圧縮 XML 文書に対する XQuery 式の直接評価手法における上方向探索時のキャッシュの効果の分析

宮本 健矢　　水谷 響　　石原 靖哲

南山大学 理工学部

概要　圧縮 XML 文書に対する XQuery 式の直接評価手法が小椋らによって提案・実装されている．この手法では，問合せ処理時間短縮のため，XML 文書を上方向に探索する際に問合せ中間結果をキャッシュする工夫がなされている．しかし，その効果は実験的には確認されていなかった．本稿では，小椋らとは異なるキャッシュ手法を新たに提案した上で，小椋らの手法や素朴に上方向を探索する手法に対し問合せ処理時間を実験的に比較することで，上方向探索時のキャッシュの効果を分析する．

1　はじめに

マークアップ言語 XML は，Web ページの記述に用いる XHTML や画像データの記述に用いる SVG など，世の中で幅広く利用されている．XML は自己記述性をもつため，プラットフォームの壁を越えたデータ交換が容易に行えるという長所がある．一方で，タグを利用して記述するため，ファイルのサイズが大きくなりデータの交換量が膨大になってしまうという問題がある．

このような背景のもと，圧縮された XML 文書に対する問合せを，圧縮を展開せずに直接評価する手法の研究が盛んに行われている．その中で小椋らの手法 [1] [2] は，特殊な処理エンジンを必要とせず，XML 文書を展開して問合せ処理する場合と比べてメモリ必要量を約 4 分の 1 に削減できる場合があると報告されている．その一方，問合せ処理時間が大幅に大きくなるという問題があるため，改善策として小椋らは，XML 文書を上方向に探索する際に問合せの中間結果をキャッシュしておくことで XML 文書内の探索にかかる時間を短縮する手法を提案・実装している．しかし，その改善策の効果は実験的には確認されていない．

本稿では，上方向探索時のキャッシュの効果について実験的に分析した結果について述べる．対象は，素朴に上方向を探索する手法と，小椋らが提案した手法，そして我々が提案した，小椋らとは異なるアルゴリズムで中間結果をキャッシュする手法である．XML 文書の形状を変化させつつ，問合せ処理時間にどのような影響を与えるのかについて実験的に分析した．その結果，小椋らの手法と我々の手法は，親へポインタを移動する際に兄弟ノードが多く存在する場合は素朴に上方向を探索する手法と比べ，問合せ処理時間の短縮につながることがわかった．一方で，兄弟ノードが存在しない場合は素朴に上方向を探索する手法と比べ，問合せ処理時間が長くなってしまうことがわかった．

2　関連研究

圧縮 XML 文書に対する問合せを直接評価可能な手法は数多く研究されているが，それらのほとんどは，直接評価するために専用の問合せ言語で問合せを記述することを前提としている．それに対し，一般的な問合せ言語である XQuery で記述された問合せを扱える手法として，TraCX [3] や XQuery.Z 式 [1] [2] がある．

TraCX は，文書ファイルを独自の圧縮方法に則って圧縮することで文書を圧縮したまま問合せを評価することが可能な手法である．しかし，この手法では，問合せ処理のために独自の処理エン

表 1: XPath の軸

軸名	結果
self	カレントノード
child	全子供
parent	親
descendant	全子孫（子供や孫など）
descendant-or-self	全子孫とカレントノード
ancestor	全祖先（親や祖先など）
ancestor-or-self	全祖先とカレントノード
following-sibling	カレントノード後の兄弟
preceding-sibling	カレントノード前の兄弟
following	カレントノードの終了タグの後にある文書全体
preceding	カレントノードの開始タグの前にある文書全体

ジンが必要である．それに対して，XQuery.Z 式が優れている点は，通常の XQuery 処理エンジンをそのまま用いて圧縮 XML 文書に対して問合せを評価することが可能であるという点である．しかし，XML 文書の展開をシミュレートしながら問合せ処理を行うため，XML 文書を展開して問合せ処理する場合と比べて，トータルの処理時間がしばしば大きくなるという欠点がある．

3 諸定義

3.1 XQuery

本稿では，XML に対する問合せ言語として XQuery を使用する．本稿で対象とする XQuery のクラスを以下に示す．

$$
\begin{aligned}
e &::= () \mid var \mid \mathrm{doc}(arg) \mid e/\alpha :: \tau \mid (e, \ldots, e) \mid \text{for } var \text{ in } e \text{ return } e \\
\alpha &::= \text{self} \mid \text{child} \mid \text{parent} \mid \text{descendant} \mid \text{descendant-or-self} \mid \text{ancestor} \mid \text{ancestor-or-self} \mid \\
&\quad \text{following-sibling} \mid \text{preceding-sibling} \mid \text{following} \mid \text{preceding} \\
\tau &::= label \mid *
\end{aligned}
$$

XQuery の一部は，XPath と呼ばれる言語から構成されている．XPath は XML 文書を特定の方向に探索する機能をもつ．探索方向を指定する構文要素を軸と呼ぶ．上の α が軸の定義にあたる．それぞれの軸の意味を表 1 に示す．

XML 文書では根ノードの上にドキュメントノードと呼ばれるノードが存在する．`doc(`ファイル名`)` により，ファイル名に対応する XML 文書のドキュメントノードを参照できる．そして，以下の問合せ

```
for $v in doc(ファイル名) return $v/child::生徒データ/child::生徒/child::名前
```

は，ドキュメントノードの子供でノード名が “生徒データ” であり，その子供でノード名が “生徒” であり，さらにその子供でノード名が “名前” であるようなノードを出力する．

3.2 TreeRePair

本稿では XML の圧縮ツールとして高い圧縮効率をもつことで知られる TreeRePair [4] を利用する．TreeRePair は，XML 文書を SL 文脈自由木文法で表現することにより圧縮を行う．ここで SL（straight-line）文脈自由木文法とは，各非終端記号に対して適用できる規則がちょうど一つ存在し，かつ，どの非終端記号も自分自身を導出できないような文脈自由木文法である．この制約により，SL 文脈自由木文法が導出する木言語の要素数はちょうど 1 になる．

TreeRePair では以下の流れで XML 文書を圧縮する．

1. XML 文書で出現するテキストノードを全て削除する．
2. fcns 符号化により 2 分木に変換する．
3. 複数出現するパターンを SL 文脈自由木文法で使われる非終端記号に置き換える．

ここで fcns（first-child/next-sibling）符号化とは，子の数に制限のない木を，その情報を失うことなく 2 分木に変換する手法である．具体的には以下の規則に基づいて変換する．

- v が子を持つとき，v の最も左の子を v の左の子にする．そうでないとき，ラベル#をもつ新たなノードを v の左の子にする．
- v が右に兄弟を持つとき，v のすぐ右隣の兄弟を v の右の子にする．そうでないとき，ラベル#をもつ新たなノードを v の右の子にする．

例えば，図 1 の構造をもつ XML 文書を fcns 符号化すると図 2 のようになる．さらに，図 2 において共通の構造を非終端記号に置き換えることで得られる SL 文脈自由木文法（の導出規則）の例を，fcns 符号化を復号した形で図 3 に示す．

3.3 XQuery.Z

XQuery.Z とは，小椋らが提案した，圧縮 XML 文書に対して直接評価可能な XQuery の部分言語である．文献 [2] では，非圧縮文書に対する XQuery 問合せを XQuery.Z 問合せに変換する手法も与えている．問合せ変換の流れは，まず，XQuery 式を字句解析して，軸をすべて抽出する．次に，抽出した各軸を，圧縮文書上で同じ振舞いをするように定義された関数に置換する．そうすることで，非圧縮 XML 文書に対する XQuery 式を，圧縮 XML 文書に対する XQuery.Z 式に変換することができる．この XQuery.Z 式と圧縮 XML 文書を用いて，通常の XQuery 処理エンジンで問合せ結果を得ることができる．

圧縮 XML 文書に対する走査（ポインタ移動）においては，文書が fcns 符号化されていることを考慮する必要がある．中でも，親方向に走査する軸は，parent 軸，ancestor 軸，ancestor-or-self 軸，preceding-sibling 軸，preceding 軸であり，それらの走査に伴うポインタ移動には以下の 3 つの場合がある．

- fcns 符号化前の文書上の兄ノードまで移動する場合（図 4）．
  fcns 符号化前の文書上の兄ノードまで移動するには，fcns 符号化後の文書上で親ノードへ移動する．移動後のノードの右の子がポインタ移動前のノードであれば，移動後のノードが fcns 符号化前の文書上の兄ノードである．

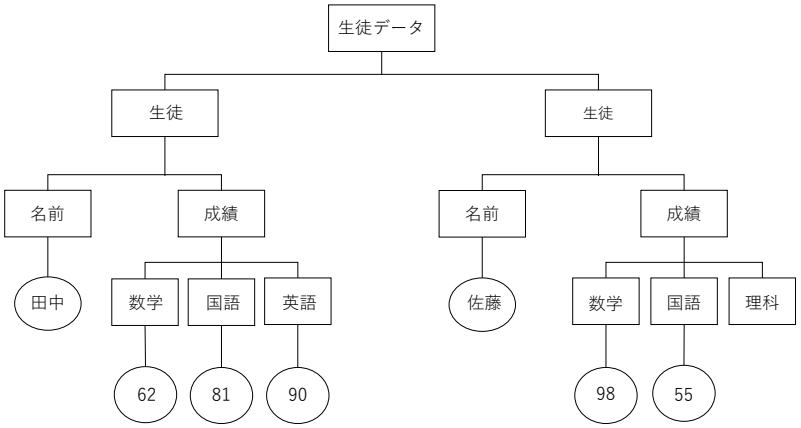

図 1: XML 文書の木構造

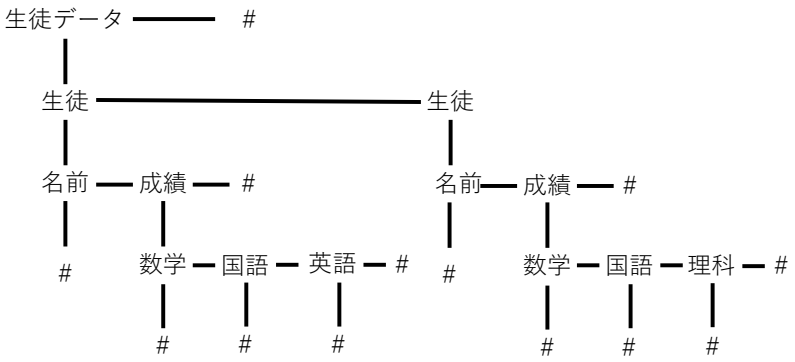

図 2: fcns 符号化

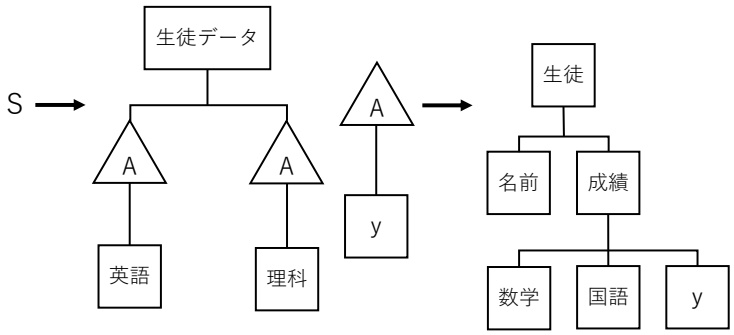

図 3: SL 文脈自由木文法による圧縮

- fcns 符号化前の文書上の親ノードまで移動する場合（図 5）.
  fcns 符号化前の文書上の親ノードまで移動するには，fcns 符号化前の兄ノードが存在する限り兄ノードへ移動し続ける．そして，兄ノードが存在しないノードから一度親ノードへ移動した先のノードが，fcns 符号化前の文書上の親ノードである.
- fcns 符号化前の文書上の先祖ノードまで移動する場合（図 6）.
  fcns 符号化前の文書上の先祖のノードまで移動するには，fcns 符号化前の文書上の親ノードへの移動を可能な限り続ける．親ノードにあたるノードがドキュメントノードであれば処理を終了する.

4　先祖ノードの情報をキャッシュするアルゴリズム

前節で述べたように，fcns 符号化前の文書で上方向のポインタ移動をする際，fncs 符号化後の文書上ではすべての兄ノードをたどる必要がある．そこで小椋らは，上方向の探索する際に問合せの中間結果をキャッシュしておき，キャッシュが利用できる場合はそれを利用することで，すべての兄ノードをたどることを回避する手法を提案・実装している.

まず，小椋らの手法における parent 軸の処理を説明する．ノード集合 V に属する各ノードの親からなる集合 U を求める場面を考える．文書順序で V の各ノード v に対して以下の処理を行う.

1. $x = v$ とおく.
2. fcns 符号化後の文書において x の兄 y が存在するかを調べる.
   (a) y が存在しない場合，x の親 u について $u \in U$ とし，さらにペア (v, u) をキャッシュして，v に対する処理を終了する.
   (b) y が存在する場合，ある u に対してペア (y, u) がすでにキャッシュされているかを確認する.
      i. キャッシュされている場合，$u \in U$ とし，さらにキャッシュ内容を (v, u) に置き換えて，v に対する処理を終了する.
      ii. キャッシュされていない場合，新たに $x = y$ とおいて 2 の処理を繰り返す.

V 中のノードを文書順序で処理するため，キャッシュするのは直前に調べたノードとその親のペアだけで十分である.

小椋らの手法における ancestor 軸や ancestor-or-self 軸の処理は，上で述べた親ノード集合を求める処理を，ドキュメントノードに到達するまで再帰的に繰り返すものである．したがってキャッシュされる情報は処理対象のノードの親の情報のみである.

次に，我々が提案する，先祖ノードの情報をキャッシュする手法を述べる．ノード集合 V に属する各ノードの先祖からなる集合 U を求める場面を考える．文書順序で V の各ノード v に対して以下の処理を行う.

1. $x = v$ とおく.
2. fcns 符号化後の文書において x の兄 y が存在するかを調べる.
   (a) y が存在しない場合，x の各先祖 u について $u \in U$ とし，さらに各ペア (v, u) をキャッシュして，v に対する処理を終了する.

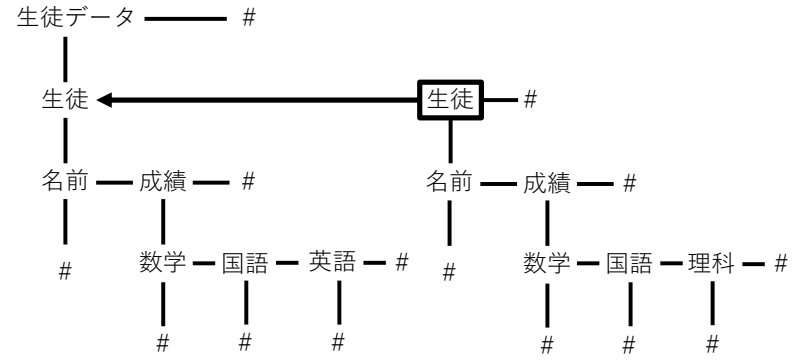

図 4: 兄ノードにポインタを移動

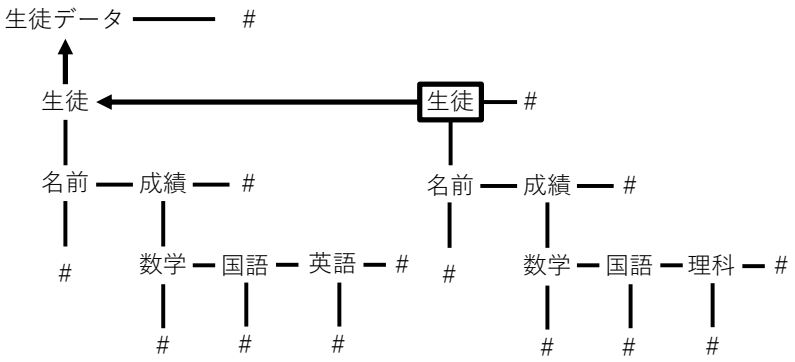

図 5: 親ノードにポインタを移動

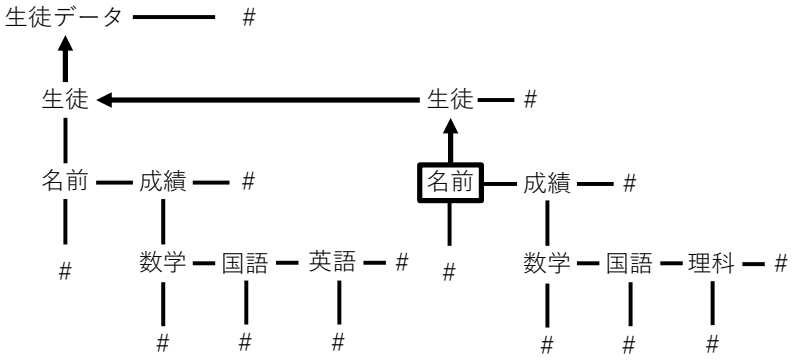

図 6: 先祖ノードにポインタを移動

表 2: 実験に用いるパス式

	パス式
1	/descendant::dataset/ancestor::datasets
2	/descendant::reference/ancestor::datasets
3	/descendant::reference/ancestor::dataset

(b) y が存在する場合，ある u に対してペア (y, u) がすでにキャッシュされているかを確認する．

i. キャッシュされている場合，そのような各 u について $u \in U$ とし，さらにキャッシュ内容を (v, u) に置き換えて，v に対する処理を終了する．

ii. キャッシュされていない場合，新たに $x = y$ とおいて 2 の処理を繰り返す．

5　XML 文書の形状に対するキャッシュ効果の分析

本節では，XML 文書の形状に対するキャッシュの効果を実験的に分析した結果について述べる．以降では，キャッシュをせずに素朴に上方向をたどる手法を old と呼び，小椋らによるキャッシュ手法を new と呼ぶ．さらに我々の提案したキャッシュ手法を our と呼ぶ．

実験では，地球から見た恒星の座標の観測などについての文書である Nasa.xml [5] を実験の目的にあった形に変更した文書を用いる．Nasa.xml の圧縮前のファイルサイズは 553,470KB であり，圧縮後のファイルサイズは 61,044KB である．また，Nasa.xml の文書構造は深さ 0 のノード（根ノード）datasets の子に複数の dataset があり，その子に reference がある構造となっている．

5.1　実験環境

実験に用いた PC は Intel Core i7 3.2GHz，16GB メモリであり，OS は Windows 10 Pro 64 ビットである．VirtualBox 6.0.8 に Ubuntu 18.04.2 LTS をインストールし，その上で実験した．XQuery 処理に用いたのは Saxon-HE 9-8-0-3j（Java のバージョンは 1.9.0-181）である．

5.2　実験内容と結果

実験で使用するパス式を表 2 に示す．実験結果は全て 5 回試行した平均の値である．

5.2.1　深さ 1 から深さ 0 へたどる問合せに対するキャッシュの効果の比較

問合せとしてパス式 1 を用いる．このパス式は深さ 1 のノード dataset から深さ 0 のノード datasets をたどる．深さ 1 のノードの兄弟が多いほどキャッシュの効果が発揮され，new の方が old より問合せの実行時間が短くなると予想されるが，実際はどの様な変化が見られるのか実験する．用いる文書は，1 つの dataset を 10 個から 1500 個までつなぎ合わせた文書をそれぞれ作ったものを使用する．

実験結果を表 3 と図 7 に示す．予想した通り，dataset の個数が 100 個付近までは，new，our，old の問合せ実行時間は近い値であるものの，dataset の個数が増えていけば増えていくほど new と our の方が old より問合せ実行時間は短くなった．

5.2.2 深さ 2 から深さ 0 へたどる問合せに対するキャッシュの効果の比較

問合せとしてパス式 2 を用いる．このパス式は深さ 2 のノード reference から深さ 0 のノード datasets をたどる．

まず，深さ 1 の各ノードがちょうど一つの子をもつ場合に，深さ 1 のノードの個数がキャッシュの効果に与える影響を調べる．用いる文書は，深さ 0 のノード datasets の子に dataset を 10 個から 1500 個持ち，その子に 1 つの reference を持つ文書である．

実験結果を表 4 と図 8 に示す．前節の実験結果と同様になると予想されたが，dataset の個数を増やしていっても new と old の問合せ実行時間には差がほとんど生じなかった．しかし，our の問い合わせ実行時間は，new と old より問合せ実行時間が長くなった．

次に，深さ 1 のノードの個数は固定して，子である深さ 2 のノードの個数がキャッシュの効果に与える影響を調べる．深さ 0 のノード datasets の子に dataset を 500 個持ち，子の reference を 1 個から 20 個持つ文書を使う．

実験結果を表 5 と図 9 に示す．深さ 2 のノードである reference の個数を増やしていくと，new と our の方が old と比べ問合せ実行時間は短くなるという結果が得られた．

5.2.3 深さ 2 から深さ 1 へたどる問合せに対するキャッシュの効果の比較

問合せとしてはパス式 3 を用いる．このパス式は深さ 2 のノード reference から深さ 1 のノード dataset をたどる．

前節の前半の実験の結果から，new が深さ 2 から深さ 1 へのポインタ移動で時間がかかっていると予想されため，深さ 1 のノードに 1 つ子を持たせ，深さ 1 のノードの兄弟を増やしていった時の new と old，our の実行時間にどの様な変化が見られるのか実験する．使用する文書は，前節の前半の実験と同じである．

実験結果を表 6 と図 10 に示す．予想した通り，dataset の個数を増やしていくと new と our の方が old の問合せ実行時間より長くなっていることが分かる．また，new と our を比べると our の方が問合せ実行時間が長くなった．

5.3 考察

new と our は，先祖にポインタ移動する際に親ノードをキャッシュすることで，すべての兄ノードをたどることを回避している．一方，old では親ノードへポインタ移動する際に素朴に兄ノードをたどっている．そのため fcns 符号化した文書上で親ノードへポインタ移動する際に，多くの兄ノードをたどる必要があるケースでは old は new と our より問合せの実行時間がかかっている．

3 番目の実験で，new と our の方が old より実行時間がかかってしまっているのは，親ノードへポインタ移動する際に兄弟ノードが存在していないが，親ノードをキャッシュしてすべての兄ノードをたどることを回避しようとする処理が余分に行われているからであると考える．

表 3: 深さ 1 から深さ 0 へたどる問合せに対するキャッシュの効果の比較の実験結果

	dataset の個数	new の実行時間 (ms)	old の実行時間 (ms)	our の実行時間 (ms)
1	10	598.7	575.0	575.9
2	20	889.7	908.4	788.4
3	30	1,112.0	1,204.7	1,142.3
4	40	1,504.7	1,388.7	1,515.7
5	50	1,504.4	1,635.4	1,547.5
6	60	1,698.0	1,740.7	1,665.6
7	70	1,798.0	1,893.7	1,789.9
8	80	2,009.0	1,934.6	1,972.8
9	90	2,061.2	2,151.0	2,031.1
10	100	2,205.8	2,151.0	2,192.9
11	300	4,751.0	5,354.2	4,810.5
12	500	6,268.6	8,237.0	6,089.8
13	1,000	11,063.2	19,121.8	11,475.3
14	1,500	16,671.0	31,264.4	17,225.8

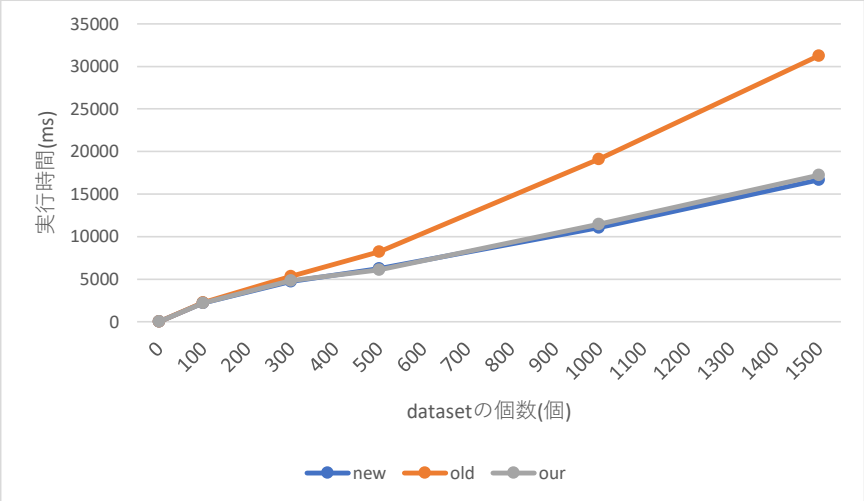

図 7: 深さ 1 から深さ 0 へたどる問合せに対するキャッシュの効果の比較の実験結果

表 4: 深さ 2 から深さ 0 へたどる問合せに対するキャッシュの効果の比較の実験結果（前半）

	dataset の個数	new の実行時間 (ms)	old の実行時間 (ms)	our の実行時間 (ms)
1	10	578.0	607.7	598.0
2	20	889.7	908.4	942.1
3	30	1,129.7	1,135.0	1,171.5
4	40	1,398.7	1,330.0	1,306.9
5	50	1,564.4	1,602.4	1,596.1
6	60	1,869.0	1,812.0	1,973.3
7	70	2,015.2	1,858.6	1,993.8
8	80	1,991.0	2,071.0	2,062.8
9	90	2,187.0	2,247.6	2,359.6
10	100	2,446.8	2,387.0	2,482.8
11	300	6,036.0	5,433.8	6,545.7
12	500	8,641.6	8,361.5	10,687.8
13	1,000	19,672.4	19,566.2	27,654.3
14	1,500	31,907.2	32,812.2	46,737.0

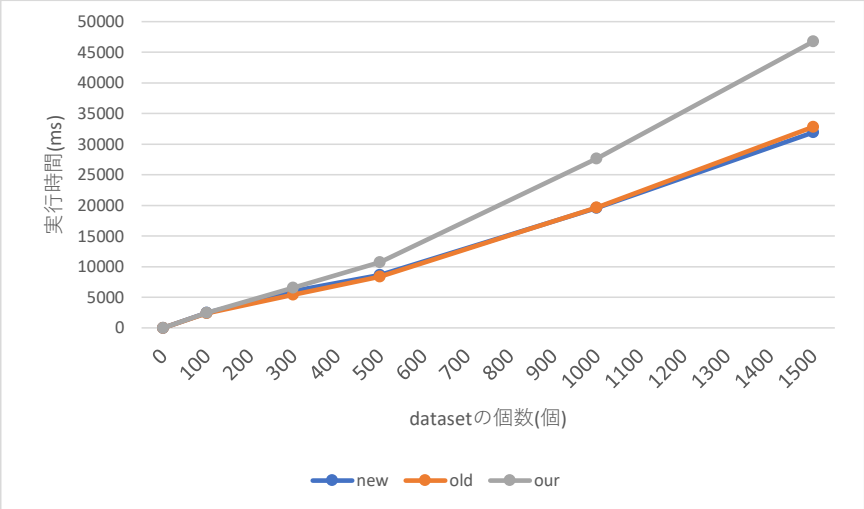

図 8: 深さ 2 から深さ 0 へたどる問合せに対するキャッシュの効果の比較の実験結果（前半）

表 5: 深さ 2 から深さ 0 へたどる問合せに対するキャッシュの効果の比較の実験結果（後半）

	reference の個数	new の実行時間 (ms)	old の実行時間 (ms)	our の実行時間 (ms)
1	1	5,923.0	6,245.0	7,235.0
2	2	6,515.7	8,717.7	8,128.7
3	3	7,737.0	10,679.0	9,370.5
4	5	13,055	20,113.0	15,384.2
5	10	33,468.3	46,572.0	34,653.7
6	20	198,574.3	234,475.3	206,600.7

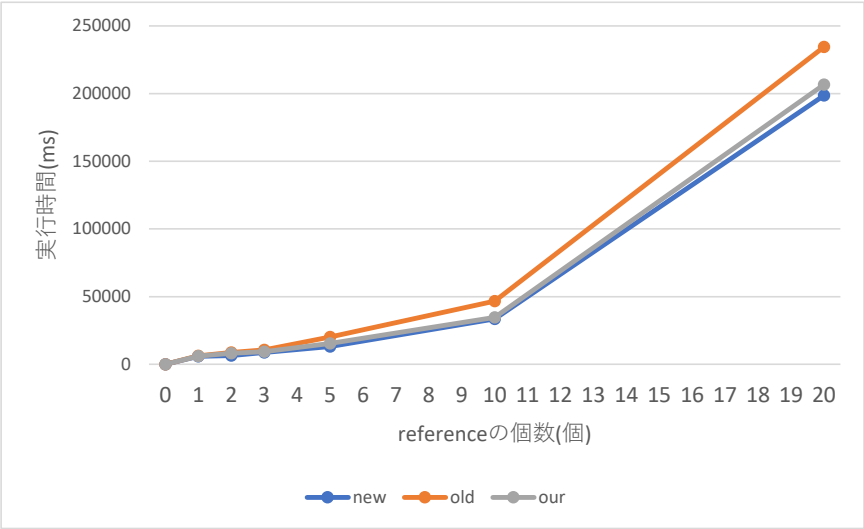

図 9: 深さ 2 から深さ 0 へたどる問合せに対するキャッシュの効果の比較の実験結果（後半）

表 6: 深さ 2 から深さ 1 へたどる問合せに対するキャッシュの効果の比較の実験結果

	dataset の個数	new の実行時間 (ms)	old の実行時間 (ms)	our の実行時間 (ms)
1	10	630.0	597.0	659.6
2	20	913.7	893.0	889.3
3	30	1,161.4	1,143.6	1,246.9
4	40	1,497.4	1,394.0	1,638.0
5	50	1,631.0	1,550.7	1,775.6
6	60	1,993.0	1,841.0	1,899.7
7	70	2,180.0	1,925.7	2,299.8
8	80	2,092.0	1,989.7	2,303.8
9	90	2,322.0	2,178.7	2,527.0
10	100	2,462.4	2,275.0	2,625.8
11	300	6,621.0	5,113.7	6,931.8
12	500	9,843.4	7,680.0	12,087.8
13	1,000	24,203.7	16,143.7	32,285.9
14	1,500	43,231.0	25,822.4	56,867.5

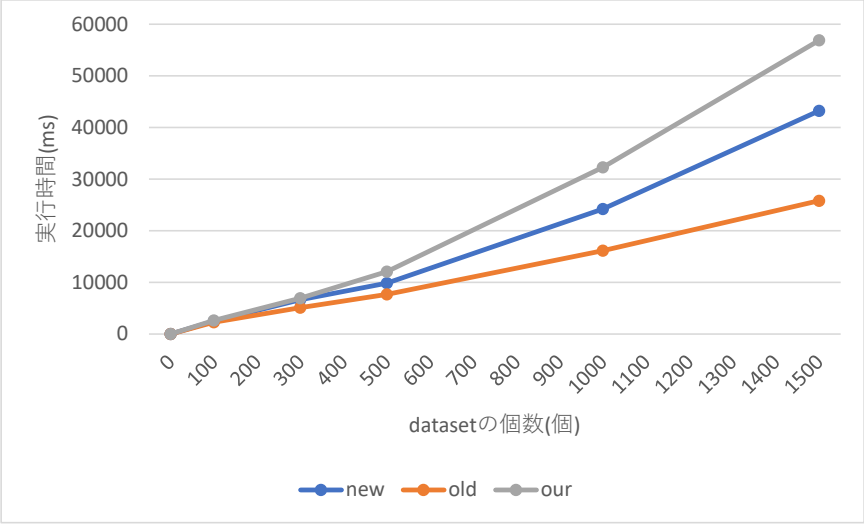

図 10: 深さ 2 から深さ 1 へたどる問合せに対するキャッシュの効果の比較の実験結果

2 番目の前半の実験で new と old の実行時間に差がほとんど生じなかったのは，new で親ノードをキャッシュすることでポインタ移動を回避し，実行時間が短くなっている部分と，親ノードをキャッシュしているが故に実行時間が長くなっている部分があるためだと考えられる．

6 まとめ

本稿では，圧縮 XML 文書に対する XQuery 問合せの直接評価手法に対して小椋や我々が提案した，中間結果をキャッシュしておくことで上方向の探索を短縮する手法について，比較実験を通してキャッシュの効果について分析した．今後の課題は，文書のスキーマ情報を利用し，兄弟ノードが 1 人しかいないなら素朴に上方向をたどる手法を用い，複数いる可能性があるならキャッシュする手法を用いる，という切り替えを自動に行うプログラムを作ることが挙げられる．

謝辞 故青山幹雄教授から生前に賜ったご厚情に深く感謝するとともに，慎んでご冥福をお祈りします．本研究は 2019 年度南山大学パッヘ研究奨励金 I-A-2 の助成を受けたものです．

参考文献

[1] 小椋寿希也, 石原靖哲, 藤原融. XQuery 問合せを圧縮 XML 文書上で評価するための変換手法の開発. 電子情報通信学会技術研究報告, SS2017-23, pp. 13–18, 2017.

[2] 小椋寿希也, 石原靖哲, 藤原融. 圧縮 XML 文書に対する XQuery 問合せ評価の効率化. 電子情報通信学会技術研究報告, SS2018-43, pp. 97–102, 2019.

[3] Stefan Böttcher, Rita Hartel, and Sebastian Stey. TraCX: Transformation of compressed XML. In *Proceedings of the 28th British National Conference on Advances in Databases*, pp. 182–193, 2011.

[4] Markus Lohrey, Sebastian Maneth, and Roy Mennicke. XML tree structure compression using RePair. *Infomation Systems*, Vol. 38, No. 8, pp. 1150–1167, 2013.

[5] XMLCompBench. http://xmlcompbench.sourceforge.net.