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

2016SC051 宮本健矢 2016SC054 水谷響
指導教員：石原靖哲

1 はじめに

マークアップ言語 XML は XHTML を用いた Web サイトや SVG を用いた画像データなど現在世の中で広く利用されている．XML は，自己記述性を持つため，プラットフォームの壁を越えてデータ交換などといった処理を行うことができる．しかし，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．

そのため現在では，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．その一方，問合せ時間が大幅に大きくなるという問題があるため，その改善策として，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 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確認されてはいない．

本研究では，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する．対象は，素朴に祖先をたどる手法と，小椋が実装した手法，そして我々が実装した，小椋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．まず最初に，圧縮 XML 文書を固定し，いくつかの問合せについて実行時間とメモリ必要量を比較した．その結果，我々が実装した手法は小椋が実装した手法や素朴に祖先を辿る手法と比べ問合せの実行時間が長くなった．しかし，メモリ必要量は，素朴に先祖を辿る手法と同じ値となり，小椋の実装した手法と比べ小さくなるという結果を得た．次に，XML 文書の形状を変化させつつ，小椋らと我々が提案・実装した改善策が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，小椋が実装した手法と我々が実装した手法は，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祖先ノードを辿る手法と比べ，問合せ実行時間の短縮につながることが分かった．一方で兄弟ノードが存在しない場合は素朴に祖先ノードを辿る手法と比べ，問合せ実行時間が長くなってしまうことが分かった．また我々が実装した手法は小椋が実装した手法と比べ問合せ実行時間が長くなるという結果を得た．

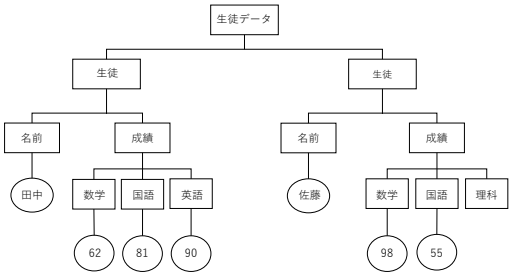

図 1 XML 文書の木構造

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，問合せを行うためには独自の処理エンジンが必要となってしまう．それに対して，XQuery.Z 式が優れている点は，XQuery の処理エンジンをそのまま用い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 文書を展開した場合と比べて問合せの処理時間がかかってしまうという欠点がある．

3 諸定義

本節では，TreeRePair，XQuery，XQuery.Z について説明する．

3.1 XQuery

本研究では，XML に対する問合せ言語として，XQuery を使用する．XQuery とは，XML 文書に対する関数型問合せ言語の一つであり，リレーショナルデータベースに対する SQL のように XML 文書に対して，さまざまな問合せを行うことができる言語である．

XQuery の一部は XPath と呼ばれる XML に準拠した文書の特定の位置を指定する言語からなっている．

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．問合せの`$v in doc(ファイル名)` で変数`$v` にドキュメントノードを格納する．図 1 に対して `for $v in doc(ファイル名) return $v/child::生徒データ/child::生徒/child::名前`という問合せを行

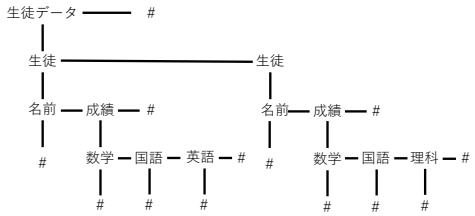

図 2 fcns 符号化

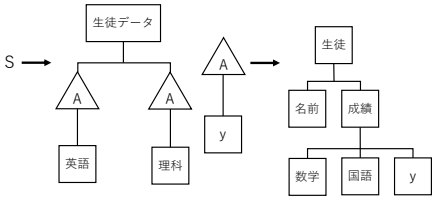

図 3 SL 文脈自由木文法による圧縮

うと，ドキュメントノードの全子供の内，ノード名が “生徒データ”，その全子供の内ノード名が “生徒”，その子供のノード名が “名前” のノードを出力する．

3.2 TreeRePair

本研究では XML の圧縮ツールとして高い圧縮効率を持つことで知られる TreeRePair [4] を利用する．図 1 を TreeRePair を用いて圧縮する流れは以下のとおりである．

1. XML 文書で出現するテキストノードをすべて削除する．
2. first-child next-sibling(fcns) 符号化により 2 分木に変換する (図 2)．
3. XML 文書を SL 文脈自由木文法に変換することで圧縮を行う (図 3)．

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 [1] [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，XML 文書の走査をする部分をすべて抽出する．次に，抽出した各部分を圧縮文書走査用の関数に置換する．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，XQuery 処理エンジンで問合せ結果を得ることができる．

また，圧縮 XML ファイルにおいてポインタ移動は，fcns 符号化された文書上で行われる．fcns 符号化された文書上では，図 2 のように 2 分木となっているため親方向へのポインタ移動をする際，すべての兄ノードを辿る必要がある．そこで小椋は，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辿ることを回避する手法を実装している．

4 祖先ノードをキャッシュする新たなアルゴリズムの実装とその評価

本節以降から，小椋によって実装されている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を new と呼び，キャッシュをせずに素朴に上方向を辿る手法を old と呼ぶ．本節では，new とは異なるアルゴリズムで祖先ノードのキャッシュを行うアルゴリズムを提案する．さらに，そのアルゴリズムに基づく実装 (以降 our と呼ぶ) と new，old について，実行時間とメモリ必要量について評価した結果を述べる．

4.1 アルゴリズムの提案

祖先を求める対象であるノード (カレントノードと呼ぶ) すべてを再帰的に処理するための関数と，各カレントノードについてその祖先ノードを再帰的にすべて求めるための関数を分けたアルゴリズムを提案する．

4.2 実験

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[*1] 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．

4.2.1 実験環境

実験を行う際の環境を示す．

- PC
  - 3.2GHz Intel Core
  - 16.0GB メモリ
  - VirtualBox6.0.8 上の Ubuntu18.04.2 LTS
- XQuery 処理
  - Saxon-HE 9-8-0-3j
  - Java ver 1.9.0-181

4.2.2 実験内容

本節では，our と new と old についての実行時間とメモリ必要量の比較実験を行い，その評価を行う．用いるパス式を表 1 に示す．また実験結果を表 2 に示す．

4.2.3 実験結果とその評価

表 2 より，我々が実装したプログラムは，小椋の実装した new と比較して，実行時間が長いがメモリ必要量は削減できていることが分かる．

実行時間が長くなってしまった理由は，我々の実装した

[*1] http://xmlcompbench.sourceforge.net

表 1 我々が実装した手法の評価実験に用いるパス式

	パス式
1	/descendant::creator/ancestor::history
2	/descendant::author/ancestor::source
3	/descendant::tableLink/ancestor::tableHead

表 2 実験のメモリ必要量と実行時間

	our のメモリ必要量 (MB)	new のメモリ必要量 (MB)	old のメモリ必要量 (MB)	our の実行時間 (ms)	new の実行時間 (ms)	old の実行時間 (ms)
1	11	13	11	3,031,142	1,068,732	199,514
2	13	17	13	2,875,219	1,122,129	327,429
3	23	25	23	4,780,663	1,196,419	647,465

プログラムは，1 つのノードごとにその祖先ノードを全て記憶しており，重複した部分を削除して保存する処理を何度も行っているため，実行時間が大幅に長くなってしまったと考えられる．一方で，メモリ必要量が削減できている理由は，小椋のプログラムではすべての祖先ノードが求まるまで 1 次元的に再帰呼び出しが続くが，我々のプログラムの再帰は 2 次元的であり 1 つのノードの祖先ノードを求め終わるごとに 1 つの再帰呼び出しが終了するため，スタックの使用量が削減できているためであると考えられる．

5 XML 文書の形状に対するキャッシュの効果の分析

本節では，小椋や我々が実装している，上方向の探索を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の効果の分析実験を行う．実験では Nasa.xml を実験の目的にあった形に変更した文書を用いる．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

実験環境は 4 節と同様である．

5.2 実験内容

この実験では ancester 軸を対象とする．この実験の結果は全て 5 回試行した平均の値である．実験で使用するパス式は表 3 に示す．

5.2.1 深さ 1 から深さ 0 へ辿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 4 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 を辿る．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 と our 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．

実験結果を図 4 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 へ辿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 5 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 を辿る．

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．

実験結果を図 5 に示す．5.2.1 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．

実験結果を図 6 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 へ辿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 6 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 を辿る．

5.2.2 の 1 つ目実験の結果から，new と our 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に new と old と 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，5.2.2 の 1 つ目の実験と同じである．

実験結果を図 7 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 は祖先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辿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

表 3　実験に用いるパス式

	パス式
4	/descendant::dataset/ancestor::datasets
5	/descendant::reference/ancestor::datasets
6	/descendant::reference/ancestor::dataset

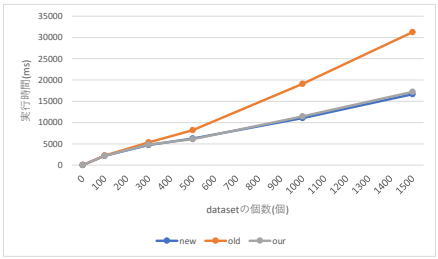

図 4　深さ 1 から深さ 0 へ辿る問合せに対するキャッシュの効果の比較の実験結果

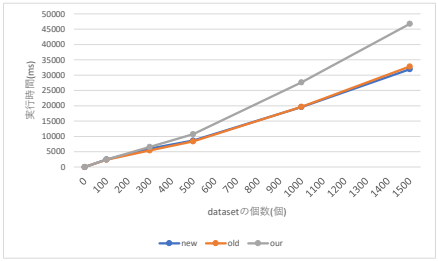

図 5　深さ 2 から深さ 0 へ辿る問合せに対するキャッシュの効果の比較の実験結果 (1)

際に素朴に兄ノードを辿っている．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，多くの兄ノードを辿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．

5.2.3 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辿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．

5.2.2 の 1 つ目の実験で 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

本研究では，祖先ノードをキャッシュする新たなアルゴリズムの提案と実装を行った．また，小椋の実装した手法

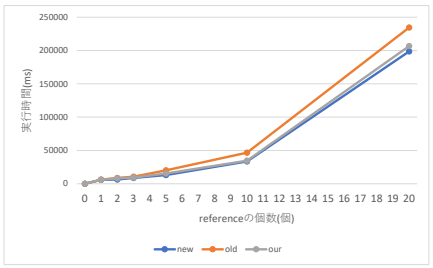

図 6　深さ 2 から深さ 0 へ辿る問合せに対するキャッシュの効果の比較の実験結果 (2)

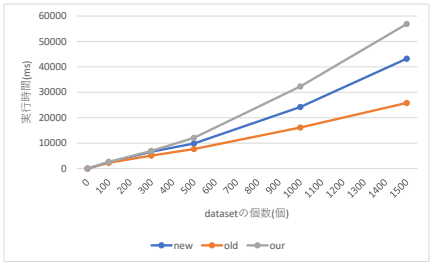

図 7　深さ 2 から深さ 1 へ辿る問合せに対するキャッシュの効果の比較の実験結果

や我々が実装した手法の，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と素朴に兄ノードを辿り親ノードへ移動する手法の比較実験をし，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辿る手法を用い，複数いる可能性があるならキャッシュのする手法を用いる，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．

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. 信学技報, 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. 信学技報, 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.