XMLデータに対する連続問い合わせの効率的な評価方法

松村　英孝† 　　田島　敬史†

† 北陸先端科学技術大学院大学　〒923–1292 石川県能美郡辰口町旭台 1–1
E-mail: †{h-matumu,tajima}@jaist.ac.jp

あらまし　日々変化するXMLデータに対して定期的に問い合わせを評価する連続問い合わせの効率化を行う．問い合わせ言語として，幅広く用いられているXPathを想定し，更新が起きたノードの情報を用いて最小限の評価を行うことで，問い合わせを一から再計算せずに，前回の解との差分を効率的に求める．このような処理を実現するために，データからあるノードが削除されると，どの問い合わせの解からどのノードが削除されるかという情報を管理する．さらに，データにノードが追加される際にも，この同じ情報を利用することで，問い合わせ処理の一部を省略し，評価の効率化を行う．

キーワード　XML，連続問い合わせ，XPath，問合せ処理

Efficient evaluation of continuous queries for XML data

Hidetaka MATSUMURA† and Keishi TAJIMA†

† Japan Advanced Institute of Science and Technology
Asahidai 1–1，Tatsunokuchimachi，Ishikawa, 923–1292 Japan
E-mail: †{h-matumu,tajima}@jaist.ac.jp

Abstract　We developed efficient evaluation techniques for continuous queries, i.e. queries evaluated periodically, on XML data changing day by day. We assume queries are written in XPath, which is widely used today. Instead of completely reevaluating queries, we compute the differences from the previous answers with minimal evaluation by using the information on changes. To achieve this, we maintain the information on deletion of which node in the data causes deletion of which node in the answers to which queries. In addition, when some node is added to the data, we can use the same information to omit the evaluation of part of queries.

Key words　XML, Continuous Queries, XPath, Query Processing

1. 背　　景

XMLはデータフォーマットとして幅広く利用されている．このXMLの特徴としては以下のようなものが挙げられる．

- データの構造と意味を保持したまま交換できる
- 構造に対する問い合わせが可能
- XMLデータはラベル付き木で表現できる

XMLは主にインターネット上でのデータ交換や，データ発信などに利用される．特にデータ発信の場合，そのデータを利用している側の各ユーザは，自分の必要なデータに対して，頻繁に起きる更新を監視したいという要求がある．

そこで，そのような要求に応えるために連続問い合わせと呼ばれる機構が研究されている．連続問い合わせ（continuous query）とは，登録された問い合わせにマッチするデータに更新が起きた際，ユーザへ通知する持続的な問い合わせである．連続問い合わせには，大きく分類すると，情報源から情報が到着するとそれに連動して実行される到着型問い合わせと，タイマーによって一定間隔で定期的に実行されるタイマー型問い合わせの2種類がある．日々変化する情報源に対して定期的に問い合わせをサーバ側で処理することで，ユーザは同じ問い合わせを何度も発行せずに，新しい情報を得ることができる．このような連続問い合わせは，膨大な量の情報が存在し，不定期に更新が起こるインターネット環境において特に有効である．

連続問い合わせを行う環境を考える場合，以下のような特徴を持っていることが多い．

- 多数の問い合わせが登録され，データの更新も頻繁
- 各更新で更新されるデータはデータ全体のごく一部
- 各更新の影響を受けるのは全問い合わせ中のごく一部
- 問い合わせの回答は，ネットワークを介して送られる

これらの特徴から生じる問題点として，まずXMLデータに更新が起きる度に，全ての問い合わせを一から再計算するのは非効率的であるという点が挙げられる．次に，同じ問い合わせを

実行するたびに，その解の全体をネットワーク経由で送ると冗長になり，ネットワークの負荷が上がるという点が挙げられる．後者については，解全体を送るのではなく，前回の解との差分だけをユーザに通知することで解消される．

そこで，本研究では，このような連続問い合わせにおいて，ノードの追加，削除の情報が与えられた際，これらの影響を受ける問い合わせの解のみについて，前回の解との差分だけを計算することにより，評価を行うサーバの負荷を減らす手法を考える．ここでは，問い合わせ言語としては，現在幅広く用いられているシンプルな問い合わせ言語の XPath [1] を想定する．

以下に関連研究，Xpath の簡単な説明，ノードの削除があった際の評価の手法，ノードの追加があった際の評価の手法について順に述べる．

2. 関 連 研 究

まず，本節では，連続問い合わせに関連したこれまでの主な研究について述べる．

最初に Xyleme [2] は，モニタリングするページの指定や，問い合わせの評価のタイミング，XML レポートの作成方法を記述するための言語を定義している．Xyleme のシステムは，一日に数百万の問い合わせを処理可能である．Xyleme のアルゴリズム [3] の特徴としては，起きた更新の影響を受ける問い合わせを一から再評価し．前回との差分を求め通知する．そのためトリガとなるアトミックイベントの集合を迅速に処理できるデータ構造を提案している．

NiagaraCQ [4] は，XML-QL に時間情報を追加した形式の問い合わせを使用している．NiagaraCQ では，登録された多数の問い合わせの中の，構造が同じで定数部分のみが異なるものをグループ化することで，効率的に処理を行う点が特徴である．さらに，最新の差分データを保存し，可能な限りオリジナルデータの代わりに問い合わせに使用しているが，この点についてはこれまでの論文では詳しく述べられておらず詳細は不明である．

3. XPath

XPath では XML データのツリー構造中の「経路」を記述することで必要なデータを特定する方法を定義している．現在のノードから次にたどるべきノードを指定する方法をロケーションパスといい，各ロケーションパスは，次のノードを探す方向を示す軸，軸の中にあるノードの種類を示すノードテスト，条件による絞り込みを行う述語の 3 つの部分からなり，「軸::ノードテスト [述語]」の形式で記述される．そして，XPath による問い合わせ式は，このロケーションステップを「/」でつないだ以下のような形になっている．

/ロケーションステップ/ロケーションステップ/...

さらに XPath では，よく用いられる軸とノードテストの組み合わせに対して，省略記法が用意されている．例えば，軸を省略した場合は，最も良く用いられる子ノードの方向を示すchild::が指定された物と解釈され，子孫の任意の型のノード全てを示す/descendant-or-self::node()/は「//」と略記可能である．XPath の問い合わせについて，図 1 の XML ツリーに対する問い合わせを例に挙げて説明する．図中のa，b，...は XML データ中でのタグ名，1，2，...は，本文中で各ノードを参照するために便宜上割り振った番号である．

Q1: /a/b
Q2: /a/b[c]
Q3: /a//d
Q4: /a/b/c/parent::b
Q5: /a/b/c/following::b

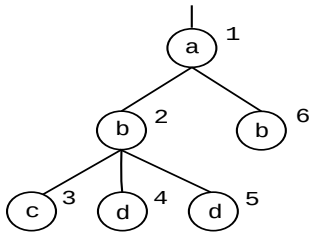

図 1　XML ツリーとそれに対する XPath 問い合わせの例

まず，最も簡単な問い合わせの例として，問い合わせ Q1 を考える．「/」は現在のノードの子にあるノードを選択するもので，Q1 の/a/b は/a の子にあるb を選択し，2，6 のb ノードを根とする部分木が解になる．

問い合わせ Q2 は述語を含む問い合わせの例である．[ ] で囲まれた部分は述語を示し，[ ] 内で指定した条件を満たすノードだけを抽出する．Q2 は/a の子の，c を子に持つようなb を選択しているので，2 のb ノードを根とする部分木が解になる．

次の例は，「//」を含む問い合わせの例である．//は現在のノードの子孫にあるノードを選択する．Q3 は/a の子孫にあるd を選択しているので，4，5 のd を根とする部分木が解になる．

次に，明示的な軸の記述を含む問い合わせの例を示す．Q4 のparent は現在のノードの親を選択するもので，/a/b/c の親にあるb を選択しているので，2 のb を根とする部分木が解になる．

Q5 のfollowing は現在のノードの XML 文書中の順番で後にあり，子孫でないノードを選択する．よって/a/b/c の後にある4，5 のd と6 のb を根とする部分木が解になる．その他に現在のノードの XML 文書順で前にあり，先祖でないノードを選択するpreceding という軸も存在する．

4. 提 案 手 法

ここでは，始めにノードの削除があった際の評価方法，次にノードの追加があった際の評価方法について順に説明する．

4.1 ノードの削除があった際の評価方法

ノードの削除の情報が与えられた際，影響を受ける問い合わせの解の変化だけを計算するために，データからどのノードが削除されると，それによってどの問い合わせの解から，どのノードが削除されるかという情報をアノテーションとして管理する．ここでは図 2 中の XML 木と，それに対する問い合わせ Q6 を例に挙げて説明する．例えば，Q6 の問い合わせを評価する際，データ中の各ノードに対してアノテーションの情報を図 2 のように付加する．ここで，(Q6,4) は，そのノードが，ノード4 を Q6 の解とするのに使われたことを表わす．これにより，3 のc が削除されると，4 のd が Q6 の解ではなくなることが，Q6 を再計算せずにわかる．

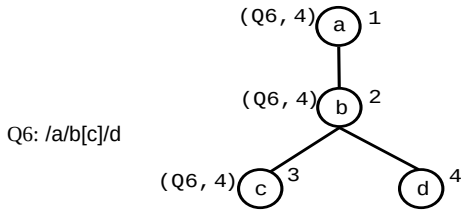

図 2　削除に関する情報を付加した XML ツリー

しかし，上の例で示したような形のアノテーションの情報だけでは不十分な場合がある．例えば，同じ Q6 を図 3 のような XML データに対して評価する場合を考える．この場合，解と

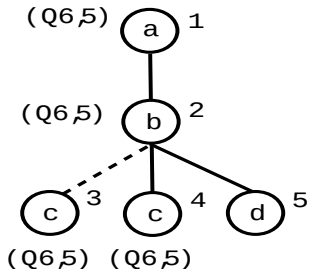

図 3　成り立たない例

なるのはノード5 であり，この解を導くマッチングとしては以下の 2 通りが存在する．

$$Q6 : \text{/a/b[c]/d} = \begin{cases} \text{/1/2[3]/5} \\ \text{/1/2[4]/5} \end{cases}$$

例えば，3 のc が削除された際，アノテーションの情報により Q6 の解から5 のd は除外されてしまう．しかし，実際には4 のc がまだ存在するため5 のd は Q6 の解のままである．

ここで，一般に問い合わせに良く用いられる/, //, [ ] だけを用いた問い合わせの場合に限定し，述語外に現れるロケーションステップにマッチするノードと述語内に現れるロケーションステップにマッチするノードの特徴を考える．最初に述語外に現れるロケーションステップにマッチするノードは以下の特徴を持つ．

- 必ず解となるノードの祖先になっている

この特徴より，述語外の部分にマッチするノードが削除される場合，そのマッチングが導く解は削除されるノードの子孫にあるため，その解となるノード自身もデータから削除されることになり，必ず問い合わせの解から除外されることが分かる．

一方，述語内に現れるロケーションステップにマッチするノードは，以下のような特徴を持つ．

- 各解毎に一つ以上ある
- 解となるノードの祖先ではない

述語外の部分にマッチするノードと異なり，述語内にマッチするノードは各解毎に一つ以上存在し，解の祖先になるとは限らないため，述語内にマッチするノードが削除されても，そのマッチングが導く解が，問い合わせの解から除外されるかどうかは判断できない．そこで，述語内にマッチするノードが削除された際に，問い合わせの解が除外されるかどうかを判定する情報をアノテーションに追加することを考える．

例として，以下のようなステップを含む問い合わせがあり，この部分にマッチするデータとして図 4 に示したようなデータがあった場合の処理について説明する．

```
···/ a [ b [ c ] [ d ] ] / ···
```

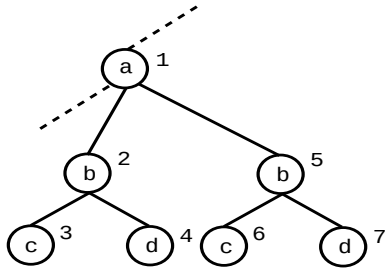

図 4　述語を含むステップ

ここで，述語内のステップにマッチするノード3 のc と4 のd が削除された際に1 のa ノードは条件を充たしたままであるが，しかし述語内にマッチするノード3 のc と7 のd が削除される際には，1 のa ノードは条件を充たさなくなることが分かる．

この例からわかるように，述語内の各ステップ毎に，そのステップにマッチするノードがいくつあるかのマッチング数の情報を記録する必要がある．そのような情報を管理するために，まず，問い合わせ中の各ステップにステップ識別子を割り当てる．この時，述語内に現れる各ステップには，それぞれ異なる素数を割り当て，述語外に現れるステップには全て 0 を割り当てることにする．上記の問い合わせ中の各ステップに，これらの数値を割り当てた例を以下に示す．

```
···/ a [ b [ c ] [ d ] ] / ···
       2   3       5
```

そして，ある解を導くマッチングにおいて各ステップにマッチするノードに，そのステップの識別子を割り当てる．また，あるノードが，問い合わせ中の二個所以上のステップで使われる場合は，それらの全てのステップの識別子を，そのノードに割り当てる．そして，割り当てられた全ての識別子と，そのノードの子孫の貢献度を乗算したものを，そのノードの，その解のマッチングに関する貢献度としてアノテーションの情報に追加する．例えば，上の例では，ノード3 のc は，

```
···/ 1 [ 2 [ 3 ] [ 4 ] ] / ···
```

というマッチングで，識別子 3 が割り当てられていたステップにマッチするので，識別子 3 を割り当てられる．このノードは，この述語部分の他のステップのマッチングには使われないので，このノードに割り当てられる識別子は 3 のみであり，このノードの最終的な貢献度は 3 となる．同様に，ノード4 のd の貢献度は 5 となる．また，ノード2 のb は識別子 2 が割り当てられたステップにのみマッチし，子孫の貢献度を乗算するので，このノードの貢献度は 30 となる．

さらに，各解に対しては，そのノードは述語部分にマッチす

る他のノードがあといくつ削除されれば条件を充たさなくなるかという情報を表す値をカウンタとしてアノテーションに追加する．上記の例では，2 のb ノードには次の値をカウンタとして割り当てる．

$$2\text{ の b のカウンタ} = 3^{(1-1)} \times 5^{(1-1)} = 1$$

同様に，1 のa ノードには次の値をカウンタとして割り当てる．

$$1\text{ の a のカウンタ} = 2^{(2-1)} \times 3^{(2-1)} \times 5^{(2-1)} = 30$$

あるノードが削除される際には，各ノードのカウンタを削除されるノードの貢献度で割り，余りが 0 の場合は，そのカウンタを持つノードは条件を充たしたままで，それ以外の場合にはそのカウンタを持つノードは条件を充たさなくなる．例えば，3 のc ノードが削除される際，3 のc ノードの貢献度は 3 で2 のb ノードのカウンタ 1 であるから，余りは 0 以外となり，よって2 のb ノードは条件を充たさなくなる．しかし，2 のb が削除される際に，2 のb ノードの貢献度は 30 で1 のa ノードのカウンタは 30 なので，1 のa ノードは条件を充たしたままである．

問い合わせ全体を考えた場合，ロケーションパス中の 1 つのロケーションステップが成り立たなければ，解はない．つまり，問い合わせ中の全ての述語の条件を充たさなければ解はないため，全てのステップの述語内にマッチするノードに割り当てられた貢献度を利用しカウンタを作成する．一般的な形として，述語の各ステップにに割り当てられた異なる素数を順に，$pr_1, pr_2, pr_3, \ldots$ とし，そのステップにマッチするノードの数を順に $l, m, n, \ldots$ とすると，

```
... / a [ b [ c / d ] ] / ...
      0   pr1   pr2   pr3        (pri は素数)
```

その時の解のカウンタは次のように表現される．

$$\text{解のカウンタ} = pr_1^{(l-1)} \times pr_2^{(m-1)} \times pr_3^{(n-1)} \ldots$$

ここで，再び問い合わせ Q6 に貢献度とカウンタを追加した例を図 5 に挙げる．

```
Q6: / a / b [ c ] / d
      0   0   2     0
```

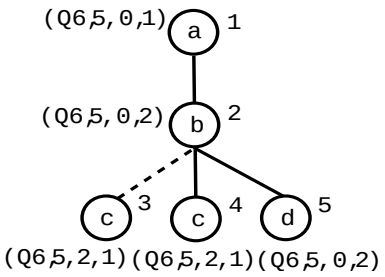

図 5　Q6 の重みとカウンタを付加した例

3 のc ノードが削除される際に貢献度は 2 で，問い合わせの解である5 のd ノードのカウンタは 2 であるから，計算した余りは 0 より問い合わせの解から除外されないことが分かる．

さらに，「//」を含む問い合わせの場合について，例として以下の問い合わせ Q7 を使って説明する．

```
Q7: / a / b [ c ] [ .//c ] / d
```

まず，この問い合わせ中の各ステップに識別子を以下のように割り当てる．

```
Q7: / a / b [ c ] [ .//c ] / d
      0   0   2       3       0
```

この問い合わせを，図 6 のデータに対して実行した場合，解となるのは，ノード6 のd である．そして，この問い合わせで 2 を割り当てたステップにマッチするノードは 1 つ，3 を割り当てたステップにマッチするノードは 2 つあるので，解6 のカウンタは次のようになる．

$$\text{解のカウンタ} = 2^{(1-1)} \times 3^{(2-1)} = 3$$

ここで，ノードに貢献度とカウンタを含むアノーテーションを追加した例を図 6 に示す．各アノーテーションは，対象となる問い合わせ，対象となる解，そのノードの貢献度，そのノードのカウンタの四つ組である．例えば，5 のc ノードが削除され

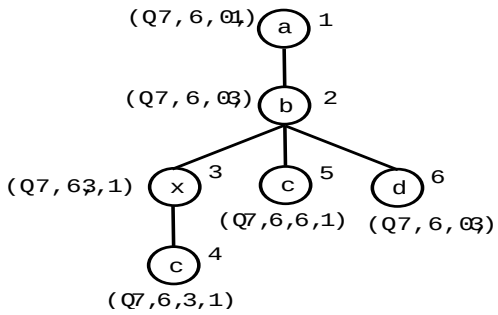

図 6　Q7 の重みとカウンタを付加した例

る際を考える，削除されるノードの貢献度は 6 で問い合わせの解である6 のd ノードのカウンタは 3 であるから，解は問い合わせの解から除外されることが分かる．5 のc ノードは問い合わせで 2 を割り当てた[c] と 3 を割り当てた[.//c] の両方にマッチすることが貢献度より判断することができる．しかし，4 のc ノードが削除される際，4 のc ノードの貢献度は 3 で，解のカウンタは 3 であるから，計算した余りは 0 より問い合わせの解から除外されないことが分かる．その後，アノテーションを更新するため，4 のc ノードの貢献度で同ステップの祖先にマッチするノードのカウンタを割り，アノテーションが削除されるノードを見つける．削除されるノードで一番親にあたる3 のx ノードの貢献度で残りの祖先である2 のb の貢献度とカウンタを割り更新を行う．このようにノードが削除された際に再計算することなく解が問い合わせの解から除外されるかどうか判定することができる．

4.2　データの追加があった際の評価手法

データに追加があった場合，追加されたノードがスタート点となるように問い合わせを変形し，評価を行う．図 7 の XML データに対する問い合わせ Q8 を例に挙げて説明する．図 7 の XML データに7 のe を追加する場合，問い合わせを以下のように変形し，評価を行う．

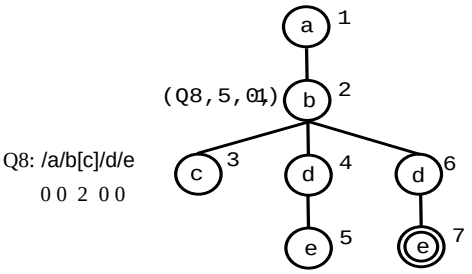

図 7　新しくノードを追加する例

Q8-1: e/parent::d/parent::b[c]/parent::a

このように追加されるノードをスタート点にして評価する際，その途中で到達するノードに，前述のような削除の際のための情報があれば，その情報を使って，それ以降のステップの評価を省略できる場合がある．例えば，図 7 の XML データでは，7 のe から6 のd と順に評価を行っていくが，2 のb まで来た時点で，このノードがあるノード（5 のe）を Q8 の解とするために使われているという情報が分かる．これにより，Q8 の問い合わせの残りの部分，すなわち/a/b[c] に対応する部分を評価せずとも，7 のe が Q8 の解になることがわかる．

また，このような場合，7 のe と5 のe の間で，/a/b[c] の部分のマッチングに関する情報を共有させることで，データに付加するアノーテーション情報の総データ量を抑えることができる．そのような共有を行うために，まず，以下の様に問い合わせの各ステップに番号を付ける．

Q8: / a / b [c] / d / e
I II III IV

そして，図 8 の様に，マッチングに関する情報の中に，新しくステップに関する情報を付加し，また，あるノードとノードがあるステップ番号については，アノテーションの情報を共有するという情報を表 1 に示したような関係に記憶する．これにより付加するアノテーションのデータ量を抑えることができる．またデータの更新が起きた際には，これらのマッチングに関するアノテーションの情報も更新しないといけないが，その更新処理についても，このように一部のアノテーションを複数の解で共有させることにより，軽減することができる．

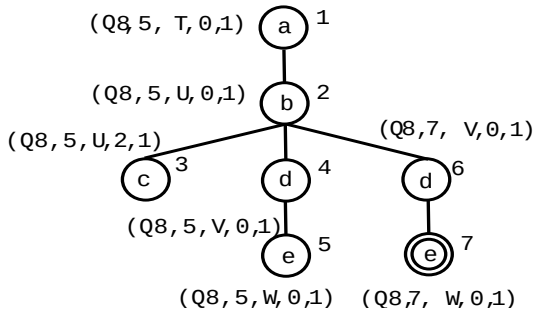

図 8　ステップ番号を追加した例

5. 考　　察

あるノードが削除される際には，影響を受ける問い合わせの解の変化をすぐに求めることができる．ノードが追加される際には，問い合わせを追加されるノード中心に変形して評価を行い，評価の途中で，既に存在するマッチングに関する情報を利用して残りの評価を省略することによって効率化を行った．しかし，途中で，利用できるマッチングに関する情報が無かった場合は，このように問い合わせを変形すると，評価の効率はどうなるかについても検討する必要がある．例えば，図 9 のような木を例にc が解になるかどうかを/a/b/c を評価して判定する場合とc/parent::b/parent::a を評価して判定する場合で考えてみる．

表 1　Q8 の解の共有カウンタ

answer	share	step
7	5	II

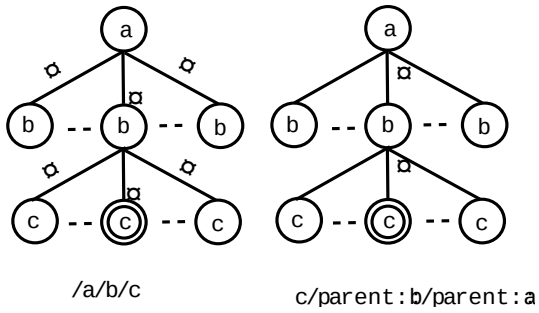

図 9　/a/b/c と c/parent::b/parent::a の比較

前者は a の子は複数存在する場合があり，b の子も複数存在する場合があるが，後者は c の parent は必ず 1 つで，b の parent も必ず 1 つである．これは XML データを RDB に格納して比較する際に，Join を行う評価時間に大幅に影響する．

逆に，もとの問い合わせ自身にparent やancestor が用いられている場合は，追加されたノードから逆にたどるように変形すると，親から子にたどるようになるため効率が悪くなる可能性がある．しかし，一般的に問い合わせの中にparent やancestor を用いる場合は，/や//を用いる場合に比べて圧倒的に少ないので，全体としては追加されたノードからたどるように変形した場合の方が効率が良い．ただし，following とpreceding に関しては，式を変形することで評価を行うノードが増え効率が悪くなる場合も考えられる．

6. アノテーションの更新

ノードに変更が起きた後，ノードに付加したアノテーションの情報を更新する必要がある．最初にノードが削除された際のアノテーションの更新アルゴリズムを表 2 に示す．まず，1 行目で削除されるノードの貢献度を調べ，0 ならば述語外のステップにマッチするノードが削除さるれることが分かる．つまり，解はそのノードの子孫にあるため問い合わせの解から除外される．問い合わせの解から除外された解についてのアノテーションは全て削除される．5 行目で，解のカウンタを削除されるノードの貢献度で割った余りが 0 の場合，述語内のステップにマッチするノードが削除されるが，まだマッチするノードが

存在するために，解は問い合わせの解のままである．この時，同ステップの祖先にマッチするノードのカウンタを削除されるノードの貢献度で割り，余りが 0 以外ならば，そのノードのアノテーションも同様に削除される．アノテーションが削除されるノードの内で，一番親に当たるノードの貢献度で残りの祖先のステップにマッチするノードの貢献度とカウンタを割り値を更新する．14 行目は，述語内のステップにマッチするノードが削除され，他にマッチするノードがないため，解が問い合わせの解から除外される場合である．問い合わせの解から除外された解についてのアノテーションは全て削除される．

表 2　ノード削除のアノテーションの更新

```
deleteNode(Node node)
1   if (削除されるノードの貢献度 = 0) {
2       /* 述語外のノードが削除された場合 */
3       除外される解のアノテーションを全て削除
4   } else {
5       if ((解のカウンタ % 削除されるノードの貢献度) = 0) {
6           /* 解は問い合わせの解のままである */
7           同ステップの祖先にマッチするノードのカウンタを
8           削除されるノードの貢献度で割り，アノテーションが
9           削除されるノードを調べる
10
11          削除されるノードで一番親に当たるノードの貢献度で
12          残りの同ステップの祖先にマッチするノードの
13          貢献度とカウンタを割り値を更新する
14      } else {
15          /* 解が問い合わせの解から除外される */
16          除外される解のアノテーションを全て削除
17      }
18  }
```

例として，問い合わせ中に以下のようなのステップがあり，この部分にマッチするデータとして図 10 に示したようなデータがあった場合の，カウンタのメンテナンスについて説明する．

```
···/ a [ b [ c ] [ d ] ] / ···
```

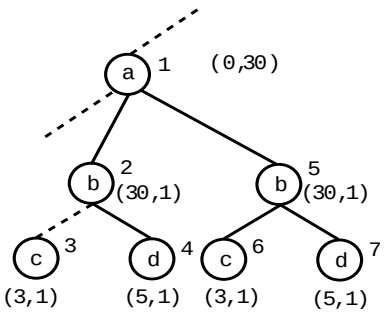

図 10　貢献度とカウンタを付加したステップ

ここで，述語内のステップにマッチするノード3 のc が削除された際のアノテーションの更新を考える．3 のc が削除された後，同ステップの祖先にマッチするノードのカウンタを3 のc の貢献度 3 で割る．この時，2 のb のカウンタを割った余りが 0 以外なので，2 のb のアノテーションも削除される．次にアノテーションが削除されるノードの中で一番親に当たるノード，この例では pf2 のb の貢献度で残りの同ステップの祖先の貢献度とカウンタを割った値が更新したアノテーションの値となる．3 のc が削除された際のアノテーションの変化を表 3 に示す．

表 3　削除の際のアノテーションの更新

node	更新前	更新後
1	(0, 30)	(0, 1)
2	(30, 1)	削除
3	(3, 1)	削除
4	(5, 1)	削除
5	(30, 1)	(30, 1)
6	(3, 1)	(3, 1)
7	(5, 1)	(5, 1)

一方，ノードが追加された際のアノテーションの更新アルゴリズムは表 4 のようになる．最初に 1 行目で，追加されるノードが述語内のステップにマッチするノードならば，同ステップの祖先にマッチするノードのカウンタに追加されるノードの貢献度を乗算し，9 行目で新しく述語にマッチするノードの貢献度を解のカウンタに乗算して，アノテーションの情報を更新する．4 行目の新しく解になる場合は，新しく問い合わせにマッチするノードに貢献度を割り当て，各ノードのカウンタを計算する．そして，9 行目で問い合わせの述語内のステップにマッチするノードの貢献度を乗算しカウンタを計算しアノテーションを追加する．

表 4　ノード追加のアノテーションの更新

```
addNode(Node node)
1   if (追加されるノードが述語内のステップにマッチする) {
2       同ステップの祖先にマッチするノードのカウンタに
3       node.contribution を乗算する．
4   } else {
5       新しく問い合わせにマッチするノードに貢献度を割り当てる．
6       各ノードのカウンタを計算する．
7   }
8
9   解のカウンタに新しく述語にマッチするノードの貢献度を乗算する．
```

7. 実　　験

本節では，前節までに提案した手法で処理時間がどの程度短縮可能かについて，問い合わせを一から再計算する方法と比較するために行った実験の結果について説明する．この実験では，XML データを関係の形で表現して関係データベースに格納し，与えられた XPath を SQL に変換して実行した．関係データベースには Oracle 9i 64bit を使用し，2CPU(900MHz UltraSPARC-CU), 6GB のメモリを搭載の Sun Blade 2000 上で実験を行った．

使用した XML データは，XML データベースの性能評価のためのツールである XMark [5] を用いて生成した．この XMark は scaling factor により生成する文書の大きさを決めることができ，

OS などに依存せず同一の文書を生成することができる．本研究では，XMark の scaling factor を 10 に設定して XML データを作成した．作成した XML データのサイズは 1,172.32(MB) で，属性ノードを除いてエンコードした結果，全部で 16,703,210 ノードで 846,755,696 バイトのデータを実験の対象とした．各問い合わせに関して，アノーテーションを付加する必要があったノードの数と，アノーテーションの総データ量を表 6 に示す．

表 5　比較に用いた XPath の問い合わせ

Path
Q1: /site[people/person/name[text()='Marek Gill']/categories/category/name
Q2: /site/closed_auctions/closed_auction[.//price[text()='146.02']]/buyer
Q3: /site/open_auctions/open_auction/bidder

表 6　アノテーションを付加したノード数とデータサイズ

問い合わせ	解の数	アノテーションの数	バイト数
Q1	10,000	20,007	434,017
Q2	5	17	421
Q3	59,777	706,193	17,099,979

最初に，ノードが削除される際，問い合わせの解が除外されるかどうか，一から再計算する場合と提案手法について比較を行った．削除するノードは各ステップ毎に，そのステップにマッチするノードの中からランダムで数個選び，それら全ての平均の処理時間を求めた．

表 7　ノード削除の処理時間の比較

問い合わせ	一から再計算した場合 (ms)	提案手法 (ms)
Q1	3,845	25
Q2	8,816	20
Q3	14,636	367

この結果より，問い合わせ Q3 の場合に他の問い合わせに比べ処理時間がかかったが，問い合わせを再計算する方法に比べ大幅に処理時間を短縮できることが分かる．しかし，提案手法ではアノテーションの更新を行う必要がある．ここで，Q1 ついてノードが削除された際，アノテーションに変更がある場合の処理時間を表 8 に示す．表 8 は解が削除されずアノテーションの貢献度やカウンタの更新が行われた場合と，解が削除されてアノテーションの削除が行われた場合の各々について，最も時間がかかった場合の処理時間を示す．

表 8　Q1 のアノテーションの更新時間

評価方法	処理時間 (ms)
アノテーションの更新	135
アノテーションが削除される場合	390

結果として，アノテーションが削除される場合は比較的時間がかかってしまった．これは共有されている解が削除された場合に，新しくアノテーションを共有する解を見つけて更新しなければならない．そのため，比較的時間がかかってしまう．しかし，共有する解の数が共有する解の数に比べ多いため，このような処理時間がかかる場合は少ない．このように，Q1 においてアノテーションを更新する時間を含めても問い合わせを再計算する場合に比べて効率が良かった．

ノードが追加される際，提案手法では最初にノードが問い合わせにマッチするかどうかを調べ．もしマッチするノードが見つかれば，マッチしたノードはアノテーションを持っているかを確認する．そのため，各ステップ毎に 2 つの SQL を評価する必要がある．例えば，ノードc が追加される場合に，問い合わせ/a/b/c を評価する際，parent::b を評価し，マッチするb があれば，b はアノテーションを持つかどうか評価を行う．もしアノテーションを持てば，そこで問い合わせの残りのステップの評価を省略できる．もし，アノテーションがなければ，parent::a についても同様に評価を行う．追加の際の処理時間の実験では，追加されるノードを含むようなマッチングについて，アノテーションを用いずに問い合わせ全体を一括して計算する方法との比較を行った．

最初に問い合わせ Q1 についての比較を表 9 に示す．この表は上から再計算を行った場合の処理時間，次に提案手法により最初のステップでアノテーションを見つけ，残りの問い合わせを省略した場合の処理時間，2 番目のステップでアノテーションを見つけ，残りの問い合わせを省略した場合の処理時間と続き，最後までアノテーションの情報を見つけることができなかった場合の処理時間を示す．この問い合わせは平均の処理時間が大幅に短縮され，最悪の場合でも，アノテーションを用いないで問い合わせ全体を一括して計算する場合の 104.7%と大幅に効率が悪くなることなく問い合わせを行うことができた．

表 9　Q1 の処理時間

評価方法	処理時間 (ms)
一括計算	3,395
最後から 1 ステップ目を評価	61
最後から 2 ステップ目を評価	81
最後から 3 ステップ目を評価	131
全てを評価	3,556

次に，追加されたノードが問い合わせにマッチしない場合を考える．表 10 は，問い合わせ Q1 を用いて行った実験の結果を示している．ここでは，Q1 を最後のステップから評価を行っていき，1 ステップ目で問い合わせにマッチしなかった場合，2 ステップ目で問い合わせにマッチしなかった場合と続き，途中のステップまでマッチしていたが最後のステップでマッチしなかった場合の処理時間の比較を行った．

表 10　問い合わせにマッチしない Q1 の処理時間

マッチしないステップ	一括計算 (ms)	提案手法 (ms)
最後から 1 ステップ目	20	10
最後から 2 ステップ目	20	71
最後から 3 ステップ目	3,345	3,426
全てを評価	11,537	11,557

問い合わせにマッチしない場合，一括計算を行う処理時間に

比べ，処理効率が大きくは変わらない結果となった．

次に問い合わせ Q2 についての表 11 を見ると，平均の処理時間は大幅に短縮することはできなかったが，Q1 と同様に最悪の場合でも一から一括計算する場合に比べ，処理効率が大きくは変わることなく問い合わせを行うことができる．

表 11　Q2 の処理時間

評価方法	処理時間 (ms)
一括計算	1,862
最後から 1 ステップ目を評価	34
述語を評価	1,870
最後から 2 ステップ目を評価	1,981
全てを評価	2,014

次に，追加されたノードが問い合わせ Q2 にマッチしない場合の実験結果を表 12 に示す．

表 12　問い合わせにマッチしない Q2 の処理時間

マッチしないステップ	一括計算 (ms)	提案手法 (ms)
最後から 1 ステップ目	10	10
述語内	1,963	1,976
最後から 2 ステップ目	931	1,880
全てを評価	941	1,991

結果として一括計算を行う処理時間に比べ，大幅に効率が悪くなってしまった．ここで，提案手法について処理時間の内訳を表 13 に示す．

表 13　提案手法の処理時間の内訳

マッチしないステップ	前ステップまで	問い合わせ	合計 (ms)
最後から 1 ステップ目	0	10	10
述語内	34	1,942	1,976
最後から 2 ステップ目	1,870	10	1,880
全てを評価	1,981	10	1,991

提案手法の場合，追加されたノードからスタートして一ステップずつ評価をしていくので，追加されたノードに近いステップに処理時間がかかる述語表現があり，追加されたノードから遠いステップで問い合わせにマッチしなくなる場合，処理時間がかかる述語表現を評価した後に，そのノードは結局，問い合わせにはマッチしないということがわかることになってしまう．これに対し，一括計算では，関係データベースシステムの最適化処理により，早い段階で問い合わせにマッチしないことが発見されるため，提案手法よりも処理時間が短くなっていると思われる．この点については，処理時間に大きく影響する可能性のある述語部分は，他の部分が問い合わせにマッチすると確定してから，最後に評価するなど，評価の順序を変更することで改善することができると考えている．

次に，Q3 の結果を表 14 に示す．単純に親をたどるようなパスでは一括計算する方がはるかに早い結果となった．原因として，各ステップの問い合わせ処理の時間が非常に高速で，マッチングしたノードがアノテーションを持つかどうか調べるための処理の方がかかってしまった．

表 14　Q3 の処理時間

評価方法	処理時間 (ms)
一括計算	17
最後から 1 ステップ目を評価	131
最後から 2 ステップ目を評価	356
全てを評価	466

これらの実験により，必ずしも常に評価の省略による効率化が行えるとは限らないことが分かった．そこで，更新が起きた際に，各問い合わせ毎に，問い合わせを一括計算する手法とアノテーションを用いる手法のどちらが効率が良いかを予測し，使い分けることで効率化を図ることが考えられる．評価の省略を行っても逆にアノテーションを持つか調べるための時間がかかり効率化を行えない場合がある．そして，問い合わせにより短縮する処理時間が大幅に変化することが分かった．例えば，問い合わせ中で処理時間のかかるステップがルートの方に偏っている場合は平均の処理時間の効率化が期待できるが，葉の方に偏っている場合の問い合わせは平均の処理時間の効率化はあまり期待できなかった．しかし，この点については，述語の場合は評価を後で行うようにすることで，ある程度改善できると考えられる．

8. ま と め

XML データのための連続問い合わせシステムにおいて，データからノードが削除される際に，登録されている問い合わせの解の変化を，その問い合わせを再計算せずに，どのノードが削除されたかの情報を用いて，最小限の評価で計算する手法を提案した．さらに，この情報を利用し，データの追加の際にも，評価すべきノードの数を減らすことで効率化を行った．今後の課題として，多数の問い合わせによりアノテーションの情報が大きくなった際，アノテーションを確認する処理時間の比較や，更新の際の処理時間の比較．さらに，前述のように，問い合わせを再計算する手法とアノテーションを用いる手法を使い分けることで，さらなる処理の効率化を行うことが挙げられる．

文　献

[1] J. Clark, and S. DeRose, editors, ”XML Path Language (XPath) Version 1.0 - W3C Recommendation”, `http://www.w3.org/TR/xpath.html`, Nov, 1999.

[2] ”Xyleme homepage”, `http://www.xyleme.com`

[3] Benjamin Nguyen, Serge Abiteboul, Grégory Cobena, and Mihaí Preda, ”Monitoring XML Data on the Web”, 2001.

[4] Jianjun Chen, David J. DeWitt, Feng Tian and Yuan Wang ”NiagaraCQ: A Scalable Continuous Query System for Internet Databases”, SIGMOD 2000, ppp.379-390.

[5] ”XMark homepage”, `http://monetdb.cwi.nl/xml/`