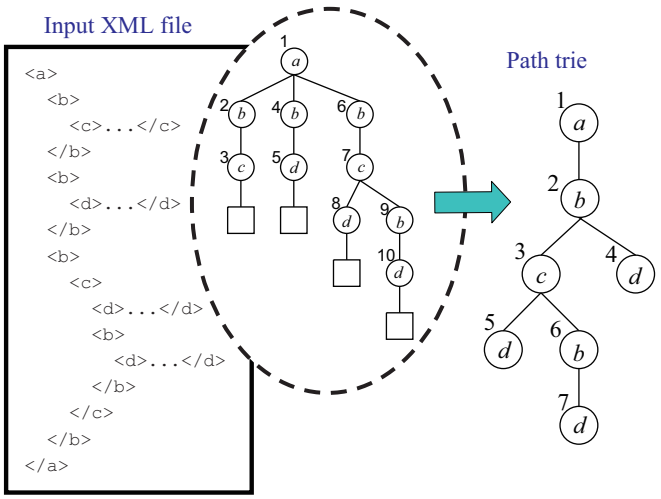


図 1. XML データと生成されるパストライ

Fig. 1. XML data and the path-trie DXAXEN generates.

のをいう．パストライは DataGuide [5] と呼ばれるデータ構造の一種であり，XML データに対する DataGuide ということができる．

XML データと生成されるパストライとの関係を図 1 に示した．図 1 中央は XML 木を示しており，正方形はテキスト節点を表わしている．右に示したのがパストライである．パストライの節点横の数はその節点の ID である．

2.2 パストライのサイズ

実際の多くの XML データが小さな DataGuide をもつことが示されており [7]，パストライも同様となる．一般に，対象となっている XML データが規則正しい構造をもつ，すなわち構造化されているほど，パストライは元のデータサイズに比べて小さなものとなる．(a) DBLP¶の XML データ，(b) XMark ベンチマーク [11] の xmlgen を用いて生成したランダムデータ，の二つについてパストライのサイズ等を表 2 に示す．

また，これらの XML データを先頭から処理する際にパストライのサイズが増加する様子を，図 2 に示した．データに含まれるタグ数及びタグ名数の増加の様子も併せて示している．(a) の DBLP の場合，途中 240MB 付近で傾向が変わっているが，それ以外の部分は比較的規則正しい構造をもっているため，パス文字列のバリエーションは XML データの先頭部分で出尽くしてしまい，パストライは飽和していることが分かる．(b) のランダムデータは，根の直下で 6 個のカテゴリに分かれ，それぞれが独立して異なるスキーマをもつために，パストライのサイズもそれに沿って変化している．

パストライを用いたパス照合

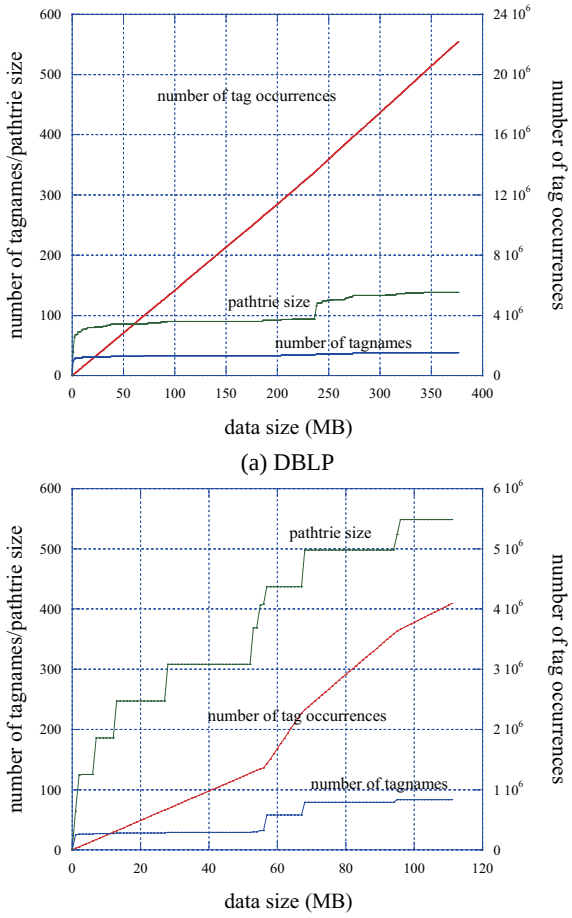
図 1 の XML データを例にとり，XML 木中，および，パストライ中における質問式 Q_1 = //b/c と Q_2 = a/b//d の生起の様子を図 3 に示す．この例では，質問式 Q_1 はパストライ中の節点 3，質問式

表 2. パストライのサイズ

Table 2. Path-trie sizes of two XML data.

	データサイズ	タグ数	パストライサイズ
(a)	377MB	22,215,944	166
(b)	116MB	4,096,360	549

¶http://dblp.uni-trier.de/



(b) ランダムデータ

図 2. パストライサイズの増加の様子

Fig. 2. Growth of path-tries.

Q_2 は節点 4, 5, 7 に生起することが分かる．さて，パストライにおける Q_1 の出現位置である節点 3 に対応する XML 木の節点は二つあるが，そのいずれとも Q_1 の出現位置である．Q_2 の出現位置についても，同様のことがいえる．2.2 節で述べたように，パストライのサイズは XML 木のサイズに比べて非常に小さい．したがって，XML 木に対してでなく，パストライに対してパス照合を行ない，それを参照することで XML 木における出現位置を得るようにすれば，パス照合処理に要する時間を大幅に削減できる．

3. 漸増的パストライ構築に基づくフィルタ処理

本節では，XML データを走査しながらパストライ構築とパス照合を同時に行なうことによって，XML ストリームに対して高速にフィルタ処理する手法を述べる．

3.1 漸増的パストライ構築とパス照合

再び図 1 で示した XML データを例にとり，本手法のアルゴリズムを述べる．いま，XML 木中の節点 1 から 7 までの処理が完了しているものとする．このとき，パストライは 1 から 4 までの節点をもつ．また，パストライを DFA として見立て XML 木の根からのパス上を走らせているため，パストライの節点 3 が現在の位置となる．XML パーサによって XML 木中の節点 8 が読まれると，パストライの節点 3 からそのタグ名 d で「状態遷移」する．今回のように，もしそのタグ名 d で遷移できなければ，遷移先となる新たな節点 5 を現在の節点 3 の子として作成する．その場合，パストライに根から節点 5 に至る新たなパスが発生したことになる

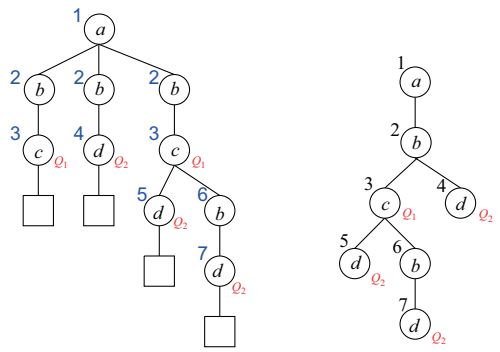


図 3. XML 木及びパストライにおける Q_1 = //b/c と Q_2 = a/b//d の生起．

Fig. 3. Occurrences of Q_1 = //b/c and Q_2 = a/b//d.

ため，すべての質問式についてこのパスに対する必要な照合を行なう．以上を XML 木のすべての節点を読み終えるまで繰り返すことで，漸増的なパストライの構築とパスの照合を同時に行なう．

2.2 節で述べたように，実際な XML データでは，パストライへの節点の追加が早い段階でほとんど行なわれなくなるため，このようにすることで高速なパスパターン照合が期待される．

3.2 キーワード照合

本手法では，XML パース処理とキーワード照合を同時に行なう．具体的には，Aho-Corasick のパターン照合機械 [1] を XML データのテキスト要素に対して適用することで複数キーワードの照合を行ない，タグの開始を表わす “<” を検出する度に独自の XML パーサを呼び出し，終了を表わす “>” に出合うまでの間，タグ名の切り出しや属性の処理等を行なう．これにより，高速なストリーム処理が可能となる．

3.3 NFA によるパス照合

パストライに新たな節点が追加された際に行なう，パス照合の詳細について述べる．ここでは二種類の手法を提案する．

DXAXEN-A: 1 番目の手法として，YFilter [4] のように，複数の質問のパターンを表わすトライ構造を作成し，これを NFA に変換して用いる方法が考えられる．パストライの各節点 n には，根から n までこの NFA を走らせた際の active な状態の集合をもたせておく．こうすることにより，パストライに新たな節点が追加された場合，その部分だけ NFA を動作させればよい．

DXAXEN-B: 2 番目の手法として，質問式ごとにビット並列化手法 [9] を用いて NFA を作成し，これを動作させる方法が考えられる．この方法は，質問の個数だけ NFA を走査させなければならない反面，NFA の状態遷移をビット並列化により高速に行なえるという利点をもつ．パストライの各節点 n には，根から n まで NFA を走らせた際の active な状態の集合を表わすビットベクトルをもたせておく．このビットベクトルのすべてのビットが 0 になると，それ以上照合を続けても受理することはありえないため，ビットベクトルを保持する必要はない．すなわち，パストライの各節点は，受理の可能性のある NFA の番号とそのビットベクトルの組をもつことになる．

次節では，これら二つのパス照合手法の比較を行なう．

4. 実装実験

XML データとして，2.2 節で述べた 377MB の DBLP のデータと 116MB の XMark ベンチマークのランダムデータを用いた．質問式には，XMLTK が任意の位置ステップ (location step) での述語の使用を許していないため，線形パスパターンを用いることとし，YFilter の pathgenerator[‖] によってランダムに生成した．ここで，//，*の生起確率はそれぞれ (1%, 1%) とした．実験に用いた計算機環境は，3.4 GHz Intel Pentium 4, 4.0 GB RAM, Linux (Kernel 2.6.18) である．

以下では，フィルタリング手法としての有効性を検証するために，実際の実行時間，メモリ使用量，スループットの三つの観点から XMLTK との比較実験を行なった結果を述べる．なお，XFilter, YFilter や XSQ は，いずれも，速度，使用メモリ量，質問数に関する規模耐性の点で XMLTK を下回るため [10]，ここでは，XMLTK のみを比較の対象とした．

4.1 実行時間の比較

表 3. 実行時間の比較
Table 3. Filtering time comparison.

(a) DBLP

質問数	経過時間 (sec)		
	DXAXEN-A	DXAXEN-B	XMLTK
1	3.30	3.34	8.99
10	3.50	3.55	8.99
100	4.11	4.15	13.82
1000	4.71	4.69	15.65
10,000	11.10	10.99	40.18
100,000	72.34	72.13	294.73

(b) ランダムデータ

質問数	経過時間 (sec)		
	DXAXEN-A	DXAXEN-B	XMLTK
1	0.78	0.79	1.76
10	0.84	0.86	1.99
100	0.94	0.95	2.29
1000	1.08	1.06	2.97
10,000	1.96	1.96	7.11
100,000	10.78	10.68	73.21

表 3 は，線形パスパターンを 1〜100,000 個ランダムに与えたときの実行時間を測定したものである．ここでは公平な測定とするため，すべてのプログラムで出力形式を質問式毎にマッチした節点の個数を求めるよう条件を合わせている．

DXAXEN と XMLTK を比較すると，おおよそ 2〜6 倍 DXAXEN が高速である．質問数が増えるほど，その差は顕著に現れる．

4.2 メモリ使用量の比較

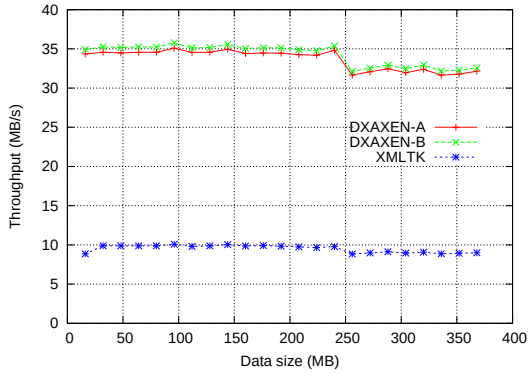
表 4 は，ランダムデータに対する質問数が 10,000 と 100,000 のときのメモリ使用量を表わしている．メモリ使用量は，論理空間上にどれだけメモリを確保したかではなく，プログラム実行中に実際にメモリにアクセスしたサイズの最大値を測定している．これは，例えば Linux 環境ならば，RSS (Resident Set Size) の値を参照することで測定できる．

DXAXEN を XMLTK と比較すると，質問数 10,000 のとき 1/5

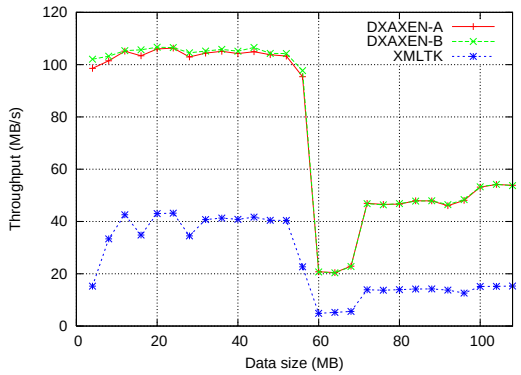
表 4. メモリ使用量の比較
Table 4. Memory usage comparison.

質問数	メモリ使用量 (KB)		
	DXAXEN-A	DXAXEN-B	XMLTK
10,000	14,400	7,268	38,780
100,000	35,216	17,452	369,036

[‖] http://yfilter.cs.berkeley.edu/code_release.htm



(a) DBLP, 質問数 10,000



(b) ランダムデータ, 質問数 10,000

図 4. スループットの比較
Fig. 4. Throughput comparison.

程度，質問数 100,000 では 1/20 程度しかメモリを使用しないことが分かる．DXAXEN-B は DXAXEN-A と比較して 1/2 程度のメモリ使用量となった．

4.3 スループットの比較

図 4 は質問数が 10,000 のときのスループットを比較したグラフである．x 軸は XML データの先頭からのデータサイズを表わす．

(a) の DBLP では，データが比較的規則正しいスキーマをもっており，XMLTK と DXAXEN の両方に有利な条件であるが，DXAXEN が圧倒的に高いスループットを示している．

(b) のランダムデータでは，どの手法もスループットが安定しない．データ前半の XMLTK の局所的なスループットの低下は，図 2 (c) に示したパストライの節点数の増加に対応しており，節点の追加によるパス照合処理が増大していると考えられる．これより，DXAXEN はパストライの節点数の増加に対して比較的堅牢であることが分かる．後半部分の全体的なスループットの低下は，図 2 (c) に示したデータ中のタグの出現頻度に対応しており，テキスト部分の処理に対してタグ部分の処理に時間がかかるためと考えられる．しかしながら，DXAXEN は常に XMLTK より高いスループットであることが分かる．

5. まとめ

本論文では，漸増的なパストライ構築に基づくストリーム型の XML フィルタリング手法を提案した．XAXEN で必要であった前処理を排しつつも，なお XMLTK と比較して高い性能をもつことを実験により示した．

[文献]

[1] Aho, A. V. and Corasick, M.: Efficient string matching: An Aid to Bibliographic Search, *Comm. ACM*, Vol. 18, No. 6, pp. 333–340 (1975).

[2] Altinel, M. and Franklin, M.: Efficient Filtering of XML Documents for Selective Dissemination of Information, *VLDB'00*, pp. 53–64 (2000).

[3] Avila-Campillo, I., Green, T. J., Gupta, A., Onizuka, M., Raven, D. and Suciu, D.: XMLTK: An XML Toolkit for Scalable XML Processing, *PLANX'02* (2002).

[4] Diao, Y., Altinel, M., Franklin, M., Zhang, H. and Fischer, P.: Path sharing and predicate evaluation for high-performance XML filtering, *ACM TODS*, Vol. 28, No. 4, pp. 467–516 (2003).

[5] Goldman, R. and Widom, J.: DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases, *VLDB'97*, pp. 436–445 (1997).

[6] Green, T. J., Gupta, A., Miklau, G., Onizuka, M. and Suciu, D.: Processing XML streams with deterministic automata and stream indexes, *ACM TODS*, Vol. 29, No. 4, pp. 752–788 (2004).

[7] H., L. and D, S.: XMill: an efficent compressor for XML data, *SIGMOD'00*, pp. 153–164 (2000).

[8] Mitarai, S., Ishino, A. and Takeda, M.: Light-weight acceleration for streaming XML document filtering, *SWOD2007* (2007).

[9] Navarro, G. and Raffinot, M.: *Flexible pattern matching in strings: Practical on-line search algorithms for texts and biological sequences*, Cambridge University Press (2002).

[10] Peng, F. and Chawathe, S. S.: XSQ: a streaming XPath engine, *ACM TODS*, Vol. 30, No. 2, pp. 577–623 (2005).

[11] Schmidt, A., Waas, F., Kersten, M. L., Carey, M. J., Manolescu, I. and Busse, R.: XMark: A benchmark for XML data management, *VLDB'02*, pp. 974–985 (2002).

[12] 御手洗秀一，石野　明，竹田正幸：パストライを用いた高速軽量な XML 文書フィルタリング，*DBSJ Letters*, Vol. 6, No. 1 (2007).

[13] 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 (2006).

萩尾 一仁　Kazuhito HAGIO
九州大学大学院システム情報科学府情報理学専攻修士課程在学中．日本データベース学会学生会員．

御手洗 秀一　Shuichi MITARAI
九州大学情報基盤研究開発センター特任助教．2002 九州大学大学院システム情報科学府情報理学専攻修士課程修了．修士（理学）．半構造データベースに関する研究と開発に従事．日本データベース学会会員．

石野 明　Akira ISHINO
東北大学大学院情報科学研究科助教．1999 北海道大学大学院工学研究科電子情報工学専攻博士課程修了．博士（工学）．半構造データベース，自律型ロボットに関する研究と開発に従事．日本データベース学会会員．

竹田 正幸　Masayuki TAKEDA
九州大学大学院システム情報科学研究院教授．1989 九州大学大学院総合理工学研究科情報システム学専攻修士課程修了．博士（工学）．系列データの高速パターン照合処理，大規模系列データからの知識発見に関する研究と開発に従事．