DEWS2006 1C-i6

パスプルーニングと決定性有限オートマトンを用いた大規模かつ高速なXQuery処理システムの実装

池末　修也† 　石野　明†† 　御手洗秀一††† 　竹田　正幸†††,††††

† 九州大学大学院システム情報科学府
†† 九州大学大学評価情報室
††† 九州大学大学院システム情報科学研究院　情報理学部門
〒 812-8581 福岡市東区箱崎 6-10-1
† 科学技術振興機構　戦略的創造研究推進事業
E-mail: †{s-ikesue,mitarai,takeda}@i.kyushu-u.ac.jp, ††ishino.uoc@mbox.nc.kyushu-u.ac.jp

あらまし　本稿では，XML データに対する大規模な質問式群を高速に処理する手法を提案する．この手法は，XML データに対して複数の XQuery の部分族の質問式を同時に処理できることが特徴である．最初に XML データ走査機を用いて XML データの効率のよい解析を行い，XML データ中に存在する全てのパスの構造を得る．これを用いてパスプルーニングを行い，複数の質問式から 1 つの決定性有限オートマトンを構築し，各状態に応じた処理をそれぞれ定める．これにより，複数の質問式を同時に効率よく処理する事ができることを示す．さらに，実際に実装を行い，大量の質問式に対する実験結果も示す．

キーワード　XQuery，XMLDB，大規模 DB，パスプルーニング，パススキーマ，決定性有限オートマトン

An Implementation of Massively and High-Perfomance XQuery Processor By Using Path Pruning and A Deterministic Automaton

Shuya IKESUE†, Akira ISHINO††, Shuichi MITARAI†††, and Masayuki TAKEDA†††,††††

† Department of Informatics, Kyushu University
†† Office for Information of University Evaluation
††† Department of Informatics, Kyushu University
Hakozaki 6-10-1, Higashi-ku, Fukuoka, 812-8581 Japan
† SORST, JST
E-mail: †{s-ikesue,mitarai,takeda}@i.kyushu-u.ac.jp, ††ishino.uoc@mbox.nc.kyushu-u.ac.jp

Abstract　We consider a high-performance XQuery processor for a large number of queries. The distinct feature of our works is that we process many XQueries at once by using a DFA. The DFA is obtained from path pruning for the XPath expressions in XQueries. The DFA processes the XML data and outputs results for each queries by processing appropriate tasks corresponding with the queries. We also implemented the XQuery processor, validating the techniques and providing performance results.

Key words　XQuery, XMLDB, Massively DB, path pruning, path schema, DFA

1. はじめに

W3C によって策定が進められている XQuery 1.0 [15] も勧告候補が公開され，いよいよ本格的に XMLDB の利用が広がりつつある．XQuery 処理システムとして，これまでにも数多くの研究 [2], [3], [9]〜[11], [17] が成されている．

近年，XML の持つ柔軟な構造による記述の容易さから，遺伝子情報，天体情報などの科学的データをはじめ，さまざまな大規模データが XML の形式で作成されている．さらに，インターネットの普及に伴い RSS など，XML は様々な方面へと活用されはじめている．このように，急速に広がる XML の応用に対して，大規模な XML データを効率よく処理する XQuery 処理システムの開発は次世代の重要な基盤技術であるといえる．

我々はこれまでに XQuery の部分族を決定性有限オートマ

トン (DFA) を用いて効率的に処理する手法を提案した [18]. XML 文書の走査に用いるシステムは，XML データを先頭から順に一方向に走査しながら処理を行うストリーム指向によるもので，結合処理や親軸の取り扱いなど難しい問題もあり，XQuery の部分族をここでは取り扱うにとどめるが，その反面，少ない領域で高速に処理が行えるという利点がある．

本稿では，XML データベースに対して大量の問合せが同時にあるような場合を考える．大量の質問式を処理する場合，それぞれの質問式に対して DFA を構築し XML 文書を 1 文字読み込むたびにそれぞれの DFA を動作させて処理する方法が考えられる．しかし，この方法では質問式の数に比例した処理時間がかかるという問題がある．

そこで本稿では，文献 [18] の手法を拡張し複数の質問式から１つの DFA を構築し，それぞれの状態に処理を設定することで大規模な XML データに対して大量の質問式を同時に効率よく実行する XQuery 処理システムを提案する．ストリーム指向では膨大なデータを扱う場合に，そのデータを読み込むための時間が問題となると考えられるが，ただ一つの DFA だけを用いて複数の質問式を同時に処理することができ，大量の質問式を効率的に処理することができることを示す．

この手法を実際に実装し評価実験を行った．また，比較実験として市販のデータベース管理システムである Tamino との比較や XPath 処理システムである XMLTK との比較実験を行った．

本稿の構成は次の通りである．2 節では，関連研究について述べる．3 節では，これまでに著者らが提案した XQuery 処理手法について述べる．本稿で扱う XQuery の部分族を定義し，XML データを効率よく読み込むための XML データ走査機について述べ，パスパターンの照合および XQuery の質問式の処理手法を与える．また，XQuery 処理を拡張した複数 XQuery の処理手法を提案する．4 節では，これら手法を実際に実装し，実験を行った結果を示す．既存の手法との比較実験も行う．5 節では，複数の XQuery 質問式の処理を行った結果を示し，最後に 6 節でまとめを行う．

2. 関 連 研 究

XQuery の処理手法として，これまで Zhang ら [17] が関係データベースに対してマッピングすることで XQuery を処理する手法が提案された．Diao と Franklin [2] は，XQuery の一部をフィルタリング処理する手法を述べている．Josifovski ら [9] は表を用いた XQuery の処理手法を述べている．Ludäscher ら [11] はトランスデューサーを拡張し on-the-fly な XML ストリームに対する XQuery の処理手法を述べている．Florescu ら [3] は，完全な XQuery 処理システムを述べている．

これらに対して，我々はの手法は DFA を用いたストリーム処理であり，XML データ走査機に SAX の代わりに Aho-Corasick 法を応用したものを用いて高速にデータの解析をおこなうのが特徴である．

XQuery 処理は，パスに対しての処理を定めたものなので，XPath を効率よく処理することが XQuery 処理全体の高速化となる．XPath を処理する手法としては，森川ら [21] が NFA を効率よく実行することによってパスを処理する手法を述べている．また，Green ら [6] [7] が NFA から実行中に DFA を構築しながら大量のパスを効率よく同時に処理する手法を述べている．NFA から作られる DFA は，状態数が指数関数的に増大し大量のメモリを必要とすることが問題となる．

我々の手法の特徴としては，NFA を構築してからそれを DFA に変更するのではなく，パススキーマを用いてパスプルーニングを行うことで質問式から直接コンパクトな DFA を構築することである．

パスに対する処理としては，入力 XML 文書を走査して得られたデータ構造から，パスに対して最適化の処理を行う方法が Goldman と Widom [4] や，より詳しくは McHugh と Widom [12] によって述べられている．

我々の手法で用いられるパススキーマは，ここで述べたデータガイドの一種とみなすことができるが，我々の手法は，正規表現を含むパスを展開した際に，増加するパスから 1 つの DFA を構築することによりパスの数に依存しない処理を行うことができるという点で異なる．

3. パスプルーニングと DFA による XQuery 処理

我々は XQuery の部分族をパスプルーニングと DFA を用いて効率的に処理する手法を文献 [18] において提案した．ここでは，その手法について簡単に述べ，その性能についての評価結果を示す．

3.1 XQuery の部分族

本稿では XQuery [15] の定義を，質問対象である XML データを先頭から順に一度だけ読み込むことによって処理することができるものに限定する．これにより，質問式を処理するために必要な，対象データの読み込み回数が質問式によらず常に 1 度だけとなる．このことは，大量の質問式を同時に処理することを考える際に，質問処理時間全体が最悪となるものに依存することからも大事な点である．

まず，パスとして，子軸と子孫軸のみに限定し，親軸や先祖軸は扱わない．親軸や先祖軸の情報は処理を行っている対象データ中の位置よりも前方にあり，さらにそこからの子軸や子孫軸へと辿った情報は対象データ中の前方にも後方にも現れうる．これは対象データを先頭から順に一度だけ読み込むことで処理を行う本稿の手法とはそぐわないばかりか，親軸や先祖軸を扱った場合，処理時間がパスの大きさに対して指数関数的に増大する場合があることが知られている [5].

また，条件式はパスの最後だけに制限される．条件式がパスの途中に現れる場合，その条件式中で指定されるノードはパスの最後で指定されるノードから見て親軸へと辿った先にあり，先の議論の通り，これらは本稿の手法では扱わないこととする．

パスの指定以外にも，比較演算，算術演算，論理演算，統計関数といった関数を扱う．

最後に，FLOWR(for-let-order-where-return) 形式を扱う．ただし，並べ替え (order) 処理は本稿では扱わない．また，let,

where, return 中のパスは，外の for で指定されたパスに対して子軸，子孫軸を用いたものに制限される．すなわち，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いわゆる結合処理は扱わない．

3.2 XML データ走査機

本稿では，タグやキーワードの検出を行う XML データ走査機として，複数文字列照合アルゴリズムのひとつである Aho-Corasick 法 [1] を拡張語頭符号法のもとで表現された文字列に対して応用したものを用いる [14], [19], [20].

XML データ走査機は，入力データのサイズに対して線形時間でデータ全体を処理することができ，かつ，タグやキーワードの検出と識別を同時に行うにもかかわらず，既存の SAX パーサーと同等の速度で動作する．

3.3 パスパターンの照合

XQuery の質問式はパスによって示されるノードに対する操作を指定したものであるといえる．XQuery の処理の対象は，全データのうちのある部分であり，対象となる部分をいかに高速に見つけるかが処理全体の高速化につながる．

質問式中のパスには，子孫軸を表す//やワイルドカード文字*が含まれることがある．そこで，質問式中のこのようなパスを**パスパターン**と呼び，また特に，//や*を含まない，タグ名と子軸のみを用いて構成されたパスパターンを**基礎パスパターン**と呼ぶことにする．

条件式を含むパスの基本形として $p[q]$ を考える．この $p[q]$ は，まず，根からの経路が p である節点 x が存在し，かつ，その節点 x からの経路が q となる節点 y が存在する場合の節点 x を指定している．このとき，パス p のことを**親パス**といい，パス q のことを**子パス**と呼ぶことにする．図 1 に親パスと子パスの関係を示す．

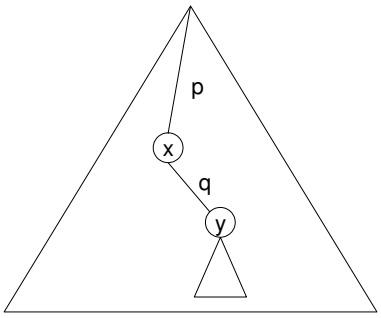

図 1　親パスと子パス

パスパターンに対応するノードを XML データから検出するために，パスパターンに対する NFA を構築し用いる数多くの手法 [6]〜[8], [21] が提案されている．それに対して我々は，対象となる XML データを事前に調べて XML データ中に存在する全てのパスの構造を得る．ここで言う全てのパスの構造とは，XML データ中の根から到達する全てのノードへのパスで，かつ重複するものは 1 つにまとめたものである．これを**パススキーマ**と呼び，パススキーマを用いることでパスパターン中の//や*といった部分をパススキーマ中に現れるタグ名と子軸のみで表される基礎パスパターンに変換する．得られた基礎パスパターンに対する DFA を用いることで対応するノードを検出する手法を提案し，さらに，複数のパスパターンを 1 つの DFA によって同時に処理できることを示す．

パスパターン中の//や*といった部分を基礎パスパターンに変換することを**パスプルーニング**と呼ぶことにする．

まず 3.3.1 節で非決定性有限オートマトンによるパスパターンの照合手法を述べる．この手法では，パスパターンの照合時間が質問式のサイズに依存するという問題がある．そこで，次に 3.3.2 節で NFA を構築することなく，パスパターンを前処理としてパスプルーニングすることで基礎パスパターンを得て，それから 1 つの DFA を構築する手法を述べる．

3.3.1 非決定性有限オートマトンによるパスパターン照合

パスパターン $p[q]$ に対するパスパターンの照合機は，次のような 2 つの非決定性有限オートマトンからなる．まず，1 つ目は，パスパターン p に対応する NFA を N_p，同様に q に対応するものを N_q としたとき，この N_p と N_q を ε 遷移で繋いだものである．この NFA を N_{p_q} とする．2 つ目は，パスパターン q の転置パターン q^R に対する NFA で N_{qR} とする．

この 2 つの NFA が受理状態となるときパスパターン $p[q]$ が存在する．

ここで N_{qR} が必要なのは，N_{p_q} が受理状態となったとき p に対応する節点 x が存在することを保証するためである．

例えば，a，b をタグ名とするとき，パスパターン $a//b[a//b]$ を考える．図 2 に $a//b[a//b]$ に対するパスパターン照合機を示す．このとき，パスパターン $a//b[a//b]$ とパス $aabbabab$ との対応を考えると，$aabb[abab]$ と $aabbab[abb]$ の 2 つが考えられる．この $aabb[abab]$ と $aabbab[abb]$ の 2 つは，N_{p_q} が受理状態ならば N_{qR} も受理状態となる．

このようにして非決定性有限オートマトンを用いた場合のパスパターンの照合は行われる．

複数の質問式を考えた場合，NFA によるパスパターンの照合では，質問式中に含まれるパスパターンごとに対応する NFA を用意する必要があり，かつ，それら全てを開始タグや終了タグが現れる度に動作させる必要があるので，パスパターンの数に比例する検索時間がかかるという問題がある．

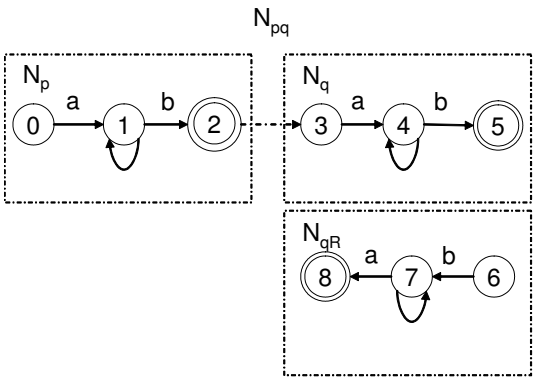

図 2　$a//b[a//b]$ に対するパスパターン照合機

3.3.2 パスプルーニングによる決定性有限オートマトンの構築

節 3.3.1 でパスパターンの照合は，NFA を用いて行うことができることと，その問題を示した．

これに対して，NFA を構築することなく，質問式中のパスパターンに対してパスプルーニングを行うことで基礎パスパターンを得て，その得られた基礎パスパターンから 1 つの DFA を構築することでパスパターンの照合を行うことができることを示す．

ある XML データから得られたパススキーマが図 3 である場合を考える．このとき，パスパターン//text, //mail[//to] はパススキーマを用いることで図 4 のように基礎パス/site/regions/mail/text，/site/regions/keyword/text と/site/regions/mail/to に変換することができる．

基礎パスパターンにおいては，パスの共通する部分を束ねることができ，図 5 に示されるような DFA を容易に構築することができる．また，基礎パスに変換することにより上で述べた N_{qR} のような処理をする必要がなくなる．

基礎パスを受理する DFA の状態数は，最大でもパススキーマのノード数を超えることはなく，DFA は単純な表参照を用いて実装できるので高速な動作が可能である．

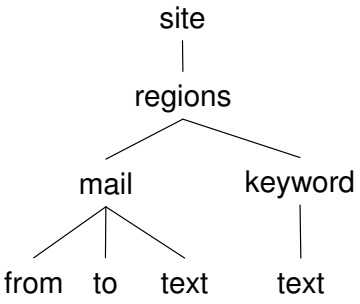

図 3　パススキーマ

図 4　パスの変換の例

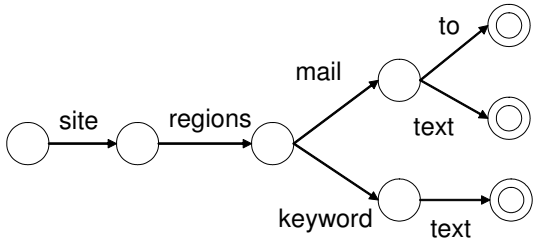

図 5　複数パスパターンから得られる DFA

3.4 XQuery 処理

上述の通りパスパターンに対しては DFA を用いることでその出現を検出できることをみた．このとき，あるパスパターン p に対して対応する状態 s_p に p を出力する処理を設定することで，XQuery の質問式としてのパスパターン p を処理が設定されることになる．この処理を形式的に次のように書くことにする．

$$s_p : output(p).$$

さらに，条件式が加わり $p[q]$ となったパスパターンについて考える．これは，p/q となるパスが存在する場合は状態 s_p において p を出力するということであるので，次のように書くことができる．

$$s_p : output(p) \text{ if } v(s_p, p/q),$$
$$s_{p/q} : v(s_p, p/q) := true.$$

親パスおよび子パスに対応する状態に適当な処理を設定することで，算術演算 $+, -, \ldots$，論理演算 $\wedge, \vee, \ldots$，比較演算 $<, >, \ldots$，統計関数 $count$, $average$, min, max などを扱うことができる．

例えば，$p[count(q) > 3]$ という質問式に対しては，次のように処理を設定すればよい．

$$s_p : output(v(s_p, count(p/q)) \text{ if } v(s_p, count(p/q)) > 3,$$
$$s_{p/q} : v(s_p, count(p/q)) := v(s_p, count(p/q)) + 1.$$

for 文も同様に，例えば for \$ v in p return e は，式 e の値をパスパターン p に対応する状態 s_p へと保存し，それを式の値とすることで処理される．

例えば，次の質問式を考える．

```
for $a in /site/*[keyword="banners"]
return $a/text
```

ここで，p を/site/*，q を/site/*/keyword，r を/site/*/text とするとき，これらのパスパターンに対応した状態に対して設定される処理は以下の通りとなる．

$$s_p : output(v(s_p, e)) \text{ if } v(s_p, p[keyword = "banners"]),$$
$$s_q : v(s_p, p[keyword = "banners"]) := true \quad \text{if } s_q = "banners",$$
$$s_r : v(s_p, e) := n_r.$$

3.5 複数 XQuery 処理

ここでは，複数の XQuery 質問式を同時に効率よく処理できる方法を提案する．

前述したように，複数のパスパターンを含む質問式が 1 つの DFA で処理されることをみた．質問式中にパスパターンがいくつ含まれていようとも，各パスパターンの対応する状態に適当な処理を設定し，対象となる XML データに対して DFA をただ 1 つだけ状態遷移することで XQuery の部分族を処理することが可能であった．

さらに，この手法は複数の質問式に対しても容易に拡張することができる．

各パスパターンに対応して設定される処理はそれぞれ独立しており，複数の質問式中に含まれるパスパターンに対応する処理全てを 1 つの DFA に設定することによって複数の質問式を同時に処理することが可能となる．

2 つのパスパターン $p[q]$，$r[s]$ を考える．前節で示したパスパターンが 1 つの場合の処理を拡張して，次のように書くことができる．

$$
\begin{aligned}
s_p &: output(p) \text{ if } v(s_p, p/q),\\
s_{p/q} &: v(s_p, p/q) := true.\\
s_r &: output(r) \text{ if } v(s_r, r/s),\\
s_{r/s} &: v(s_r, r/s) := true.
\end{aligned}
$$

これは，状態 s_p において子パス p/q が存在するならば p を出力し，状態 s_r において子パス r/s が存在するならば r を出力するということである．

このように，前節の処理を拡張し，親パスおよび子パスに対応する状態に適当な処理を設定することで，複数の質問式でも算術演算，論理演算，比較演算，統計関数などを 1 つの DFA で処理することができる．

例えば $p[count(q) > 3]$，$r[min(s) < 1]$ というパスパターンの異なる質問式に対しては，次のように処理を設定すればよい．

$$
\begin{aligned}
s_p &: output(v(s_p, count(p/q))) \text{ if } v(s_p, count(p/q)) > 3\\
s_{p/q} &: v(s_p, count(p/q)) := v(s_p, count(p/q)) + 1.\\
s_r &: output(v(s_r, min(r/s))) \text{ if } v(s_r, min(r/s)) < 1\\
s_{r/s} &: v(s_r, min(r/s))\\
&\quad \text{if } v(s_r, min(r/s)) > data(s)\\
&\quad \text{then } v(s_r, min(r/s)) = data(s).
\end{aligned}
$$

for 文も同様に拡張することで処理できる．例えば，for $\$v$ in p return e と for $\$v$ in s return c を考えた時，式 e，c の値をパスパターン p，s それぞれに対応する状態 s_p，s_s へとそれぞれ保存し，それを式の値とすることで同時に処理できる．例えば，次の 2 つの質問式を考える．

```
for $a in /site/*[keyword="banners"]
return $a/text
for $a in /site/item[type="01"]
return sum($a/price)
```

ここで，p を/site/*，q を/site/*/keyword，r を/site/*/text，s を/site/item，t を/site/item/type，u を/site/item/price とするとき，これらのパスパターンに対応した状態に対して設定される処理は以下の通りとなる．

$$
\begin{aligned}
s_p &: output(v(s_p, e)) \text{ if } v(s_p, p[keyword = \text{"}banners\text{"}]),\\
s_q &: v(s_p, p[keyword = \text{"}banners\text{"}]) := true\\
&\qquad \text{if } s_q = \text{"}banners\text{"},\\
s_r &: v(s_p, e) := n_r.\\
s_s &: output(v(s_s, sum(c))) \text{ if } v(s_s, s[type = \text{"01"}]),\\
s_t &: v(s_s, s[type = \text{"01"}] := true\\
&\qquad \text{if } s_t = \text{"01"},\\
s_u &: v(s_s, sum(c)) := v(s_s, sum(c)) + data(n_u).
\end{aligned}
$$

このように，複数質問式に対しても DFA の各状態に，それぞれの処理を設定することで 1 つの DFA で処理することが可能である．

4. 実装実験

上記の手法を計算機上に実装し実験を行った．実験に用いた環境は OS: RedHat Linux Advanced Server 2.1, CPU: Intel Pentium 4, 2.4 GHz, メモリ: 2.0 GB. 言語は Java（J2SE 1.4.2）を用いた．

まず，XMark ベンチマーク [13] のために提供されている XML 文書生成機である xmlgen により生成された様々なサイズの文書に対して 1 つの質問式について処理を行う．前処理 (preprocess) および本処理 (run) にかかる時間を図 6 に示す．

図より全体の処理時間に対して前処理の処理時間が大きなことが分かる．前処理は，XML データ中に含まれるパスを調べる処理と質問式に対してパスプルーニングを行う処理，パスプルーニング後に得られた基礎パスパターンから DFA を構築する処理からなる．この時，1 つの質問式を処理するのにかかる時間は実行時間に対して非常に小さい．

また，XML データ中に含まれるパスを調べる処理は XML データが決まれば一度だけ行えばよい処理であり，質問式には依存しない．したがって，今後の議論では前処理中に含まれる XML データに対する処理の時間は考慮しない．

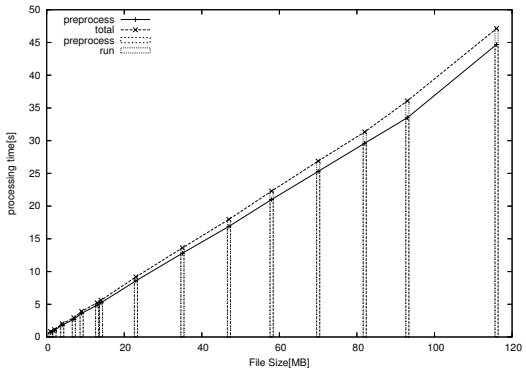

図 6　各処理にかかる時間

次に，XQuery の処理システムとして定評のある Tamino との性能比較を行った結果を示す．実験の対象データは，先ほどと同じく xmlgen によって作成されたサイズの異なる複数のランダム XML 文書である．

Tamino はクライアント・サーバで動作することから，クライアントから質問式を送信した時点からサーバからの返答が得られるまでの時間をサーバでの動作時間を処理時間として計測した．また，比較のために我々の手法も同様にクライアント・

サーバとして実装し，同じ条件で動作時間を計測した．

図 7 にその結果を示す．Tamino は質問式の種類によっては処理が 10 分という制限時間内に終わらないものがあり，また，処理時間が処理対象の XML データのサイズに必ずしも依存していないことが分かる．それに対して，我々の手法の処理時間は図 7 で，ほぼ x 軸に沿って示されているものであり，常に安定して Tamino よりも高速に処理ができていることが分かる．

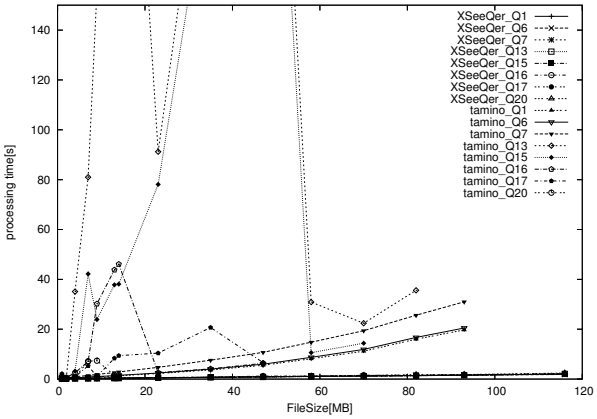

図 7　XMark ベンチマークの比較

もちろん，我々の手法は XQuery の質問式全てを扱えるものではないが，その制限された範囲内においては，市販されている XQuery 処理システムと比較しても，十分に高速であることが確認できた．

5. 複数 XQuery 質問式の同時処理

ここでは，本稿で提案した複数の XQuery 質問式の同時処理を実験によって評価する．

5.1 複数パスパターンの同時処理

XQuery 処理においても，基本となるのはパスパターンの処理速度である．そこで複数パスパターンの処理速度を XPath の処理システムである XMLTK [6], [7] と比較した結果を示す．ここで，計測したのはパスの読み込みから実行までの時間である．実験に用いるパスパターンは，YFilter [16] によって提供されている pathgenerator を用いてランダムに作成した．pathgenerator に与えたパラメータは，*の確率を 1%，//の確率を 10%，同じパスパターンが出現する確率は 20%とした．パスパターンの数を 100,000 まで増加させたときの処理時間を計測した．なお，実験で用いたデータは xmlgen によって作られた 25MB のランダム XML データである．

図 8 に XMLTK との処理時間の比較結果を示す．

また，XMLTK では複数のパスパターンは OR パターンとして扱われており，例えば，//from と/site/regions/from のように表現が異なるが同一のものを指すパスパターンに対して 1 度の出力しか行わない．それに対して，我々の手法ではそれぞれが独立した質問式であるため，例え同じ質問式が与えられても，それぞれの質問式に対する出力を行うといった相違点がある．

また，処理時間を質問式の個数で割った見かけ上の 1 つの処理にかかる時間を図 9 に示す．図からも分かるように，質問式

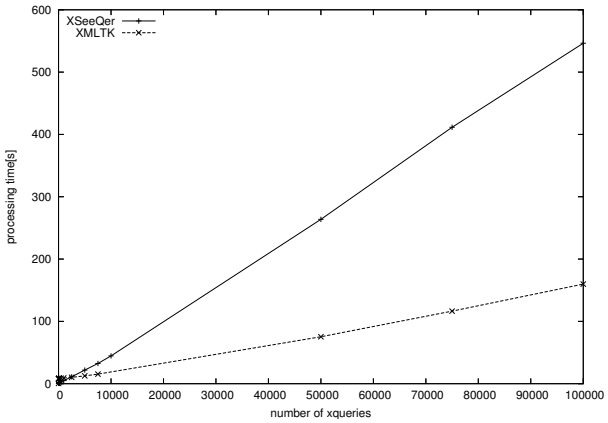

図 8　XMLTK との比較

の増加に対するオーバーヘッドはなく，ほぼ一定した性能で処理が行えている．

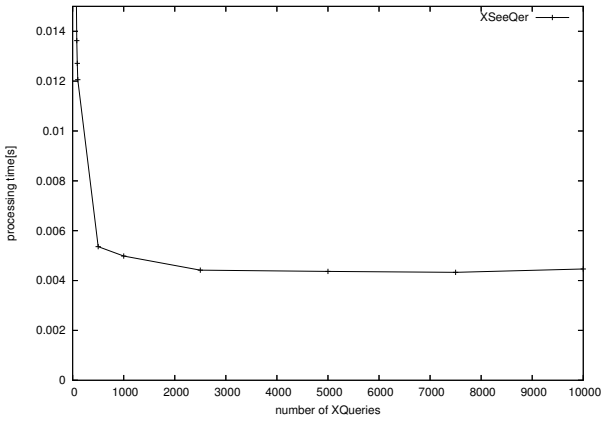

図 9　1 つの XQuery を処理するのにかかる時間

5.2 複数の質問式の同時処理

count, sum，average，max，min それぞれを用いた質問式を複数同時に処理を行った実験結果を示す．また，それらの質問式を混合したもの (multi) および複数の for 文に対する質問処理についても実験を行った．

実験に用いたランダム XML データやパスパターンの生成方法は，先の実験と同じである．

図 10 に実験結果を示す．全ての処理において，処理時間は線形に増加しており，質問式に依らず安定した性能が得られた．

for 文の処理を行う質問式は，10,000 個の式以上は今回の実験環境では処理することができなかった．今回，実験を行った for 文は for $p in $p return $p/$q といった形のものであるが，return で指定された$p/$q に相当する出力を記憶するための領域が必要なことが，他の例ほど多くの質問式では実験を行うことができなかったことの原因である．

6. ま と め

本稿では，複数 XQuery 質問式を 1 つの決定性有限オートマトンを用いることで一度に処理する手法について述べた．質問式中に表れるパスパターンの*，//をパスプルーニングとい

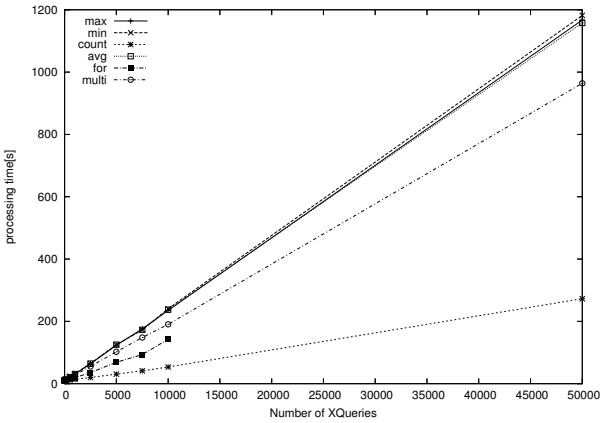

図 10　大量の質問式の処理

う手法を用いることで実際のタグ名と/で置き換え基礎パスにする．得られた基礎パスの共通部分を束ねることにより１つの DFA を構築し各状態にそれぞれの処理を適切に設定することで複数 XQuery 質問式を一度に処理することを可能にした．

実際に提案手法を実装し，実際の XML データを用いた実験結果を示した．実験により，本稿で提案した手法は既存の手法と比較しても十分に高速に動作し，また大規模な対象データおよび大量の質問式に対する規模耐性でも優れていることが確認された．

本稿の処理手法は，質問式をそれぞれ独立したものとして扱うため，同じ質問式が複数あった場合でも１つ１つの質問式に対して処理を行う．この手法では，質問式に比例した処理時間がかかるので，今後このような場合は，それらを束ねて一度だけ処理をおこなうようにする．これにより一層の高速化が期待できる．

文　　献

[1] Aho, A. V. and Corasick, M. J.: Efficient String Matching: an aid to Bibliographic Search, *Commun. ACM*, Vol. 18, No. 6, pp. 333–340 (1975).

[2] Diao, Y. and Franklin, M.: Query Processing for High-Volume XML Message Brokering, *Proc. of the 29th VLDB Conference* (2003).

[3] Florescu, D., Hillery, C., Kossmann, D., Lucas, P., Riccardi, F., Westmann, T., Carey, M. J. and Sundararajan, A.: The BEA Streaming XQuery Processor, *The VLDB Journal The International Journal on Very Large Data Bases*, Vol. 13, No. 3, pp. 294–315 (2004).

[4] Goldman, R. Widom, J.:Enabling Query Formulation and Optimization in Semistructured Databases, *VLDB'97: Proceedings of the 23rd International Conference on Very Large Data Bases, Morgan Kaufmann Publishers Inc.*, pp.436-445(1997).

[5] Gottlob, G., Koch, C. and Pichler, R.: Efficient Algorithms for Processing XPath Queries, *VLDB 2002*, pp. 95–106 (2002).

[6] Green, T. J., Miklau, G., Onizuka, M. and Suciu, D.: Processing XML Streams with Deterministic Automata, *ICDT '03: Proceedings of the 9th International Conference on Database Theory*, Vol. LNCS 2572, pp. 173–189 (2003).

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

[8] Gupta, A. K. and Suciu, D.: Stream Processing of XPath Queries with Predicates, *Proc. SIGMOD*, pp. 431–442 (2003).

[9] Josifovski, V., Fontoura, M. and Barta, A.: Querying XML streams, *The VLDB Journal*, Vol. 14, pp. 197–210 (2005).

[10] Koch, C., Scherzinger, S., Schweikardt, N. and Stegmaier, B.: FluXQuery: An Optimizing XQuery Processor for Streaming XML Data, *Proceedings of the 30th VLDB Conference*, pp. 1309–1312 (2004).

[11] Ludäscher, B., Mukhopadhyay, P. and Papakonstantinou, Y.: A Transducer-Based XML Query Processor, *Proceedings of the 28th VLDB Conference* (2002).

[12] McHugh, J., Widom, J.:Compile-Time Path Expansion in Lore, *Proc of the Workshop on Query Processing for Semistructured Data and Non-Standard Data Formats* , 1999.

[13] Schmidt, A., Waas, F., Kersten, M.L, Carey, M.j., Manolescu, I. and Busse, R.:XMark:A Benchmark for XML Data Management, *Proc.of the 28th VLDB conference,* pp.974-985(2002).

[14] Takeda, M., Miyamoto, S., Kida, T., Shinohara, A., Fukamachi, S., Shinohara, T. and Arikawa, S.: Processing Text Files as Is: Pattern Matching Over Compressed Texts, Multi-Byte Character Texts, and Semi-Structured Texts, *SPIRE 2002*, Vol. LNCS 2476, pp. 170–186 (2002).

[15] W3C: XQuery 1.0: An XML Query Language, W3C Working Draft (2004).

[16] YFilter: Filtering and Transformation for High-Volume XML Message Brokering.
`http://yfilter.cs.berkeley.edu/code_release.htm`

[17] Zhang, X., Mulchandani, M., Christ, S., Murphy, B. and Rundensteiner, E. A.: Rainbow: Mapping-Driven XQuery Processing System, *Proceedings of the ACM SIGMOD International Conference on Management of Data* (Franklin, M., Moon, B. and Ailamaki, A., eds.), pp. 614–614 (2002).

[18] 石野　明，竹田正幸：パスプルーニングと決定性有限オートマトンを用いた　　ストリーム指向の XQuery 処理，*DBWeb2005*(2005).

[19] 竹田正幸，石野　明，辻　寿嗣，　宮本：ストリーム指向の高速 XML データ処理技法について，データベースと Web 情報システムに関するシンポジウム (DBWeb2003) 予稿集 (2003).

[20] 喜田拓也，宮本　哲，竹田正幸：文字列照合技術に基づく XML データ処理，情報科学技術フォーラム 2002 (FIT2002)，pp. 55–56 (2002).

[21] 森川裕章，浅井達哉，有村博紀：データストリーム処理のための効率良い XPath 問合せ機構，情報処理学会研究報告 DBS-131, Vol. 71, pp. 211–218 (2003).