DTM に基づく XML データベースのための逆経路索引

油井　誠† 宮崎　純† 植村　俊亮†

† 奈良先端科学技術大学院大学 情報科学研究科　〒 630–0192 奈良県生駒市高山町 8916–5
E-mail: †{makoto-y,miyazaki,uemura}@is.naist.jp

あらまし　我々はこれまでに DTM(Document Table Model) という XML データ表現形式の一種を拡張した二次記憶との相性の良い XML ストレージ手法を提案してきた．DTM を用いた問合せ処理は，子ノードや兄弟ノードといった近傍ノードへのアクセスが支配的な参照局所性が高い問合せに対しては有効であるが，子孫ノードを全探索するような参照局所性が低い問合せについては，検討の余地があった．本稿では，参照局所性が低い問合せについて，逆経路索引を用いることでアクセスが効率化されることを示す．

キーワード　XML ストレージ, XQuery, XML データベース

Reverse-Path Index for an XML Database based on DTM

Makoto YUI†, Jun MIYAZAKI†, and Shunsuke UEMURA†

† Graduate School of Information Science, Nara Institute of Science and Technology
Takayama 8916–5, Ikoma, Nara 630–0192 Japan
E-mail: †{makoto-y,miyazaki,uemura}@is.naist.jp

Abstract　We have been proposed an efficient XML storage scheme for secondary storage based on a variant of Document Table Model(DTM) which expresses an XML by a table form. The query processing with DTM was efficient enough for queries whose locality of referred blocks is high, but it needed more consideration on queries whose locality of referred blocks is low, such as scanning whole descendant nodes. In this papaer, we introduce a Reverse-Path index and show that the proposed index-access method improves the performance.

Key words　XML Storage, XQuery, XML Database

1. はじめに

W3C で標準化された XML は，組織間のデータ交換形式として標準の地位を確立し，計算機上のデータ永続化形式としても利用が広がっている [1]．今後，蓄積される XML データの増加に比例して，XML データベースシステムに求められる XML データ処理能力も高まっていく事が予想される．

これまで XML データベースの研究は，XML の木構造 (XML 木) のノード符号化方式，索引方式，XPath のパス処理の効率化といった問合せ処理の最適化を中心に研究されてきた [3]．大規模 XML データベース処理を実現するにあたっては，問合せ処理手法にも影響を与える XML データの二次記憶への格納方式も欠かせない因子である．しかし，これまで XML データベースの研究は関係データベースを活用した手法が主流であり，XML ストレージ方式の研究事例は少ない．

XML 木の関係表への写像では，一般的に，XML 木のノードを基本単位とする．その為，XML データのサイズの増加に比例して関係表のタプル数が増加することが問題として指摘されている [7]．また，タプルに写像された XML ノード群は，一般的な関係データベースの行領域確保の仕組みでは，ディスク上の配置が疎らになり XML 木における近接関係が失われる可能性が高いことにも留意が必要である．

XML 木の近接関係は，XML 問合せ処理に重要な役割を果たす．XML 問合せ処理においては，XML 木の軸を辿る操作が中核操作であるが，軸を辿る操作は起点ノードを中心とする局所性の高いものであることが多いからである．

我々はこれまでに DTM(Document Table Model) という XML データ表現形式の一種を拡張した二次記憶との相性の良い XML ストレージ手法を提案してきた [12]．DTM を用いる手法は，XML 木を表に写像するという点で関係データベース手法との類似性があるが，多くの関係データベースを用いた XML データベース手法と異なり，我々の手法は，(1) XML 木の近接関係 (文書順) がディスク上の配置に反映され，(2) アクセスの単位はノード単位ではなく，読み込みはエクステントを利用して一括して行い，書き込みはページ単位に細粒度に行う．その為，関係データベースを用いた XML データベース手法でしばしば問題とされてきたタプル数の増加に対して強固である．

一方で，DTM を用いた問合せ処理は，子ノードや兄弟ノードといった近傍ノードへのアクセスが支配的な参照局所性が高い問合せに対しては有効であったが，子孫ノードを全探索する

ような参照局所性が低い問合せについては検討の余地があった．本稿では参照局所性が低い問合せについて，逆経路索引を用いることでアクセスが効率化されることを示す．また実験により，これまでの関係データベース手法 [13] [15] では実現されてこなかった高いスケーラビリティを示す．

2. DTM に基づく XML データベース

2.1 Document Table Model(DTM)

Document Table Model とは，Apache Xalan-J [10] XSLT Processor に実装されたことで注目された，実行性能の効率化と記憶領域の最小化を目的とした計算機上での XML データの内部表現形式である．DOM(Document Object Model) に対して，オブジェクトを用いない数値型で構成される表で XML 文書を表現する特徴から Document Table Model と呼ばれる．DTM ではノード固有の整数 (表の索引) をノードの識別に用いる．そのノード ID により，QName(名前空間+ローカル名)，整数のインデックス，そして文字列バッファにおけるそれぞれのノードのテキスト値へのオフセット等を管理する．XML の木構造の保存はノード ID を用いたリンクによる．

原始データ型で XML の木構造を表現できるという特徴により，オブジェクト生成による実行効率の劣化，及びオブジェクトが消費するメモリフットプリントを抑制することができる．オブジェクト生成負荷の高い Java や C#等における実装において XML を扱う場合の効率に優れる．主要な XQuery/XSLT 処理系 [11] [10] が DTM と類似の内部表現形式を取っている為，物理スキーマが DTM に基づく我々のアプローチは，これら実装手法における一次記憶上のデータ表現に対して二次記憶への拡張を行う際の透過性に優れる．

2.2 二次記憶への格納

DTM の永続化は DTM の表 (**DTM 表**) をブロック化し，二次記憶上に配置されるファイルに対してページングを行うことにより実現する (図 1)．

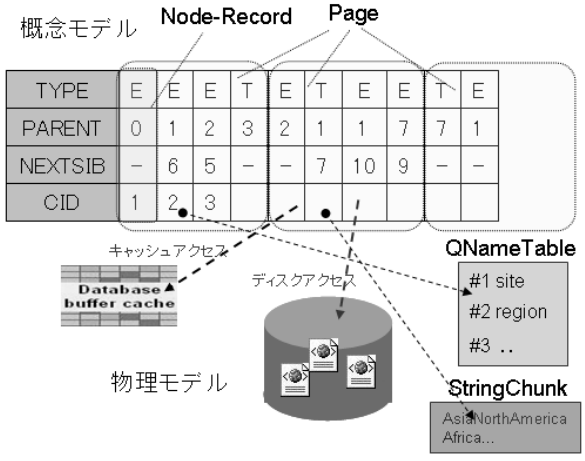

図 1　DTM の概念図

エクステントを利用した物理構成

DTM の物理構成は，図 2 のように 4 つの層から構成される．DTM 列は DTM 表の物理モデルであり，ノード情報が格納される基本レコードである．ページは提案システムにおける基本 IO 単位である．**エクステント**(Extents) は連続するページをまとめた概念で，デフォルトの設定では 32 個のページをまとめた 64KB の記憶領域である．一般に，エクステントとは論理的に連続した記憶領域を提供するものであり，XFS [14] などのファイルシステムや一部の商用関係データベースにおいて利用されている．我々の提案においては，連続するページをエクステントに連続して割り当てることでディスクブロックが連続するように促し，ディスクアクセスの効率化を図っている．

DTM におけるページング処理では，読み込みはエクステント単位に行い，書き込みはページ単位に行う．ページとエクステントを別に設けたのは，(a) ページの読み込みを粗粒度で行うことにより IO 回数を抑えることと，(b) 細粒度の更新手段を提供するためである．更に，エクステントを用いることは，(c) ページ読み出し時に要求ページに対しての先読み (**プリフェッチ**) に相当する．

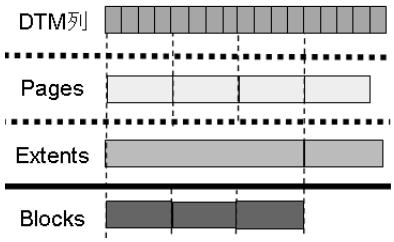

図 2　DTM の物理構成

2.3 アクセスパターンの分析

DTM を用いた問合せ処理は，子ノードや兄弟ノードといった近傍ノードへのアクセスが支配的な参照局所性が高い問合せに対しては有効であるが，子孫ノードを全探索するような参照局所性が低い問合せについては検討の余地があった [12]．

本節では，検討材料として XML ベンチマークツール XMark [6] のスケールファクタ 1 の XML 文書 (113MB) を用い，DTM に対する直接アクセスが支障をきたす参照局所性の低い問合せ (Q7) のアクセスパターン，アクセスパターンの分析より浮上したバッファ管理上の問題について議論する．

参照局所性の低い問合せ

起点ノードから参照するノード群 N_i=$\{N_1,N_2,...,N_n\}$(n はノード数) を結ぶ枝数をそれぞれ El_i=$\{El_1,El_2,...,El_n\}$ とし，参照ノード群のその枝数の平均 $avg(El)$ を $\frac{\sum_{i=1}^{n} El_i}{n}$ とすると，$avg(El)$ が高い問合せほど XML 木レベルでの参照局所性が低くなる．参照局所性のある問合せを効率的に処理する為に効果的なノードの二次記憶への配置は，問合せ処理方式に依存する．例えば，child::site/child::regions という child 軸を二度辿る処理をする場合，set-at-a-time 方式で処理する場合では子ノードがまとめて近接配置されている方が効率的であるが，tuple-at-a-time 方式でパイプライン処理をする場合では，site[1]/region[1], site[1]/region[2],..,site[last()]/region[last()] という順にレコードアクセスされるため，文書順にノードが配置されている方が効率的となる．我々の XQuery プロセッサは，BEA/XQRL XQuery プロセッサ [28] や Saxon [11] と同様にイテレータ木に基づく処理モデルを採用しており，問合せはパイプライン処理される．

提案システムにおける実際の DTM へのアクセスパ

ターンの例を次に示す．図 3 は，複数の //(descendant-or-self::node()/child::) を有する問合せであり，図 4 がそのページ要求パターンである．// を含む問合せは，一般的に $avg(El)$ が高く，参照局所性が低い問合せの代表的なものである．

図 4 の縦軸は要求されたページ番号を表し，横軸は 1 回のレコード要求を 1 単位時間とする時間軸である．縦軸のページ番号は，下限ほど文書ルートの割り当てられたページに近く，上限ほど XML 文書の末尾付近に現れるノードが格納されたページに近い．つまり，文書順に昇順である．

y 座標の 0 から 40000 付近までの XML 文書に割り当てた多数のページを要求する右上りのアクセスパターンが見てとれる．これは，count 関数の引数における // のアクセスが図に表れている．$p 変数が，$doc(\text{"}auction.xml\text{"})/site$ ノードに割り当てられているが，この site ノードは文書ノードの為，その子孫ノードは残りすべてのノードである．このような // と含む問合せにおいては，順次多くのページが読み込まれることとなる．

// を含むような参照局所性が低い問合せについては，逆経路索引等の索引手法との連携が不可欠と言える．論文 [26] で Wang らは，近傍ノードへのアクセスを効率的に扱うことを主眼として設計された NoK [9] ストレージ手法の問題点として，// を含む問合せの評価に時間がかかるということを挙げている．提案手法においては，この問題に対処するために，アクセスパスの最適化として 3. 節で述べる逆経路索引を導入する．

XMarkQ7 複数の // を有する問合せ

```
let $auction := fn:doc("auction.xml") return
  for $p in $auction/site return
    count($p//description) + count($p//annotation)
  + count($p//emailaddress)
```

図 3　XMark 問合せ Q7

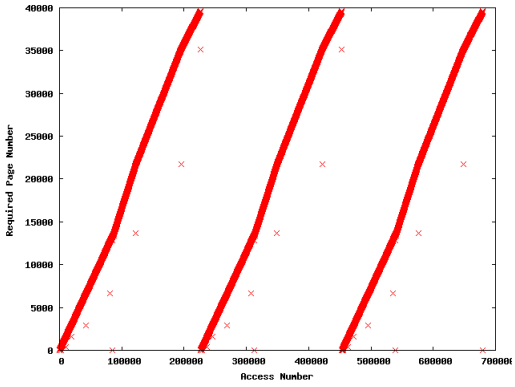

図 4　XMark Q7 のページ要求パターン

バッファ管理上の問題点

シリアライズ処理においては，シーケンシャルアクセスが頻繁に発生するため，バッファ管理戦略について局所参照に対処する必要がある．シリアライズ処理後の結果サイズが大きい XMark Q10 におけるページ要求パターン (索引利用なし時) は図 5 に示す通りである．

提案システムでは，バッファ管理アルゴリズムをこれまで利用していた LRU からシーケンシャルスキャンに強固な **2Q** [25] に変更した．結果として，バッファのワームアップを施していないケースでも，XMark のスケールファクタ 10 の Q10 で，DTM の IO 回数が 8%減少する効果が得られ，シーケンシャルスキャンに強固なバッファ管理戦略を取ることの意義が確認された．

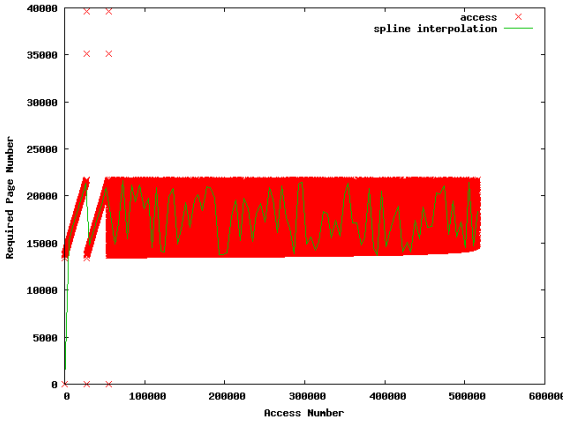

図 5　XMark Q10 におけるページ要求パターン (索引利用なし)

3.　逆経路索引

提案システムでは，DTM に対する直接アクセスが性能に支障をきたす一部のパス処理 (顕著な例として // を含む問合せやステップ数が大きい経路式) について，逆経路索引を用いるようにアクセスパスを最適化する．提案システムで用いる逆経路式の定義を図 6 に与える．

提案システムは XQuery [4] 仕様の全構文をサポートするが，経路情報を利用するアクセスパスの書き換えを行うのは，問合せ式の部分式が接頭辞として，論文 [22] で定義されている単純絶対経路式 $XP(/,//,*)$ を持つときである．相対的な経路式表現 $RelativePathExpr$ [4] については，巡航アクセスされる．

逆経路索引を用いた索引アクセスへのアクセスパス書き換えは，図 6 で与える逆経路式の定義にマッチし，かつ，// を含むかステップ数が 2 以上の場合に有効となる．これは，局所性の高い問合せクラス $XP(/)$ については，DTM に対して直接アクセスを行う場合と比較して，索引参照後の間接アクセスのオーバヘッド(注1)が存在するためである．

経路索引の問題点

経路索引は構造結合の回数を抑え性能を向上させる有効な手段である [23] が，既存の経路索引を実用的な面から見ると，次のような問題がある．

(a)　経路式はサイズが大きく，索引のキーと索引全体 (経路式と経路式識別子が消費する領域の総和) のサイズが大きい．

(b)　名前空間に対応していない為，全ての要素指定アクセスにおいて名前空間を意識する必要がある XQuery や XPath 2.0 [5] においては経路索引がフィルタとしてしか機能しない

(c)　$//site$ のような索引参照時に利用される接頭辞が短くなる経路式の処理が非効率である．

(注1)：索引参照と索引アクセス後に該当するノードレコードを参照するコストが存在する

定義 名前空間に対応する逆経路式

```
ReversePath ::= Step Separator
              | Step Separator ReversePath

Step ::= NameTest
       | '@' NameTest

NameTest ::= QNameID

Separator ::= '/#'

QNameID ::= Integer
```

図 6　名前空間に対応する逆経路式

$(a)(b)$ については，Pal らが提案している手法 [2] におけるコンパクトな逆経路式中のロケーションステップに QName 識別子を (UTF-8 符号化により圧縮し) 割り当てることで対応した(注2)．一般に文書中に現れる QName の重複を除いた総数は限られるため，経路索引のサイズ低減にも繋がる．Pal らの経路索引においては，// 軸を辿る操作について付加的な操作が必要であったが，吉川らが論文 [15] で提案しているデリミタを用いることでパターンマッチのみで // を処理可能とした．関係データベースにおける like 句と同様の処理 (B+木の範囲スキャンと文字列照合) で，// を処理する．

さらに，索引のサイズを抑える目的で Simple-Prefix B-trees [16] に基づく B+木の拡張を導入した．Simple-Prefix B-trees は，葉ノードページ以外におけるノードページのキーとして，B+木を辿るのに必要最小限の情報 (Shortest Possible Separator) が保存される B+木の改良である．

提案システムにおいては，Simple-Prefix B-trees に加えて，ノードページに含まれるキー群の共通接頭辞を算出し，各キーには共通接頭辞からの差分を保存することで，ノードページにおけるエントリ充填率を高め，索引サイズの増加を抑制した．重複キーが存在する場合には，一次記憶上ではキーのインスタンスを共有し，ノードページを二次記憶にページアウトする際には，重複キーの代わりにキーが共有されているかのフラグを書き込むようにすることで，ノードページにおけるエントリ充填率と B+木全体の必要格納領域を減量した．重複値を効率的に管理することで，二段階の索引参照 (経路式 → 経路識別子 → ノード識別子) を一段とすることが現実的となる．これらの拡張は，重複値の多い経路索引や接頭辞の共有部分の多い XML 木のラベリングにおいて特に有効である．

尚，B+木においてはキーだけでなく各値のポインタも全体からみて大きな比重を占める．提案システムでは，8 バイトのポインタについて variable-byte coding [27] で 1-9 バイトに符号化している．

(c) については，逆経路索引を利用することが索引による選択率の向上に繋がる．例えば，*//site* という検索において既存手法において索引による前方一致探索を行う場合の選択率は低いが，逆経路にすることで有効に索引が利用される．一般的に，XML 木においてはルートに近いノードまでの経路ほど経路式中での出現率が高いことが多い (例えば，ルートノードまでの経路は全ての経路式で共有される) ため，接尾辞をノードの選択に用いることや後方一致探索を利用することが，経路式のパターンマッチでは有効となることがある．

(注2)：本稿中の表記では可読性の為に QName 識別子をそのまま用いる

3.1　索引利用時のアクセスパス

本節では，XMark の問合せ Q7 と Q8 を例にとって索引を用いた場合の問合せ処理を解説し，索引を用いた場合のページ要求パターンの変化を示す．

単純経路処理

XMark Q8(図 7) の問合せ処理では，問合せ処理器は最適化フェーズにおいて，次のアクセスパス書き換えを行う．以下，アクセスパスの書き換え後の問合せプランを図 8 を用いて説明する．

- (a) の箇所について

/child::site/child::people/child::person を 34/#33/#0/#をキーとする逆経路索引へのアクセスに書き換える．数値は QName の識別子である．

- (b) の箇所について

/child::site/child::closed_auctions/child::closed_auction を 74/#73/#0/#をキーとする逆経路索引へのアクセスに書き換える．

実行時には，それぞれ逆経路索引をキーとして対応するノードのアドレス (DTM 配列の添え字) を求める．

XMarkQ8 構造結合を有する問合せ

```
let $auction := doc("auction.xml") return
for $p in $auction/site/people/person
  let $a :=
   for $t in $auction/site/closed_auctions/closed_auction
   where $t/buyer/@person = $p/@id
   return $t
  return <item person="{ $p/name/text() }">
       { count($a) } </item>
```

図 7　XMark 問合せ Q8

```
1. for $p in
2. (a) RewriteInfo − > 34/#33/#0/#
3. return
4.    <item person="$p/child::name/child::text()" (e)>
5.    fn:count(
6.      (b) RewriteInfo − > 74/#73/#0/#
7.          [child::buyer/@person (c) = $p/@id (d)]
8.    ) </item>
```

図 8　**XMark Q8** 擬似アクセスプラン

ループ中で束縛される変数を持つ経路処理

XMark Q7(図 3) の問合せ処理では，問合せ処理器は最適化フェーズにおいて，次のアクセスパス書き換えを行う．

- (a) の箇所について

/child::site を 0/#をキーとする逆経路索引へのアクセスに書き換える．

- (b) の箇所について /child::site/descendant::description を 9/%#0/#をキーとする逆経路索引へのアクセスに書き換える．この時，結果から$p 変数の子孫であるものだけを抜き出す．
- (c) の箇所についてはキーを 65/%#0/#とする (b) と同様の処理である．
- (d) の箇所についてはキーを 35/%#0/#とする (b) と同様の処理である．

---

1. for $p in ($a$) **RewriteInfo − > 0/#**
2. return fn:count(
3. (b) **RewriteInfo − > 9/%#0/#, filter $p**)
4. + fn:count(
5. (c) **RewriteInfo − > 65/%#0/#, filter $p**)
6. + fn:count(
7. (d) **RewriteInfo − >35/%#0/#, filter $p**))

図 9　**XMark Q7** 擬似アクセスプラン

Q8 は$auction/site の処理結果シーケンスからアイテムごとに return 句の処理を行う問合せである．return 句にある (b)(c)(d) は共に，(a) の結果に依存する．for ループの処理では$p 変数の値に依存する結果のみを抽出する必要がある．そこで，木ラベルを用いた構造結合を行う．B+木の性質を利用することで効率的に処理している (以後，本稿中でこのフィルタを**構造フィルタ**と呼ぶ)．そのアルゴリズムを擬似コード (図 10) を用いて解説する．

IN: idxCond, ancestor　OUT: ptrs

1. ptrs := パス索引.find(idxCond);
   経路索引を参照し，経路にマッチするノードのアドレスを取得
2. ptrs := sortIfRequired(ptrs);
   必要ならば，結果アドレスを文書順に並び替える．
3. ptrs := filter(ptrs, ancestor);

4. **function** filter(ptrs, ancestor) {
5.　ancestorLabel := ラベル索引.getLabel(ancestor);
   ancestor ノードの識別子からそのノードのラベルを取得する．
6.　for(i:=0; i<ptrs.length; i++) {
   全ての ptrs ノード ID 群についてノードアドレスごとに
7.　　targetLabel := LabelIndex.getLabel(ptrs[i]);
   そのラベルを取得し，そのラベルを targetLabel とする．
8.　　if(!isDecendantOf(targetLabel, ancestorLabel))
9.　　ptrs[i] := nil;
   targetLabel と ancestorLabel を比較し，target が ancestor の子孫でなければ，その結果アドレスを無効化する．
10.　}
11.　return ptrs; フィルタリングされた結果を返す．
12. }

図 10　**Algorithm** 構造フィルタ

引数の idxCond は逆経路索引に対する問合せを示すオブジェクトであり，ancestor は，図 9 のケースを例に取るとフィルタの基準となるループ変数$p のノード識別子を示す．3 行目で呼び出す filter 関数が，構造フィルタの中核モジュールであり，ここで構造結合を行う．図 10 では言及していないが，ノード ID 群が文書順に整列されている場合に二分探索で無効化するアドレスを発見し，配列中の無効化した要素より左に位置する要素群を返すという最適化を行う (6–11 行目に相当)．

構造結合のアルゴリズムとして定評のある Stack-Tree-Desc [18] では，構造判定を行う処理コストは $Cost_{st} = |A|+|D|+$結果ノード数 ($|A|$ は先祖のノード数，$|D|$ は子孫ノードの数) で表すことができるため，効率的である．経路情報と Stack-Tree-Desc を用いる先行研究も存在する [23]．一方で，Stack-Tree は入力集合とするノード情報をエントリ数の分，二次記憶から読み込む必要があり，その際の索引参照するなどの二次記憶へのアクセスコストが問題点として知られている．

文献 [24] によると構造結合のコストは，(1) 索引アクセスのコスト，(2) ソート操作のコスト，(3) 構造判定のコストからなる．我々の手法では，Ancestor ごとに Descendant の二分探索を行うため，構造判定のコストは $|A| * |D|log(|D|)$ となるが，主記憶上での構造判定を行うコストに対して二次記憶へのアクセス回数を重視し，非起点ノードの情報を二次記憶から取得する (索引アクセス) 回数を抑える戦略をとっている．

3.2　索引利用時のアクセスパターン

本節では，逆経路索引を利用することで，DTM におけるページアクセスパターンがどのように変化するか，どのような傾向を示すかを説明する．

索引を利用したときの Q8 についてのアクセスパターンが図 12 である．提案システムでは，図 8 における内部ループの Join 属性 (d) と Join 属性の値 (b) を予め計算し，次に外部ループを実行して (c) に対応する Join 属性の値を取得するという Join 処理をしている．図 11 及び図 12 の見方としては，y 軸の 35000〜40000 付近が (b) 及び (c) に関するアクセスで，y 軸の 14000〜22000 付近が (a)，(d) 及び (e) に関するアクセスである．(b) 及び (c) に関するアクセス箇所については，索引を利用しない場合の図 11 と比べて，索引を利用した場合の DTM へのアクセス回数が落ちている．外部ループのアクセス箇所については索引を利用した場合の方が，y 軸のアクセスされるページ数が少なくなっている．これは (a) を索引アクセスにしているためである．DTM へのアクセス回数に大差が表れていないのは，索引を利用していない箇所 (e) に依存して変数 p に束縛されたノード付近のページが読み込まれるためと考えられる．

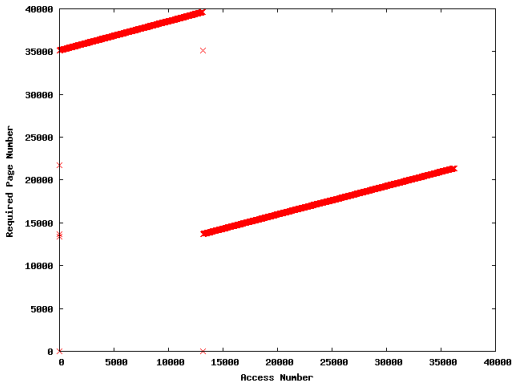

図 11　XMark Q8 のページ要求パターン (索引利用なし)

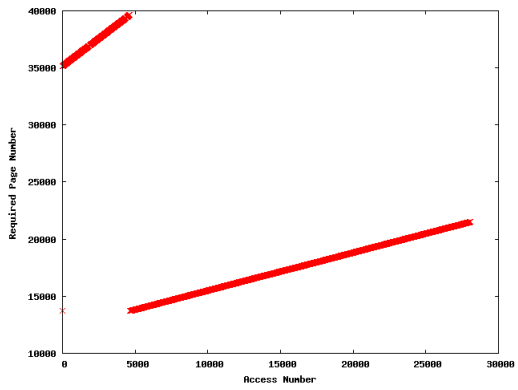

図 12　XMark Q8 におけるページ要求パターン (索引利用あり)

Q7 については，fn:count 関数の呼び出しを索引に一致したエントリ数から計算するように最適化しているため，索引アクセスだけで問合せ処理が完結する．4.2 節で，索引を利用した場合と索引を利用しない場合の DTM に対する IO 回数を比較する．

4. 実　　験

提案する DTM に基づく XML データベースに関して索引を含めた時の問合せ処理の効果を測るために，XML データベースの代表的なベンチマークツールである XMark [6] を用いて問合せ処理時間の計測を行った．

XMark のスケールファクタ 1〜10 の間の複数のテストデータセットを用いて提案手法のスケーラビリティを計測した．このテストの中で索引を用いない場合の性能を併記し，逆経路索引によるアクセスパス最適化の有効性を検証する．

実験環境は，次に示す通りである．

CPU	Intel Pentium D 2.8GHz
OS	Windows XP
メモリ	2GB
ディスク	SATA 7200rpm
Java	Sun JDK 1.6
JVM option	-Xms1024m -Xmx1024m

$-Xms$ は，JVM に初期ヒープサイズを指定するオプションである．
$-Xmx$ は，JVM の上限ヒープサイズを指定するオプションである．

比較対象

提案実装についての他の実装と比較した性能面での位置付けを与える意味で，我々のシステムが動作するのと同じく Java VM 上で動作する XQuery プロセッサとして代表的な SAXON-SA(注3) バージョン 8.73 の性能結果を参考までに併記する．

Saxon は論理モデルとして TinyTree という内部表現を利用している．TinyTree はリンク情報などからなる複数の配列から構成されるが，論理的に表構造であり，TinyTree は本質的には DTM と同様の構造である．

実験環境において，$-Xmx$ オプションで JVM に割り当て可能なヒープ量は 1.4G バイトが限界であった．この制限のもと，Saxon を評価できたのは XMark テストスイートにおけるスケールファクタ 3 の約 340MB のデータセットまでである．スケールファクタ 4 以上では，Saxon の問合せ実行中にメモリが足りないというエラーが発生した．そのため，Saxon の性能測定はスケールファクタ 3 までの実施となっている．表 1 中の DNF(Did Not Finish) は長時間結果が返ってこないために計測を中止した項目を意味する．

(注3)：Saxon の商用版である

4.1　索引を利用した場合と利用しない場合の性能比較

表 1 は，二次記憶に格納した DTM に対して，XMark のスケールファクタ 1(113MB)，スケールファクタ 3(340MB)，スケールファクタ 5(568MB)，スケールファクタ 10(1.1GB) のデータセットにいての計測結果をまとめたものである．表 1 より，索引なしの DTM に着目して見るとデータサイズの増加に対して線形以上の性能が得られていることがわかる．

// を含む問合せである Q6, Q7, Q14, Q19 については全て，索引を利用した場合の性能が勝ることから，提案手法の課題であった局所性の低い問合せに対して，逆経路索引が有効に働くことが確認できた．また，ステップ数が 15 の比較的長い単純絶対経路式を含んでいる Q15 についても逆経路索引がある程度有効に働いていることがわかる．

一方で，問合せクラスが XP(/,[],*) の局所性が高い問合せについては索引を利用しても性能が変わらないか，むしろ若干性能が悪化するケースがみられ，局所性の高い問合せクラスについては，近傍アクセスに対して強固なストレージ手法を導入した場合，索引を利用することがメリットとならない場合が多くあった．このことは，近傍ノードへの巡航アクセスに強固な DTM ストレージの設計が有効に働いている証拠とも言える．

4.2　ディスク IO の比較

索引を利用した場合にかえって性能が悪化した理由の一つとして，索引を利用することにより DTM に対して間接アクセスが発生することになるが，索引アクセスのコストとその間接アクセスのコストの和が，DTM のデータブロックをエクステントを利用して順次読み出す場合に比べ劣ることが考えられる．本節では，実験で用いた問合せからディスク IO を調査し更なる考察を与える．

表 2 は，XMark SF10 に対する XQuery 問合せついて，索引を利用しない場合，索引を利用した場合のそれぞれについて，(1) 処理に要した時間，(2) 読み込み IO 回数，(3) 読み込まれたブロック数の合計をまとめた表である．

Q20 が顕著な例で，索引を利用した場合に，索引に一致して参照したエントリの一部が，その後のシリアライズ中にバッファから吐き出され，結果としてバッファ管理のヒット率が悪化して性能が劣化するという現象が見られた．既に問合せ処理のパイプライン化によって性能劣化を防いでいる(注4)が，更なる検討が必要な部分である．こうした問題に対しては，バッファで管理するエントリを明示的に pin/unpin する機能を付加することで性能の改善が得られる可能性があり，今後の検討課題となる．

なお，今回の実験ではキャッシュの効果を打ち消すため，問合せごとに Java コマンドを実行して性能を計測したことで，索引のバッファが充分に機能していないこともある若干，実験

(注4)：索引に一致したエントリへの参照を call-by-need に評価するパイプライン化を行わない場合はさらに性能が落ちる

結果に影響している．実際の XML データベース処理ではバッファ管理は問合せを跨いで行うため，索引が有効なケースが増えると考えられる．

5. 関連研究

Zhang らは，Next-of-Kin(NoK) Pattern Match という木構造における近親ノードを効率的に選択するパターンマッチ手法を提案している [9]．NoK パターンマッチの主張は次のとおりである．XQuery UseCase [17] に現れるクエリの構造関係のうち 2/3 が child 軸であることに代表されるように，一般的に XML 問合せにおける構造関係は局所性が高いものが多く，選択率が低い．こうした局所性が高く選択率が低い問合せに対して，構造結合 [18] を用いる事は非効率であり，近親ノードを選択する軸評価 (例えば，child や next-sibling) には巡航アクセスを用いるとしている．そして NoK パターンマッチを効率的に評価する為に，近親ノードを物理的に近接配置し，予想される IO コストを抑える物理格納手法を提案している．Zhang らの研究と我々の研究は，参照局所性を意識した二次記憶への格納手法という点で類似するが，二次記憶上での XML 文書表現方法は大きく異なる．Zhang らの研究は Subject-tree と呼ばれる文字列表現により XML を表現するのに対し，我々の手法は論理的な表である DTM に基づいた物理格納を行う．また，Nok では局所性の低い問合せへの対応が特別になされていないが，我々の手法では局所性の低い問合せに対処するために逆経路索引を利用した．

XQuery をサポートする NXDB として著名な eXist [19] 及び BDB XML [20] は，ノード ID をキーに B+木の葉に DOM ノードを保存する仕組みになっており，ノード情報を参照する際の IO アクセスは B+木へのアクセスに基づく．DTM と異なり一時記憶上の表現はオブジェクトであるので，多くのノードを処理した場合の GC の負荷が無視できない．MonetDB の研究グループが公開している XMark ベンチマーク結果(注5) によると，結果サイズが大きく多くのノードを参照する必要がある問合せ Q10 について，eXist 及び BDB XML は極端な性能劣化が現れている．一方で，表形式を用いてオブジェクトの生成を抑えている我々の方式では GC による顕著な性能劣化は現れない．我々の提案実装を，SF10 の XMark 問合せ (索引なし時) をプロファイラを用いて計測した範囲では 5〜6%程度の GC 負荷であった．尚，他の XML ストレージ手法との比較については，論文 [12] を参照されたい．

6. まとめ

本論文では，DTM に基づく XML データベース手法を補う逆経路索引を提案した．鍵となるのは，エクステントを利用することで必要とされたページに対して文書順のアクセスパターンに適した先読みを行うこと，そして逆経路索引を利用することで局所性の低い問合せにも対応したことである．

今後の課題として，アクセスパターンを動的に分析することによって得られる情報により，プリフェッチやバッファ管理戦略を切り替えるといったデータベースの自動最適化を行うことや，XQuery Update [21] の更新処理への対応が挙げられる．XQuery Update は現在のところ仕様策定段階であるが，トランザクション隔離レベルとしてスナップショット隔離レベルが想定されている．スナップショット隔離レベルをサポートする為の DTM の構造について検討を進めていく．

(注5)：http://monetdb.cwi.nl/projects/monetdb/XQuery/Benchmark/XMark/

7. 謝辞

本研究の一部は，科学技術振興機構戦略的創造研究推進事業 (CREST) 「情報社会を支える新しい高性能情報処理技術」ならびに科学研究費基盤研究費 (A)(2)(課題番号:15200010)，若手研究 (B)(課題番号：17700109)，文科省の大都市大震災軽減化特別プロジェクト (代表:河田惠昭) の支援による．ここに記して謝意を表す．

文献

[1] OASIS Open Document Format for Office Applications (OpenDocument) http://www.oasis-open.org/committees/office/
[2] Shankar Pal, Istvan Cseri, Gideon Schaller, Oliver Seeliger, Leo Giakoumakis, Vasili Vasili Zolotov: Indexing XML Data Stored in a Relational Database, VLDB (2004).
[3] 吉川 正俊: データベースの観点から見た XML の研究，日本学術会議，2002 年情報学シンポジウム講演論文集, pp. 25–31 (2002).
[4] XQuery 1.0: An XML Query Language http://www.w3.org/TR/xquery/
[5] XML Path Language (XPath) 2.0 http://www.w3.org/TR/xpath20/
[6] A. R. Schmidt, F. Waas, M. L. Kersten, D. Florescu, I. Manolescu, M. J. Carey, R. Busse. The XML Benchmark Project. Technical Report INS-R0103, CWI (2001).
[7] 天笠俊之, 植村俊亮: リージョンディレクトリを用いた関係データベースによる大規模 XML データ処理, 日本データベース学会 Letters, Vol.3, No.2, pp.33–36, 日本データベース学会 (2004).
[8] System RX: One Part Relational, One Part XML, Kevin Beyer(IBM Almaden), SIGMOD, pp. 347–358 (2005)
[9] N. Zhang, V. Kacholia, and M. T. Ozsu, A Succinct Physical Storage Scheme for Efficient Evaluation of Path Queries in XML, ICDE, pp. 56–65 (2004).
[10] The Apache Xalan Project. http://xml.apache.org/xalan-j/
[11] Saxonica Saxon http://www.saxonica.com/
[12] 油井 誠, 宮崎 純, 植村 俊亮: 効率的な XQuery 処理のための DTM に基づく XML ストレージ，電子情報通信学会技術研究報告, Vol.196, No.148, DE2006–34, pp.73–78 (2006).
[13] 油井誠，森嶋厚行: PostgreSQL を用いた多機能な XML データベース環境の構築，情報処理学会論文誌：データベース, Vol. 44, No. SIG 12 (TOD19), pp. 11–22, 情報処理学会 (2003).
[14] Adam Sweeney: Scalability in the XFS File System. USENIX Annual Technical Conference (1996).
[15] Masatoshi Yoshikawa, Toshiyuki Amagasa, Takeyuki Shimura and Shunsuke Uemura: XRel: A Path-Based Approach to Storage and Retrieval of XML Documents using Relational Databases, ACM Tran. Internet Technology (TOIT), Vol. 1, No. 1, pp. 110–141 (2001).
[16] Rudolf Bayer, Karl Unterauer: Prefix B-trees, ACM Trans. Database Systems (TODS), v.2 n.1, pp.11–26 (1977).
[17] XML Query Use Cases http://www.w3.org/TR/xquery-use-cases/
[18] Shurug Al-Khalifa, H. V. Jagadish, Nick Koudas, Jignesh M. Patel, Divesh Srivastava, and Yuqing Wu: Structural Joins: A Primitive for Efficient XML Query Pattern Matching, ICDE (2002).
[19] Wolfgang Meier: Index-driven XQuery processing in the eXist XML database, XML Prague (2006).
[20] Berkeley DB XML http://www.oracle.com/database/berkeley-

	113MB (SF: 1)			340MB (SF: 3)			568MB (SF: 5)		1.1GB (SF: 10)	
	Saxon	索引なし	索引あり	Saxon	索引なし	索引あり	索引なし	索引あり	索引なし	索引あり
Q1	9.875	1.235	1.219	26.875	2.11	2.94	2.922	3.172	4.828	4.859
Q2	10.203	1.657	1.61	27.968	3.141	3.157	5.47	5.61	8.907	8.343
Q3	10.328	2.62	2.94	27.641	4.281	4.344	5.344	5.14	11.375	11.281
Q4	10.375	2.531	2.562	27.219	5.703	5.937	9.406	9.563	17.25	17.625
Q5	10.25	1.703	1.782	25	2.89	5.234	5	5.86	7.813	9.796
Q6	9.906	3.969	0.859	27.328	10.797	1.281	17.375	1.469	52	2.281
Q7	11.047	8.78	1.484	27.734	23.172	3.468	39.344	5.422	102.547	8.937
Q8	11.344	2.5	2.485	27.875	5.578	5.687	8.672	8.906	20.484	16.313
Q9	914.672	5.875	6.172	DNF	18.672	23.391	39.344	43.63	90.703	95.968
Q10	13.422	19.328	19.5	36.828	68.672	69.875	142.562	146.453	333.703	340.671
Q11	766.344	8.485	8.797	DNF	50.734	53.281	135.391	141.391	522.703	512.109
Q12	351.812	6.625	6.359	DNF	35.453	36.609	90.641	93.469	332.94	322.438
Q13	10.719	1.656	1.734	30.484	2.828	4.657	5.844	5.734	8.75	8.641
Q14	10.781	7.172	5.594	32.234	17.625	20.859	44.734	26.547	72.515	47.78
Q15	9.828	1	0.812	26.719	1.547	1	2.63	1.328	9.61	6.563
Q16	10.5	1.109	1.62	25.797	1.781	1.766	2.5	2.468	4.359	6.454
Q17	10.469	1.515	1.516	27.016	2.797	2.766	4.78	4.31	7.343	8.563
Q18	10.672	2.62	2.16	27.36	4.47	4.14	6.313	7.125	12.391	17.25
Q19	11.5	6.31	5.375	34.625	23.125	21.531	44.172	40.828	101.312	97.969
Q20	10.313	2.547	2.875	27.938	6.688	6.875	10.891	13.641	24.609	37.344

表 1　pDTM のスケール特性 (計測単位:秒)

	出力サイズ	索引なし			索引あり		
	(KB)	処理時間	IO 回数	ブロック数	処理時間	IO 回数	ブロック数
Q1	1	4.828	2605	83328	4.859	2598	83136
Q2	3425	8.907	4317	138112	8.343	4310	137920
Q3	1688	11.375	4357	138112	11.281	4351	139232
Q4	0	17.25	4352	139232	17.625	4345	139040
Q5	1	7.813	1454	46525	9.796	1447	46303
Q6	1	52	17162	549139	2.281	0	0
Q7	1	102.547	38461	1230712	8.937	1	31
Q8	9543	20.484	4056	129758	16.313	4046	129471
Q9	12067	90.703	96149	3075997	95.968	96129	3075391
Q10	250181	333.703	252139	8068385	340.671	252112	8067584
Q11	9974	522.703	6954	222465	512.109	6900	220800
Q12	1773	332.94	6954	222465	322.438	6900	220800
Q13	62002	8.75	449	14337	8.641	439	14048
Q14	594	72.515	12821	410254	47.78	4332	138623
Q15	104	9.61	1455	46536	6.563	813	26009
Q16	57	4.359	1455	46536	6.454	1448	46314
Q17	4876	7.343	2605	83328	8.563	2598	83136
Q18	694	12.391	4352	139232	17.25	4345	139040
Q19	10814	101.312	50239	3076352	97.969	50229	3076031
Q20	1	24.609	10414	333123	37.344	18141	580512

表 2　ディスク IO の比較 (SF:10)

db.html

[21] XQuery Update Facility
http://www.w3.org/TR/xqupdate/

[22] Thomas Schwentick: XPath query containment, ACM SIGMOD Record, Vol.33, No.1, pp.101–109 (2004).

[23] 江田 毅晴, 鬼塚 真, 山室 雅司: XML データの要約情報を用いた高速な XPath 処理方法, 電子情報通信学会論文誌 D, Vol.J89–D, No.2, pp.139–150, (2006).

[24] Yuqing Wu and H. V. Jagadish: Structural Join Order Selection for XML Query Optimization, ICDE (2003).

[25] Theodore Johnson and Dennis Shasha: 2Q: A Low Overhead High Performance Buffer Management Replacement Algorithm, VLDB, pp.439–450, (1994).

[26] W. Wang, H. Wang, H. Lu, H. Jiang, X. Lin, J. Li. : Efficient Processing of XML Path Queries Using the Disk-Based F&B Index, VLDB, pp. 145–156 (2005).

[27] I. H. Witten, A. Moffat, T. C. Bell: Managing Gigabytes, 2nd ed, Mrogan Kaufmann, 1999.

[28] D. Florescu, C. Hillery, D. Kossmann, P. Lucas, F. Riccardi, T. Westmann, M. J. Carey, and A. Sundararajan: The BEA/XQRL Streaming XQuery Processor, VLDB Journal 13, No. 3, 294–315 (2004).