社団法人　情報処理学会　研究報告
IPSJ SIG Technical Report

2005－DD－52（1）
2005／9／30

XML データベースにおける全文検索機能及びスコアリングの実現

金輪 拓也

(株)東芝　研究開発センター

概要 XQuery を全文検索機能に対応させた XQuery-Full-Text という仕様が標準化されつつある。これにより XQuery をベースとしたスコアリングに対する記述が可能となるが、XML データベースにおいてスコアリング処理する場合には、従来の XQuery の処理速度を如何に低下させずに、かつ精度の高い結果を取得するか、という点が課題となってくる。本論文では、XML データベースにおいてスコアリングを高速に実現する方法を提案した。その方法として、TF-IDF 法を拡張したスコア計算方法と、取得件数が指定されている top-k クエリにおける最適化方法を提案する。前者は問合せパターンに応じてそのスコア計算方法を変更させるのもので、スコア計算をしない従来の検索方式と比較して 15%程度の性能劣化程度に収まることが分かった。後者に関しては、構造照合を含むクエリを実行した場合に、取得件数 k の値を 100 件程度とした場合、従来方式と比較して 50%以上の計算時間を削減できることを確認した。

キーワード XQuery, 全文検索, スコアリング, XML データベース

XQuery-Full-Text Processing and Scoring Method in a Native XML Database System

Takuya KANAWA

Toshiba Research & Depelopment Centor

Abstract XQuery is standard XML query language, and recently W3C proposes XQuery full-Text Language which is extended for full-text search function. This Language enables description of scroing for XML with XQuery. To apply scoring XML in XML database system, it is critical mission of high-speed processing and high-precision for XQuery. We proposes two techniques focused on high-speed rather than high-precision. One is calculation method based on extended TF-IDF, and only 15 % performace degration is occured comparing to no scoring. The other is query optimization fot top-k query processing considerd to scoring value and calculation cost, and it is possible to reduce the calculation time by more than 50% if target query fetch number is about 100.

keyword XQuery, XML full-text search, scoring, XML database

1　はじめに

XML はインターネット上の交換・蓄積を行なう上での標準形式として重要視されており、それに伴い膨大な XML データを管理することが可能な「XML データベース」に対する期待は非常に大きい。XML データベースにおける管理方法としてはスキーマが必要か否かで大きく分類される。前者はスキーマ情報などを基に、リレーショナルデータベースへのマッピングを行い管理する方法が主流であるが、マッピングコストやスキーマで表現が難しいデータなどの扱いが難しいことが問題であった。後者は一般的にネイティブ XML データベースと呼ばれるもので、XML の持つ柔軟性や拡張性を活かすべく、スキーマレスでデータを XML 処理に適した形式で管理することが特徴である[1]。本論文における XML データベースとは主にネイティブ XML データベースのことを指す。

XML データベースに対する標準の問合せ言語として XQuery[2]という言語が W3C から提案されている。XQuery は表現能力が非常に高く、構造と語彙（キーワード）を組み合わせた複雑な問合せが可能であり、関連研究としては処理をいかに高速に実現するか、といった性能を重視したものがほとんどである[3,4,5]。

一方、全文検索（IR:Information Retrieval）の研究分野ではフラットな文書に対して、近傍検索やスコアリングなど、全文検索機能に対する研究が古くからなされていた。特にスコアリングに関する研究は盛んで、近年はフラットな文書だけでなく、XML などの構造化文書に対するアプローチが増えつつある[6,7,8]。しかしながら、XQuery のような複雑なクエリではなくパス式レベルの簡単なクエリ式で問い合わせを行う場合が多く、主に簡単なクエリ式でいかに精度の高い文書を上位に返すか、という点に研究の主眼が置かれることが多かった。

これらデータベース分野(性能)と IR 分野(精度)の相反する要求を統合した言語として、XQuery に全文検索機能を記述可能とした XQuery-Full-Text というドラフトが提案され注目を浴びている[9,13]。

特に、XQuery-Full-Text では XQuery では考慮されていなかった XML に対するスコアリングの概念が追加されており、XML データベースにおいて XQuery の処理高

速性を保ったままこれらスコアリングの概念をどのように実現するかが今後の研究の焦点となる。本論文では、XML データベースにおいて特に性能を重視したスコアリング方法を提案する。本論文の構成を以下に述べる。

まず、2 章において、一般的なネイティブ XML データベースに関する処理概要と、XML データベース上で XQuery-Full-Text を実装する際の注意点を列挙する。3 章において、TF-IDF を拡張したスコア計算方法を提案し、4 章においては、スコア値と処理コストを考慮したクエリの最適化方法を提案する。これら手法に対する実験を 5 章で行い、6 章でそれらについてまとめるとともに今後の課題について述べる。

2 XML データベースとスコアリング

2.1 ネイティブ XML データベース

XML データを管理する際に、DTD などのスキーマ情報を与えずに格納するネイティブ XML データベースに関する研究が近年盛んとなっている。データベースに対する問合せ言語としては XQuery が用いられる場合が多く、これら検索処理を高速にするために様々な問合せ処理方法が提案されている[1]。

図 1 はネイティブ XML データベースシステムを実現する一般的な構成図である。データベースに格納される XML データに対して、システムがデータガイド(DataGuide)と呼ばれる構造を集約した情報を管理することで、構造に関する記述を含むクエリを高速に処理する。データガイドは構造パスレベルで等しい要素集合群に関して同一の ID が割り当てられ、これら各データガイド上のノードに対してデータガイド ID が割り当てられる。

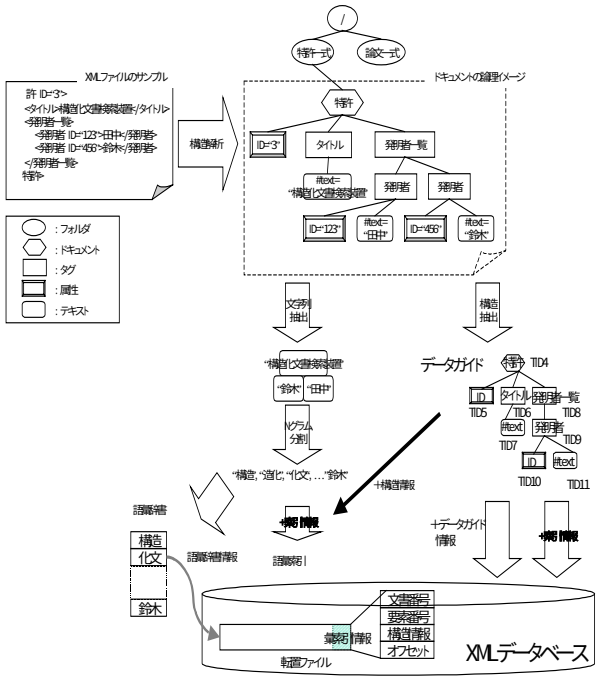

図 1 ネイティブ XML データベースシステム構成図

XML におけるテキストノード部分に関しては、従来の全文検索で用いられるように辞書情報と転置ファイルから構成される語彙索引を作成する。テキストの分割方法としては形態素辞書方式や N グラム方式が考えられるが、データベース的観点から言えば、検索漏れが発生しないことが重要であるために N グラム方式を採用する場合が多い。語彙索引における各索引は、<文書番号, 要素番号, オフセット, データガイド ID>のように、データベース上の位置情報をユニークに特定できる情報を持つ。

クエリ処理時においては、クエリを解析しコスト情報などを用いて最適プランを作成し、プランに対応してオペレータと呼ばれる処理単位ごとに候補集合を作成していくことで、最終的な解集合を得る。

2.2 XQuery-Full-Text とスコアリング

XQuery は全文検索的な用途として contains, startsWith, endsWith などの組み込み関数が用意されており、それぞれ中間一致、前方一致、後方一致レベルの検索は可能であった[10]。ただし、近傍検索やスコアリングなど、従来のテキスト検索で用いられているような機能を表現することはできず、それを踏まえて W3C が XQuery-Full-Text というドラフトを提案している。その中で、本論文は特に XQuery-Full-Text におけるスコアリング方法について注目して話を進める。

XQuery-Full-Text においてスコアリングを表現する場合の例を図 2 に示す。従来の FLOWR 構文に加えて新たに score と ftcontains という表記を用意することでスコア計算を行うことが可能となる。スコア計算方法に対してはドラフトにおいて明確な仕様は規定されておらず、これらは実装依存であるとしている。XML データベースに XQuery-Full-Text によるスコアリング方法を組み込む場合は以下の点を考慮に入れる必要がある。

1) XQuey-Full-Text は XQuery の記述を完全に包含しており、従来の処理性能を極力劣化させないこと
2) スコア計算方法は実装依存であるが XQuery-Full-Text に適した方法を提案し、極力精度を向上させること

これら要求を考えるにあたって、a) 基本的なスコア計算方法、b) スコア値を考慮したクエリ最適化方法 の 2 点に焦点を絞り、それらに対する手法を以降の章で提案する。

3 スコアリング計算方法

3.1 スコア値の計算手順

一般的な全文検索エンジン等でスコアリングを行う際には、検索語彙に対して転置索引から TF-IDF 法などを用

```
for $b in /books/book
score $s as $b/content ftcontains "web site" && "usability"
where $s > 0
order by $s descending
return <result>
        <title> {$b//title} </title>
        <score> {$s} </score>
       </result>
```

図 2 XQuery-Full-Text におけるスコアリング表現

いてスコアリングする一次検索を行った後、語彙に対する意味的解析をスコア値に反映させる二次検索を行う手順を取る場合が多い[11]。二次検索は検索精度を高めるためには有効な手法だが計算時間を要する場合が多く、ここでは検索速度を重視して転置索引を用いた一次検索のみを行う。

3.2 TF 決定方法

TF(Term Frequency)値は、転置索引中の各索引要素をスキャンする際に動的に頻度を計算することで求められる。XQuery-Full-Text においては、文書全体に限らず要素単位で条件指定したり検索結果を取得したりするので、基本的には要素内で発生した頻度を TF 値とするが、対象がシーケンスであるかどうかでその計算方法を変更する。

score で指定する式がシーケンス(複数個数から構成される)の場合はそれら TF 値を加算する。例えば図 3 の文書 1 の場合は、$b/本文の結果は 2 個の要素オブジェクトから構成されるシーケンスである。この文書に対してクエリ 1 を実行した場合、これら TF 値を加算する(TF 値 3)。クエリ 2 は$b 自体が要素そのものなので加算は行わない(各候補に対する TF 値は 1 と 2)。

ftcontains 内で&&(AND 演算)や||(OR 演算)を行う場合も、基本的にはシーケンスの要素ごとに計算を行う。クエリ 3 に対しては TF 値 4, クエリ 4 に対しては OR の場合は条件中での最小値を TF 値とし、TF 値 3 とする。

TF を要素単位で計算した場合は、サイズ補正を行ったとしてもスコア対象の粒度的に問題が生じる場合がある。例えば図 3 において/特許/タイトル ftcontains “XML”というクエリでは、対象タグだけでスコア計算を行ったとしても、検索指定した構造が持つテキストサイズが小さすぎてほとんどスコアに差がつかない。

そこで本方式においては、図 4 のように文書格納時にデータガイド単位でテキストサイズの統計値を記憶しておく。この値が基準値を下回る場合は同一の文書内において近似のデータガイドまでスコア対象となるテキスト領域を拡張して TF 値を加算する。これを拡張スコアリング範囲と定義する。

この場合の TF 値に関しては、その構造的距離に応じて完全に合致するタグと比べて 1/α倍の TF 値とする。図 3 の文書 1 に対して/特許/タイトル ftcontains “XML”のクエリを実行した場合は、<タイトル>タグそのものに対してヒットした場合を 1 に、 <本文>タグ中に発生する場合を 1/2 として計算することにすれば、TF 値は 2.5 となる。

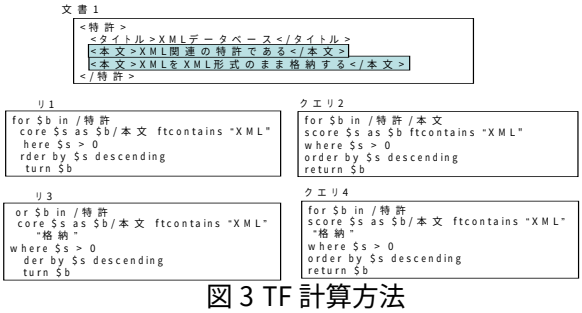

図 3 TF 計算方法

N グラム方式かつ検索キーワードが複数のグラムから構成される場合に、この方式では不当に TF 値が加算される可能性があるので N グラム中で最小の頻度を持つグラムの頻度値を基準にαを大きな値に設定し全体の TF 値への影響を抑える。N グラムが検索語彙そのものである場合はαを小さくし、TF 値への影響を大きくする。

3.3 IDF 決定方法

IDF(Inverted Document Frequency)を N グラム方式で見積もる場合、統計値を用いて正確な値を求めることは難しいとされている[12]。DF の情報をどのような単位で計算するかにより、これら値は異なってくる。

IDF 本来の意味は検索対象の文書を特徴付ける語彙であるかを決定付けるものであり、TF で対象としたスコア範囲に極力あわせる必要がある。しかしながら XQuery-Full-Text においては、検索対象は必ずしも文書ではなくクエリのパターンにより以下の 3 つの単位に分類される。

a) 文書単位
b) 要素(属性)単位
c) データガイド単位

この単位で計算した出現回数を語彙辞書に記憶させておき、クエリのパターンに応じて適宜使い分けることで検索者の意図に沿ったスコア計算が行える。例えば図 3 におけるクエリ 1 はその対象がシーケンスに対してであるので c)で求め、クエリ 2 に関しては要素そのものに対してであるので b)で求める。重複要素が存在するかどうかはデータガイド上にその統計情報を記憶させておき、判別を行えば良い。

これら IDF の値が妥当かどうかを調べるために新聞データ 5 年分(原文 2.5G 相当)で、a), b), c)方式で DF 値を求めた結果を表 1 に示す。なお、IDF 値は 1/log(総発生数/DF 値)で計算するが、ここでは DF 値のみを記す。また、新聞データの構造は図 6 のように複数の記事が纏まったものを一つの文書(新聞)とする。

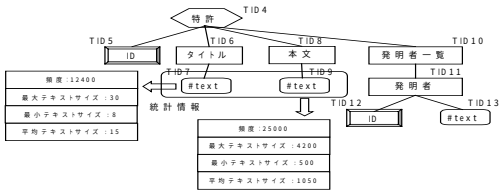

図 4 統計情報を利用した拡張 TF 計算範囲

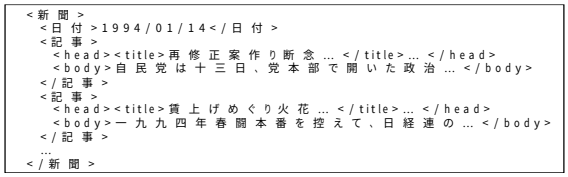

図 5 新聞記事のデータ構造

「震災」や「大統」、「統領」などは a)文書単位で見た場合は希少性が高いが、b)要素単位で見た場合は低く、c)データガイド単位で見た場合はその中間値を取る。これは「震災」は「阪神大震災」、「大統」などは「大統領選挙」の時期に集中してその単語が発生しているためである。このため同一の文書内に集中して発生する可能性が増えるため、文書単位で計算した場合の IDF 値が高くなる。要素単位はこれらまとまりとは関係無しに個別にカウントされるので DF 値としてはそれほど影響しない。これは、ランク 500「情報」とランク 513「大統」に着目した場合、a)方式だと 1.6 倍の重みの違いがあるが、b)方式だと 1.2 倍にしかならないことからも明らかである。このように IDF 値をクエリが対象とする検索範囲ごとに使い分けることによってより精度の高いスコア計算が可能となる。

なお N グラムで DF を見積もった場合は精度向上のために、最終グラム計算の際にその時点までに残った候補数で補正を行う。その候補数が DF 値と比較して明らかに少ない場合はグラム単位の DF 見積もり精度が悪いと予測されるのでこの差に応じて DF 値を修正する。

このように事前に計算しておく方法とは別に、検索時に転置索引をスキャンして DF 値を計算する方法も考えられる。ただし、この方法は、DF 値を求めるために索引を一度スキャンした後でスコア値を設定する処理を行う必要があるので、リストを 2 度スキャンすることになり、計算時間を重視した場合には好ましくない。また、N グラムベースでは結局は見積もり値になるので、本部分でコストを掛けることを避け、上記のような統計ベースの見積もり方法を採用する。

3.4 拡張 TF-IDF 計算方法

以上より、対象候補 e、検索語彙 q、検索パターン r, 構造情報を t, t における q の重みを $w_{qt}(0<w_{qt}\leqq 1)$とすると、スコア計算式は以下のようになる。

$$Score(e) = w_{qt} * (K + (1 - K) * tf(e)) * idf(q, r) \quad (1)$$

なお、K は tf の重みを調整する補正値とする。以下に本論文で提案する拡張 TF-IDF 計算方法のアルゴリズムをまとめる。なおここでは、検索対象のキーワードを分割した語彙集合 $M = \{N_k\}, N_k(k=1...p)$、語彙索引が満足しなければならない構造制約(データガイド集合)を $T = \{T_i\}(i=1,..m)$とする。

Step1: 拡張スコアリング範囲として新たに追加されるデータガイド集合を T'={φ}とする。

Step2: M を出現回数 $f(N_k)$の小さい順にソートした結果を M'とする

Step3: M'の順に N_k を取り出し、各 N_k に対して転置索引をスキャンし、索引要素を読み出して構造と位置関係を満たすものだけを候補として残す

Step4: N_x(最終のグラム処理)の場合だけ、以下のとおりスコア値計算を行う

Step4-1: T_i のテキストサイズ平均 $TS(T_i)$を、対応するデータガイドの統計情報より求め、それら $S = \Sigma TS_i(i=1->m)$を求める。

表 1 新聞データにおける DF 値統計情報

ランク	語彙	頻度	a)	b)	c)
1	改行	6992776	23275	6992776	37475
100	日本	708762	19432	388404	73918
300	問題	292001	16772	165062	45294
481	選挙	169895	9316	30773	24191
500	情報	162617	**13497**	**95363**	33986
513	大統	157240	**8295**	**77228**	21552
515	統領	157232	8297	77205	21545
558	震災	111886	6444	84726	17046
10000	活動	95068	12585	55820	26549

総文書数:23275, 総要素数: 42217506, 総データガイドノード数:791350

Step4-2: $S<t_{max}$(t_{max} は拡張スコアリング範囲を決定する閾値)ならば T に含まれない近似の位置にあるデータガイドノード NT を取り出し、S = S+TS(NT)とし、T'に NT を追加する。この手順を $S>=t_{max}$ になるまで繰り返す。この時点の状態を T'={T'i}(i = 1..y)とする。

Step4-3: TS(T'i)を求め、T'i ごとに TF の重み補正値 ST_i = h(TS(T'i), t_{max})を設定する。関数 h()は TS と t_{max} から対象構造と近似度を決定する関数である。

Step4-4: 検索対象が、文書/要素/シーケンスであるかどうかをクエリ q より判別し対応し、minDF = min (DF(Nk)) (k=1...p)する。

Step4-5: N_x(最終グラム処理開始時)の候補集合件数を C_{last} とし、C_{last} > minDF ならば minDF = C_{last}/β (β は minDF と C_{last} の格差が大きいほど minDF の値を下げるためパラメータで、β = g(C_{last}, minDF))とし、idf(q,r) = log(総候補数/minDF)とする。

Step4-6: 各索引要素 e を取り出してそれぞれに対して TF 値を求める。索引要素において文書番号が等しく、かつ取得した T''i ∈ T'ならば TF = TF + ST_i とする。T''i ∈ T ならば TF = TF + 1 とする。この値を tf(e)とする

Step4-7: スコア計算式(1) に従って該当候補のスコア値を計算し、候補として残す。全ての候補をスキャンしてこれら処理を行えば終了である。

4 スコア値を利用した最適化方法

4.1 XQuery-Full-Text 処理における top-k クエリ

一般的なサーチエンジンでは検索結果件数が予め設定(k 件)されている場合には、任意の k 個の検索結果ではない候補に対する処理をできる限り省略する top-k クエリを用いる場合が多い[13,14]。これを XQuery-Full-Text 処理に拡張する方法を提案し、更なる高速化を実現する。図 2 クエリをスコア順に 100 件取り出す場合は、XQuery-Full-Text において図 6 のようなクエリを記述することと同義である。

top-k クエリを実現するには、サーチエンジン等では語彙に対する転置索引だけを考えれば良いが、XML データベースのクエリ処理においては、プランにおけるオペレータ実行レベルでこれら処理を行う。

```
for $result at $pos in
  for $b in /books/book
  score $s as $b/content ftcontains "web site" && "usability"
  where $s > 0
  order by $s descending
  return <result>
           <title> {$b//title} </title>
           <score> {$s} </score>
         </result>
where $pos <= 100
return $result
```

図 6 図 2 クエリに対する top-100 クエリ記述方法

本方式では各オペレータで処理する最大件数(打ち切り件数)を定め、候補それぞれが持つスコア値と処理コストにより処理優先度と比較することで打ち切り候補を適宜決定する。これにより、処理コストが高く、かつスコア値が低い候補の処理を行わないことで、実行時間を削減できる。

図 7 はその処理の例を示したものである。構造照合オペレータにおける各候補の処理コストを事前に見積もり、スコア値との乗算を用いることで処理優先度を決定した上で、打ち切り閾値 0.6 以上の 2 個の候補に対してのみ処理を行っている。

以下に、top-k クエリを考えた場合の XQuery-Full-Text 処理フローを示す。

Step1: クエリを構文解析し、クエリ制約グラフを作成する。
Step2: 最適化プランを作成する
Step3: それぞれのオペレータに対して、初期の打ち切り閾値(件数)を作成する。打ち切り初期オペレータ F=NULL としておく
Step4: プランに従ってオペレータ処理を実行する。オペレータの各候補を ei とする。
Step4-1: その時点で処理する候補件数から、処理打ち切りの可否を決定する。
Step4-2: 各候補に対するスコア値の概略分布表 rank(ei)が存在する場合は、その値と Step3 で求めた打ち切り件数を照合し、残すべく最低のスコア値 α を求める。
Step4-2: 候補 ei を取り出し、スコア値 score(ei)を得る
Step4-3: ei に対する処理コスト cost(ei)を求める。
Step4-4: 処理優先度 PP_i = score(ei)*cost(ei)* σ (σ はオペレータごとに定める補正項)を計算する。
Step4-5: $PP_i < \alpha$ ならばその処理を実施せず、F=NULL の場合のみ F=現オペレータ番号を記録し **Step4-2** へ。それ以外ならば **Step4-6** へ。
Step4-6: 各オペレータにおける具体的な処理を実行する。実行した結果に対しては、スコア値の概略分布表 rank(ei)を作成しておく。
Step4-7: 全てのオペレータに対する全ての候補に対して処理を終了すれば **Step5** へ。
Step5: 検索結果件数が k 件以下の場合は F が示すオペレータから処理を再開する。F=NULL もしくは k 件以上結果が得られているならば終了。

4.2 オペレータ処理コスト

前節のアルゴリズムにおける **Step4-3** では、それぞれのオペレータ内の各候補 ei において処理コスト cost(ei)を計算する。以下に、XQuery-Full-Text のオペレータでコストを要する代表的な 2 つのオペレータについての処理コストについて考察を行う。

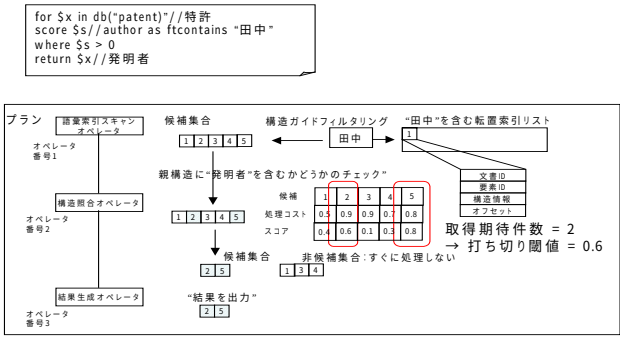

図 7 スコア値と処理コストを考慮した打ち切り方法

A) 構造照合オペレータ

構造照合オペレータは、語彙索引オペレータ実施後の出力となる候補集合を入力として、各候補が構造制約を実際に満たしているかどうかを照合し、条件を満たす候補のみ出力に残すオペレータである。

図 8 に、構造照合オペレータの入力(候補集合)100 件に対して、その候補を処理する計算時間と構造照合に要した構造段数の関係を示す。計算時間は候補によってばらつきが大きく 100 倍以上の性能差がある場合が存在する。

これら性能特異点が発生する最大の原因は、ある文書に対する 1 回目のディスクアクセスコストに起因しており、構造照合段数との関連性はほとんどないことが分かる。つまり、階層が深くなっても同一文書の 2 回目以降のアクセスは、XML データベースが管理しているキャッシュにヒットするために計算時間が削減されている。

これらより、候補集合を処理する際に同じようなスコア値を持つ 2 つの候補があったとしても、文書番号が 1 回しか存在しない候補よりは、文書番号が複数回存在する候補を優先的に処理することで、これら特異点の処理を回避でき、処理時間の削減が行える。

B) 語彙索引スキャンオペレータ

N グラム索引を用いた場合は、頻出語彙における転置索引数も膨大になる場合が多い。スコア値を基準としてこれら処理コストを削減する方法として、スコア値の高い候補を別リストとして管理する手法[15]や、スコア値に対する索引を事前に作成しておく、などといった手法[16]が考えられる。しかしながら、前者はオンラインで更新処理を行うことが必須なデータベースでは実現が難しく、また、後者はそのために B+木などで余分な索引を作成せねばならず、実用面を考えると得策ではない。

処理性能を重視すると、転置索引ファイルは一般的に複数のページ(ブロック)から構成されることから、これらページヘッダに発生する TF 値情報のサマリを残しておく方法が考えられる。検索処理時にページごとにこれらをチェックし、スコア値が高くなる候補を、ページ単位でスキップさせることで計算時間の削減が行える。

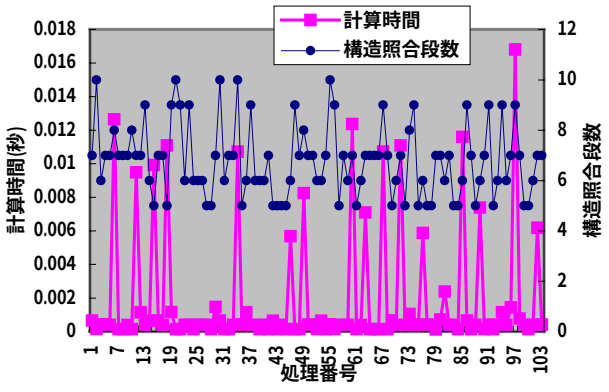

図 8 構造照合オペレータコスト

4.3 見積もり閾値の決定

XQuery-Full-Text 処理においてスコア値による打ち切りを行う上で最も難しいのが、オペレータごとに打ち切り閾値(件数)を見積らなければならない点である。論理演算、入れ子処理等 XQuery-Full-Text の仕様の複雑さに加え、N グラム索引で処理を行うといった点もその見積もりを難しくする要因の一つである。また、count 関数などを用いるクエリなど、処理の打ち切り自体を行ってはならない記述もあり、これらを事前に解析する必要もある。

本論文では、打ち切りを行うことが可能なクエリを対象とすることを前提とし、top-k クエリ処理と同様に、最終的に取得件数 k 件を満足しない場合は検索を再度実行する方式とする。但し、以下の方針を用いることで極力後戻りコストを小さくする。

1) プラン上でスコア計算に影響の無い部分に関しては、対応するオペレータの結果はそのまま保存し、再利用を行う
2) 各オペレータ処理を実行する際に、その時点で候補として残った処理件数をもとに処理打ち切り操作自体を行うかどうかを決定する
3) 後戻りが発生する際にはクエリを最初から実行するのではなく、打ち切り処理を行った最初のオペレータから処理を再開する

以上の方針のもと、4.1 節で述べたこれら打ち切り閾値の設定を行うアルゴリズムを以下に示す。

Step3-1 検索取得件数を k 件とする。

Step3-2 各オペレータ間の入出力関係を解析し、k 件取得を行うオペレータに対して出力となっているオペレータを検索する。

Step3-3 対応するオペレータに対して、出力となる件数を k'件とした場合、そのオペレータに設定された打ち切り閾値を決定する関数 γ (k')を用いて打ち切り閾値を決定する。ここで、γ に関しては例えば γ (k)=10k などで、オペレータごとに個別に設定する。

Step3-4 **Step3-3** の処理を全てのオペレータに対して伝播させていき、打ち切り閾値を計算する。

これら打ち切りの閾値に関しては、**Step4** の各オペレータが実行される度に処理件数が具体化されるので、**Step3** の打ち切り件数の値を修正すれば、この精度を高めることが可能となる。

5 実験および考察

5.1 実験環境

本方式の効果を確認するために、国内の新聞データ 5 年分データを用いて実験を行った。これらデータの特徴を以下に示す。なお、データ構造は図 7 と同じである。

- 文書数:29122 件
- 平均文書サイズ : 85K バイト
- 総文書サイズ : 2.5G バイト
- 平均タグ数/文書 : 250 個

マシンスペックは以下の通りである。

- CPU : Intel Xeon 3.06GHz×2
- メモリ:4G メモリ
- ディスク:2TB-RAID(SirialATA Raid)

5.2 拡張 TF-IDF 法に対する評価実験

ここではまず、拡張 TF-IDF スコア計算方法を行うことによって生じる性能オーバヘッドを、スコア計算を行わないで XML データベースで検索していた計算時間と比較することで検証する。なお、実験は以下の 8 個のクエリを用いた。

1)for $x in /新聞 score $y as $x//text() ftcontains “日本" return $x
2)for $x in /新聞 score $y as $x//text() ftcontains "震災" return $x
3)for $x in //記事 score $y as $x//text() ftcontains "日本" return $x
4)for $x in //記事 score $y as $x//text() ftcontains "震災" return $x
5)for $x in /新聞 score $y as $x//text() ftcontains "再利用型宇宙実験衛生フリーフライヤ" return $x
6)for $x in //新聞 score $y as $x//タイトル//text() ftcontains "阪神大震災" return $x
7)for $x in /新聞 score $y as $x//text() ftcontains "阪神大震災" && “高速道路” return $x
8)for $x in //記事 score $y as $x//text() ftcontains "阪神大震災" && "高速道路" return $x

表 2 にその結果を示す。これらより最悪でも 15%程度の劣化に収まっており、実用的にも問題無いと判断できる。クエリ 5 のような長大なグラムのほうがクエリ 3 の単一のグラムと比較して劣化が少ないように、処理性能は最終的なヒット件数に依存しているが、この原因の一つとして、ヒット件数が多い場合は、最終結果をスコア順にソートするコストも高くなることも影響しているものと考えられる。

クエリ 2 とクエリ 4 は、その検索対象がシーケンスであるか要素単位であるかが違うが、劣化率という観点から見た場合は明確な違いがなく、クエリのパターンに応じて性能が極端に異なることがないことも分かる。

精度に関する検証として、クエリ 7 及びクエリ 8 に対して、正解データ m 件を以下の手順で作成し、検索クエリ

において k=100 件を取得した場合の正解データが含まれる確率(正解含有率)を評価関数とし、実験を行った。

正解データ作成方法:本 TF-IDF 法を用いて上位 100 件を候補取得し、該当データ集合で発生する各語彙に対して更に TF-IDF により得られた上位 3 個のグラムを追加して検索を行い、その上位 50 件を正解データと見なした。

結果を図 9 に示す。スコア計算を行わない場合はおおよそ 10%程度の一致率となるが(ランダムに 10%の確率で正解データが含まれる)であるが、本方法を用いることで 70%程度まで向上させることが可能である。

また、クエリ 7 は文書単位、クエリ 8 は要素単位であるが、要素単位のほうが文書単位と比較して粒度が小さいので若干正解含有率が低下していることも分かった。本方法は、語彙間の共起関係などは用いておらず、また N グラム方式を採用しているので精度的には完全なものとはいえないが、データベース的な使い方を前提にして用いるには十分であると判断できる。

表 2 従来法との計算時間の比較

クエリ	従来法(秒)	スコア有(秒)	ヒット件数	劣化率
1	0.051	0.058	16925	1.13
2	0.042	0.045	6231	1.07
3	4.239	4.511	123444	1.06
4	0.433	0.492	21770	1.13
5	0.072	0.073	3	1.01
6	0.051	0.053	2650	1.04
7	0.137	0.15	829	1.1
8	0.225	0.244	511	1.08

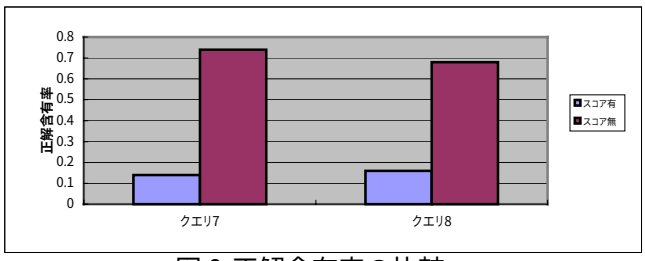

図 9 正解含有率の比較

5.3 スコア値を利用した最適化方式の評価実験

ここではスコア値を利用した最適化方式の評価実験として、取得件数 k により検索速度の違いと、打ち切り処理を設定するパラメータに対して考察を行う。

ここでは、2 語彙の AND に対する部分文書取り出しを行うクエリを考え、各語彙のヒット件数が高頻度(2-1)、中頻度(2-2)、低頻度(2-3)の 3 つのパターンを考える。

2-1) for $x in //記事 score $s as $x//text() ftcontains “日本” && “東京” return $x (106858 件 && 154420 件)

2-1) for $x in //記事 score $s as $x//text() ftcontains “阪神大震災” && “阪神高速” return $x (1179 件 && 6489 件)

2-3) for $x in //記事 score $s as $x//text() ftcontains “一般会計” && “復興関係” return $x (328 件 && 250 件)

これらクエリに対して、取得件数 k を変化させた場合の計算時間を表 3 に示す。なお、打ち切りを行わない場合は k=None とし、構造照合オペレータに対して取得期待件数の 10 倍を初期の打ち切り閾値とした。

これより、k の値が小さいほど計算時間が削減できており、特に 2-1)のような全体のヒット件数が多いような場合に効果が高いことが分かる。2-2)のような中頻度のパターンでは k=1000 程度となると、打ち切り処理を行うオーバヘッドの関係から、処理速度が逆に低下する傾向にある。2-3)のような低頻度だと、ほとんど性能的有意差は見られない。このように、本方法は最終的なヒット件数が大きい場合に特に有効である。

これは、処理打ち切り件数を増やすほど処理時間は短縮できるが、その分最終結果 k 件を満足しない確率が増えることも影響している。

図 10 は、クエリ 2-1 とクエリ 2-2 に対して、構造照合オペレータにおける打ち切り件数を変化させて、その際の最終的な結果件数(k)を示したものである。

打ち切り件数の影響度合いは、k=300 とした場合クエリ 2-1 では 1000 件程度処理すれば良いが、クエリ 2-2 では 3000 件程度処理する必要があり、影響度を考えると、クエリ 2-2 のほうが 3 倍程度大きいということが分かる。このように、クエリの後戻り処理を防ぐためには k に対してこの値を極力大きく設定する必要があるが、これら性質はクエリごとに異なり、今後、これら見積もり精度を向上させるために、語彙の共起関係などを用いるなどが必要であろう。

表 3 指定取得件数の計算時間への影響度(単位 秒)

k	1	10	50	100	300	500	1000	5000	None
2-1	0.26	0.64	0.98	1.62	3.81	4.2	5.1	42	279
2-2	0.15	0.33	0.65	1.12	1.44	1.67	1.81	1.91	1.83
2-3	0.12	0.12	0.13	0.12	0.13	0.13	0.13	0.12	0.12

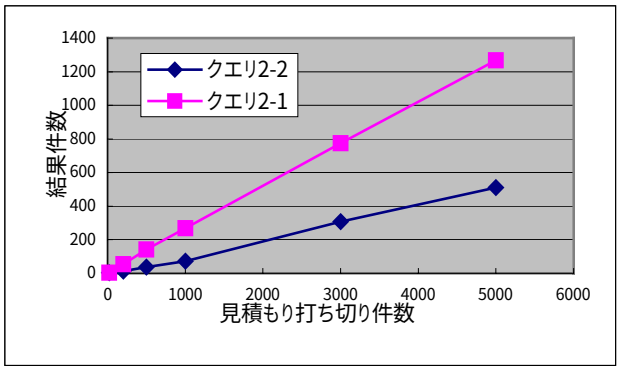

図 10 見積もり件数と結果件数の関係

6 結論と今後の課題

本論文では、XML データベースシステムにおいて特に性能を重視したスコアリング方法として、TF-IDF を拡張し、クエリのパターンによりそのスコア値を変更させるスコア計算方法と、取得件数が指定されている top-k クエリにおける最適化方法を提案した。

前者に関しては、スコアリングを用いない従来の検索処理と比較して、概ね 15%以内の劣化程度に収まることが分かった。また、検索精度に関しても作成したテストセットに対して従来法と比較して 6 倍以上の正解含有率となり、実用的に耐えうるものと判断した。なお、今回はテストセット及び評価方法は独自のものを用いたが、今後は XML の検索ベンチマークを行う団体として有名な INEX[17]データにおいて評価を行っていく予定である。

後者に関しては、キーワードを指定して部分構造を取得するクエリに対して、取得件数 k の値が 300 程度などの比較的小さい場合に計算時間を 50%以上削減できることが分かった。特に、頻出語彙を含む場合は更に効果が高い。しかしながら、処理打ち切りに関する見積もり精度を定量的に判断する方法など問題点も残っており、今後は共起関係を用いたり、パラメータチューニングを行っていく必要があろう。

更に、今回のようなクエリパターンだけでなく、途中で打ち切りを実施してはならないクエリや、より複雑なクエリに関する top-k クエリの高速化方法に関する研究も行っていく必要がある。

本論文は精度よりは性能を重視し、極量少ない情報量で高速にスコアリングを実現することを目指したが、XQuery-Full-Text で記述可能な距離情報を利用したスコア値計算[18]や、XML の要素を横断した形でクエリ処理を行い[19]、その結果をスコアリング方法に反映させるなどといった、スコア計算のための幅広い条件を組みこんで精度を向上させる、精度向上に対するアプローチに対して取り組んでいく予定である。

参考文献

[1] 服部雅一, 野々村克彦, 金輪拓也, 末田直道, “XML データベースにおける検索グラフによる検索最適化手法”, 情報処理学会論文誌 ,Vol.43, No.SIG12(TOD16), 2002

[2] The World Wide Web Consortium. XQuery 1.0: An XML Query Language, W3CWorking Draft. http://www.w3.org/TR/xquery/.

[3] Gottlob,G., Koch, C.,Pichler,R ”Efficient Algorithms for Processing XPathQueries.” In: Proc. 28th Int. Conf. on Very Large Data Bases, pp.95-106, 2002

[4] 加藤弘之, 安達淳, ”XQuery での contains()の早期評価による XML 集約ビューへの問合せ最適化手法”, NII Journal No.6, 2003

[5] Christian Mathis,Theo Härder, ”A Query Processing Approach for XML Database Systems. Grundlagen von Datenbanken”, pp.89-93, 2005

[6] Torsten Grabs, Hans-Jörg Schek, ”PowerDB-XML: A Platform for Data-Centric and Document-Centric XML Processing”, Xsym, pp.100-117, 2003

[7] L.Guo, F.Shao, C.Botev, J. Shanmugasundaram, “XRANK: Ranked Keyword Search over XML Documents”, SIGMOD 2003.

[8] Y. Hayashi, J. Tomita, G. Kikui, “Searching Text-rich XML Documents with Relevance Ranking.”, SIGIR Workshop on XML and Information Retrieval, 2000.

[9] The World Wide Web Consortium, “XQuery 1.0 and XPath 2.0 Full-Text”, W3C Working Draft 4 Apr 2005, http://www.w3.org/TR/2005/WD-xquery-full-text-20050404/

[10] The World Wide Web Consortium, “XQuery 1.0 and XPath 2.0 Functions and Operators”, W3C Working Draft, http://www.w3.org/TR/xquery operators/.

[11]Armin Hust, “Query Expansion for Web Information Retrieval”, QEWIR, 2002

[12] 小川泰嗣, 山本研策, 真野博子, 伊東秀夫, “全文検索システムのための複数転置ファイルを用いた登録高速化とランキング検索”, DEWS, 2002

[13] Emiran Curtmola, Sihem Amer-Yahia, Philip Brown, Mary Fernandez, “GalaTex: A Conformant Implementation of the XQuery Full-Text Language”, International World WIde Web Confernece, 2005

[14] Sanjay Agrawal, Surajit Chaudhuri, Gautam Das, Arstides Gionis, “Automated Ranking of Database Query Results”, CIDR2003, 2003

[15] 原田昌樹,佐藤進也,風間一洋,“索引篩法‐大規模サーチエンジンのための高速なランキング検索法”, DEWS2003, 2003

[16] Kyu-Young Whang, Min-Jae Lee, Jae-Gil Lee, Min-Soo Kim, Wook-Shin Han. "Odysseus: A High-Performance ORDBMS Tightly-Coupled with IR Features," ICDE, vol. 00, no. , pp. 1104-1005, 2005

[17] “INitiative for the Evaluation of XML Retrieval”, http://inex.is.informatik.uni-duisburg.de/

[18] 辻裕樹, 藤本典幸, 荻原健一, “検索質問に含まれる単語と適合文書内の単語の距離に着目した適合フィードバックの改善”, DEWS2004, 2004

[19]S. Amer-Yahia, L.Lakshmanan, S.Pandit. FleXPath:Flexible Structure and Full-Text Querying for XML. SIGMOD, 2004