インクリメンタルに更新可能なXPushマシンの性能向上と応用に関する考察

武川 俊†† †† 福田 直樹††† 石川 博†††
† 職業能力開発総合大学校情報システム工学科
†† 静岡大学創造科学技术大学院自然科学系教育研究科学専攻
††† 静岡大学情報学部情報科学研究科

あらまし XMLフィルタリングにおける部評価手法にGuptaらのオートマトン（XPushマシン）がある。これまでに我々は、オートマトンのマージに基づいて、XPushマシンが部分フィルタ交換を行うように改良してきた。本稿では、このXPushマシンを既存のXMLフィルタリングエンジン（YFilter）に連携させる方法について述べる。

キーワード XMLフィルタリング, XPath, ストリーム処理

Performance Improvement and Application of An Incrementally Updatable XPush Machine

Hajime TAKEKAWA†††, Naoki FUKUTA†††, and Hiroshi ISHIKAWA†††
† Department of Information System Engineering, Polytechnic University
†† Department of Information Science and Technology, Educational Division, Graduate School of Science and Technology
††† Department of Computer Science, Faculty of Informatics, Shizuoka University

Abstract In XML filtering, Gupta and Suciu proposed an automaton (called an XPush machine) for stream processing of XPath queries with predicates. Recently we improved the XPush machine based on merging automata to have a partial filter exchange function. In this paper, we will describe how to associate this XPush machine with ordinary XML filtering engine as YFilter.

Key words XML filtering, XPath, stream processing

1. はじめに

ニュース記事やセンサデータなどのXMLデータが逐次発生している。これ専XMLデータを処理するシステム（例、SDIやnotificationシステム）において、XMLフィルタリングは主要な機能であり、その機能にはXMLデータごとに多くのフィルタを効率よくマッッチングする能力が要求される[1], [2]。

XMLフィルタリングにおけるマッチングは構造マッチングと部評価からなる。既存のXMLフィルタリングエンジンのYFilter[3]（現状ではこの研究分野の到達水準の1つ）は、構造マッチングには優れているが、単純な部評価しかできない。一方、部評価だけに特化したXPushマシンが提案されている[4]。

XPushマシンはXMLドキュメントのストリームに対して、複雑な部を持つフィルタ条件の巨大集合を評価できる。

XPushマシンではフィルタ条件をXPath[5]で記述し、この条件をXPathフィルタという。XPathフィルタの部は、フィルタごとに1つ以上の述部を許させ、またマッチした位置の特定もできない。しかしORなどの論理演算、\等の比較演算が利用でき、子や子孫軸を許す。またXPushマシンは、述部評価のためのトランスパース[6]によるバックトラックも、XMLデータへのランダムアクセスも不要であるため、XMLデータごとに1回のストリーム処理で全フィルタの述部評価を完了する。

XPushマシンはXPathフィルタごとにAFA (Alternating Finite Automaton[7])を構築し、評価しているデータの構造をもとに、それからAFAから1つの決定性PDA (Push Down Automaton)へと姿を変える。それが「アクティブなAFA状態の全てをAFA上で転送させる」という状態変換処理からなり、さらにその結果はXPushマシンの内部状態 (XPush状態)や状態変換表としてキャッシュされる。そのため同じ (構造の)XMLデータに対するXPushマシンによる述部評価は効率的である。

ところでXPushマシンは更新できないために、1つのフィルタを削除することであっても、XPushマシン全体の再計算
(構築)が必要となる。すなわちオートマトンの更新コストはAF (フィルタ)の数に依存します。

そこで我々はこれまでに、XMLフィルタリングにおけるオートマトンのマージに基づいたインクリメンタルな更新手法を提案してきた[8]。その手法は、GuptaらのXPushマシンと異なり、順序付きハッシュキューを用いることによってXPushマシンの更新コストを削減させている。そのためフィルタ（ユーザ嗜好、あるいは問合せ）がダイナミックである場合にも役に立つ。

本稿では、インクリメンタルに更新可能なXPushマシンで、YFilterエンジンの述部評価を処理させることについて考えていく。

先に述べたように、従来方式のXPushマシンはマッチした位置が特定できない。そこでXMLベースに組込んだXPushマシンを提案する。このXPushマシンはXML要素ごとに述部評価結果が得られ、さらにその評価結果はSAX (Simple API for XML) [9]のstartElementイベント処理から利用できる。このようなXPushマシンを用いてYFilterの述部評価を強化する。

またXPushマシンのアクティブなAFA状態を減らし、状態列挙処理を改善する試みも行う。これは、XPushマシンのキャッシュが不十分であっても、つまりヒット率が高い間でも、XPushマシンの述部評価速度が、改良前のYFilterの述部評価速度と同等以上になることを目的としている。具体的には、XPushマシンのAFA集合を決定性PDAへ変換する前に、フィルタ間のサフィクスの共通性に基づいて1つのAFAに結合させる方法を提案する。

本稿の構成は以下のとおりである。2章で、XPushマシンの概要について述べる。3章で、XPushマシンを、XMLフィルタリングの述部評価に用いる際の問題点を述べる。4章で、XMLベースに組込んだXPushマシンによる述部評価について述べる。なお提案方式の評価については今後の課題である。

2. 従来方式：XPushマシンの概要

この章ではXPushマシンの概要について述べる。またYFilterのXPushマシンとの違いにも簡単にふれる。なおXPushマシンの決定性PDAへの変換と最適化[4]、インクリメンタル更新[8]については省略する。

XPushマシンは、ブッシュダウンオートマトンである。XPushマシンは、あらかじめXPathフィルタにオートマトン(FA)を構築し、その後XMLデータごとにそれら構築済みのAFA集合をボトムアップに状態遷移させる。状態遷移の際、バックトラックも、XMLデータへのランダムアクセスも必要としない。XPushマシンは、SAXベースのイベント処理で全XPathフィルタの評価を完結する。

例えば図1aのXPathフィルタが与えられると、図1bのように根が受理状態であるAFAが構築される。その後XMLデータごとに、図2のコールバック関数に従って、AFA集合の状態遷移が発生し、図3のように、XPushマシンのカレント状態(current state)qとスタックsが変化する。

XPathフィルタは、図1aの式P1、P2のようにXPathで記述される。XPathフィルタは構造ナビゲーション(structure navigation)と述部(predicate)で構成され、[1]の部分が述部である。XPathフィルタの述部は、XPath[10]のWhere句のように、論理演算や軸を用いたパス表現を許し、パス上の葉の全てに値との比較演算が存在する。ただしYFilterなどで扱われるフィルタとは異なり、1つのXPathフィルタには複数の述部を持たせることができない。

XPushマシンはXPathフィルタの内部表現として、AFAを利用する。AFAは、XPathフィルタが論理演算や軸を許すように、それに対応している。たとえば図1bのA1、A2では、AFA状態の5がorに対応し、1と4のように閉路によって子孫節が現れる。一方YFilterでは、述部での論理演算や軸
は扱えないため，$P_2$の述部を評価できない。

XMLデータは，AFM集合の状態を遷移させるための入力データとして扱われる。XMLデータは，データの先頭からタグ名ごとにSAXイベントとして分解され，SAXイベントごとにシーケンシャルにイベント処理される。なおXPushマシンのイベント処理は，標準のSAXとは少し違いがある。違いとしては，相対性を要素と一緒に扱うこと，1つの要素のテキスト値を含むイベントを使用すること，またYFilterも，テキスト値をテキスト値に変更しようとする要素の関数値として扱う点で，標準のSAXと異なっている。

XPushマシンは，SAXイベントに図2のコールバック関数で処理されている。コールバック関数は，状態とスタークトクを格納する変数として扱い，AFM集合のポリトロップな状態を実現している。状態遷移は，各AFAのアクティプ状態の集合として格納され，アクティプ状態がAFAの受理状態となっている間，taccept関数によるXPathフィルタの述部評価の結果はtrueとなる。

たとえば図3では，$q$の値が最終的に1と4となっている。これらの値は図1bのAFAのアクティプ状態の番号を意味している。図1bにおいて，AFA集合1と4は，それぞれ$A_1$と$A_2$の受理状態である。したがって，入力データに対してXPathフィルタ$P_1$と$P_2$は，ともにtrueと見なされる。

述部処理の作業は，コールバック関数のendElementブローシャージャに集中している。endElementブローシャージャでは，まずtop関数によって，状態を根方向に$a$（評価している要素名あるいは属性名）の遷移をさせ，その結果を$q_{aux}$に代入する。次にpop関数によって，スタックから状態を取り出し，それを$q_a$に代入する。最後に$P_{add}$関数によって，$q_{aux}$と$q_a$の和集合をもと，その結果を$q$に代入する。つまり，endElementブローシャージャでは，$q_{aux}$は$a$を頂点とした枝に対する述部評価結果が代入され，そして$q_a$は$a$と兄弟であるかつ$a$より前にでてくる枝の述部評価の合計が代入される。

なおtop関数と$P_{add}$関数の結果は，XPushマシンの内部状態（XPush状態）や状態遷移表としてキャッシュされる。

3. XPushマシンの問題点

この章では，XPushマシンを，XMLフィルタリングエンジン（YFilter）の述部評価に用いる際の問題点を述べる。

状態遷移をキャッシュするまでのXPushマシンは，フィルタの共有を述部の集にしか利用していない。そのため，例えば全ての述部が同じ形であっても，全ての形が異なる場合と同様に，キャッシュするまで遅れているフィルタ数に依存する。

アプリケーション（ここではYFilter）側で，同じ述部を取扱うことも考えられるが，部分的な一致を取り除くことは難しい。なおこのことに関する最適化は，従来方式[4]には含まれていない。

またXPushマシンは，endDocumentイベントのtaccept関数によって受理を確認するため，マッチした位置が特定できない。たとえ終了要素の順序（つまりポジショニング）で受理を確認できるようになったとしても，イベント処理のタイミングが問題点として残る。なぜならYFilterやlazyDFA[2]がそれを示すように，XMLフィルタリングの構造マッピングは開始要素の順序（つまりブロードコーディング）で受理をするためである。もしポストオーダーの述部評価を，startElementイベントで利用しようとすると，述部評価のためのトランスバースやXMLデータへのランダムアクセスが必要となり，XPushマシンを用いる利点がなくなる。

4. 提案方式：XMLバーサに組込んだXPushマシンによる述部評価

この章では本提案方式のXMLバーサに組込んだXPushマシンの概要を述べてから，フィルタ間のサフィックスの共通性に基づきXPushマシンの性能を向上させる方法，そしてXPushマシンの述部評価結果をフィルタリングエンジンに渡す方法について述べる。

4.1 XMLバーサに組込んだXPushマシンの概要

このではXMLバーサに組込んだXPushマシンの概要について述べる。

図4に，XPushマシンとYFilterによるXMLフィルタリングシステムのアーキテクチャを示す。システムはXPathバーサ，XMLバーサ，フィルタリングエンジン，データ配信の4つのモジュールで構成されている。各モジュールともYFilterのモジュールをベースに拡張している。ただデータ配信モジュールについては，YFilterからの変更はない。

本システムのXPathバーサは，YFilterのXPathバーサにクエリ接続機能を追加したモジュールである。このXPathバーサにクエリ集合が渡されると，各クエリ$Q_0$が含まれる0個以上の全ての述部がXPathフィルタ$P_0$として抽出され，各クエリ$Q_i$はその抽出箇所が0接続の単純なクエリで書き換えられ，クエリ$Q_i$とされる。抽出された$P_i$集合はXMLバーサに組込まれたXPushマシンに渡され，$Q_i$はさらに文脈解析されてから，フィルタリングエンジンに渡される。

-231-
図 4 XPush マシンと YFilter による XML フィルタリングシステム
Fig. 4 Architecture of a filtering system using XPush machine and YFilter

図 5 クエリの書き換え
Fig. 5 Rewriting queries

4.2 XQuery フィルタのサフィックス共有
ここではフィルタ間のサフィックスの共通性に基づいた XQuery フィルタの性能を向上させる方法を述べる。

3 章で述べたように、従来の XQuery フィルタは XQuery フィルタが同じ式である場合で、全ての式が異なっている場合と同様に、キャッシュするまでの時間は XQuery フィルタの数に依存する。そこで本システムでは、XQuery フィルタ間のサフィックスの部分的な一致を取ることによって、AFA に対する状態遷移の計算コストを削減させる。そのために XQuery フィルタ集合からサフィックスが共有された AFA を 1 つだけ構築する方法を提案する。

たとえば図 5b の P1 と P2 が選ばれると、図 6 の AFA が得られる。図 7 の q の 5 倍と 6 倍の値はそれぞれ 3.7 と 2 であり、従来方式の図 3 と比べて、アクティブな AFA 状態の数が削減されていることがわかる。
フィルタの構造サフィックスの変更点について述べたが、それだけでは不十分である。本提案の XML バークに組込まれた XPush マシンには、さらに図 8 のコールバック関数を用いる必要がある。

このコールバック関数には次の 3 つの改良点がある。

(A) $t_{accept}$ 関数を endElement イベントの $t_{pop}$ 関数の結果 $q_{aux}$ に対して利用している。ただしこの処理は要素に対して行い、属性にはその必要はない。

(B) A によって XPath フィルタの評価結果が XML 要素ごとに得られる。評価中の XML データの要素数だけ一時記憶する必要があるため、配列 $r$ を増やしている。

(C) プリオーダの処理で評価結果が利用できるように、$r$ には開始要素の順序で格納している。変数 $eid$ とスタック $e$ を利用して、$r$ に格納する際の添え字を計算している。

まず改良点 A と B について考える。図 9 の $q_{aux}$ のように、$t_{pop}$ 関数によって $q_{aux}$ が得られる。さらにこの $q_{aux}$ に対して、$t_{accept}$ 関数によって $r$ が得られる。$q_{aux}$ は、3 章で説明したとおり、$a$ を頂点とした枝に対する述部評価結果である。そのため $r$ には、XML 要素ごとに XPath フィルタの評価結果が代入される。

次に改良点 C について考える。図 9 の $r$ は、ポストオーダの処理で得られた終了要素の出現順であり、開始要素の出現順ではない。一方、図 10 のように、$e$ から取り出される $pop(e)$ を $r$ 上の格納位置とすれば、$r$ に格納された述部評価結果は、開始要素順となることがわかる。なお図 10 では $r$ の 1 要素ごとに event に埋め込んでいるが、実際にはその必要はなく、配列全体がインデックスとしてフィルタリングエンジンに渡されるばよ。

5. まとめ

XPush マシンを、XML フィルタリングの述部評価に用いる際の問題点を明らかにし、その問題点を解決するために、「XML
図10 開始要素順での述部評価結果の格納

Fig.10 Storage of a predicate evaluation result by start element order

パーサに組込んだ XPush マシン」を提案した。

今後の課題として、XPath フィルタのサフィックス共有アルゴリズムの開発、提案方式の評価を行う予定である。

謝辞

本研究の一部は科学研究費補助金基盤研究（B）（課題番号19300026）の助成による。

文献


-234-