

効率的な XQuery 処理のための DTM に基づく XML ストレージ

油 井 誠[†] 宮 崎 純[†] 植 村 俊 亮[†]

大規模 XML データに対する XML 問合せ処理に当たっては、二次記憶上の XML データ格納方法と XML データへのアクセス手法が、問合せの実行時性能に大きく影響する。本稿では、DTM(Document Table Model) の一形式で内部表現された XML 文書をブロック化して二次記憶に配置し、問合せ実行中に必要なブロックを主記憶に読み込む機能の特徴とする XQuery 問合せ処理系を開発し、提案手法の有効性について検証する。

XML Storage based on DTM for Efficient XQuery Processing

MAKOTO YUI,[†] JUN MIYAZAKI[†] and SHUNSUKE UEMURA[†]

In this paper, we propose an XML storage scheme based on DTM(Document Table Model) for XQuery processing. On query processing for large-scale XML data, XML storage schemes on secondary storage and their access methods greatly affect the entire performance. In our scheme, XML data is internally represented as a set of DTM blocks, which can be directly stored on secondary storage. We also evaluate the proposed method through some experiments.

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、計算機上でのデータ永続化形式としても利用が広がっている¹⁾。こうした XML の浸透に伴い、大規模 XML データを効率的に管理する XML データベースシステムの重要性が高まっている。

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、並びに索引方式、そして構造結合や XPath のパス評価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²⁾。他方で、大規模 XML データ処理に当たっては XML データの二次記憶媒体への格納方式も効率的な XML データベース処理の為に欠かせない因子である。しかし、XML ストレージ方式の研究事例は少なく、検討が十分になされてきたとは言いがたい。

W3C で勧告候補となった XQuery⁴⁾ は、XML 問合せ言語の標準と目されるが、XQuery において XPath 1.0⁶⁾ の問合せ評価のように全ての問合せ処理に索引を用いることは困難である。その理由の一つとして、Generic Comparator(=, > など) 等、比較オペレー

タのオペランド値が他方の値型によって決定される (Type-Promotion) という特性が挙げられる。この問題により、厳密に XQuery の問合せを評価するに当たっては、予め索引を張ることが困難となっている。既存の索引方式の多くは必ずしも全ての問合せに対応することが目的ではなく提案方式の有効性の検証を目的としている為、この問題への対処を取り上げていない。Beyer らが開発した System RX⁶⁾ は、この問題を取り上げている数少ない研究である。System RX では値に索引を張るのは手動であり、明示的に値の型を指定する形をとっている。この方式には、比較値が用意した索引の型に適合しない場合は、索引を利用できないという問題点がある。こうした問題に加え、XQuery の表現能力の高さからも全てのケースに索引を用意するのは、困難である。索引を利用しない場合のデータアクセスにおいても、高い性能を出すことが要求されるため、XML 文書のディスク格納方法が重要である。

XML 格納方式は、大別すると、ネイティブストレージ方式と関係データベースをバックエンドに利用するといった非ネイティブ方式に分けられる。XML データベースの研究は、XML の木構造の特定の部分木を指定する XPath を問合せ言語と捉え、過去 30 余年に渡る研究開発により洗練された実装を持つ関係データベースへの XML データ写像方式、つまり非ネイティ

[†] 奈良先端科学技術大学院大学 情報科学研究科
Graduate School of Information Science, NAIST

ブ方式を中心に研究されてきた。一方で、関係データベースを用いた XML 問合せ処理では、大規模 XML データにおいて処理効率が悪化することが指摘されている³⁾。近年では、関係データベースにおける XML 処理においても、XML データ処理においては関係データ処理エンジンとは別に XML データ処理エンジンを利用するといった研究が現れている⁶⁾。

本研究では大規模 XML データベース処理においては効率的な XML 物理格納が重要と捉え、効率的な XQuery 処理のために二次記憶への XML データ格納方式と、格納方式に付随する XML データへのアクセス手法を提案する。

2. 関連研究

XML の物理的な格納方法に着目した研究は、これまでも幾つか提案されてきた⁸⁾⁷⁾⁹⁾²¹⁾。

スキーマ情報に基づく格納手法

Meng らはスキーマを用いた効率的な物理格納アプローチ⁹⁾を提唱している。OrientStore では予めスキーマ情報を与え、スキーマグラフをブロック化する。そして、その断片に属するノードをディスク上で近接配置することで、問合せ処理において効率的なデータアクセスを実現している。

OrientStore と我々の研究の違いは、OrientStore がスキーマ情報に与えることで効率的にデータ格納することを目標とするのに対して、我々の研究はスキーマ情報なしに効率的に動作することを目標としている点である。

物理ページサイズに基づく格納手法

Natix⁸⁾は、部分木ベース (Subtree-Base(SB)) の物理格納戦略を取ることで知られるオブジェクト指向データベース技術をベースとした XML ネイティブデータベースの包括的な研究である。XML 木を物理ページサイズに適合するように分割し、分割した部分木を 1 レコードとして (前置順に) 格納する。

Natix と我々の研究の違いは、Natix が物理ページサイズに合わせて部分木に分割するのに対して、我々の研究では XML 文書の分割を物理ページサイズではなく、問合せにおける実際の物理ページ要求のパターンに則したものにするという戦略である。

巡航アクセスの為の格納手法

その他のユニークな研究として、Zhang らが提案する巡航 (Navigational) アクセスに適した物理格納手法⁷⁾の研究がある。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。NoK パターンマッチの主張は次のとおりである。XQuery UseCase¹⁴⁾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 (locality) 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 (Structural-Join) に代表される XML 問合せ処理手法²⁰⁾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、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の主な違いとしては、論理的な文書表現の違いが挙げられる。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、我々のアプローチは、物理データアクセスについても論理的な表である DTM へのアクセスに基づいている。関係データベースの行アクセスの仕組みと同様の機構であり、概念モデルが物理的データ独立性を持つ。この戦略により、シンプルな物理アクセスを可能としている。また、既存研究が XQuery のサブセットである XPath を問合せ言語の対象としているのに対して、我々は XQuery 表現に対応することを前提にしている。

3. 提案システムの概念モデル

本節では、XML ストレージ手法の前提となる提案システムの概念モデル、一次記憶上での表現形式を説明する。

3.1 Document Table Model(DTM)

Document Table Model(DTM) とは、Apache Xalan-J¹⁰⁾ XSLT Processor に実装されたことで注目された、パフォーマンスの最適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) をノードの識別に使う。そのノード ID により、ローカル名、展開された名前、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 (論文¹³⁾において同様の手法が検討されている。)。Primitive 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パフォーマンスの劣化、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ため、Java、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¹²⁾¹¹⁾の多くがDTMと同様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。

3.2 システム概要

提案システムで用いるDTMの概要を図2に示す。図2は、図1を例にとり文書順に深さ優先順にラベル付けされたXML木をDTMとして表した例である。図1のEは要素ノード、Tはテキストノードをそれぞれ指す。実際のDTMを説明のために簡略して表記した。例えば、図2は4行の配列であるが、実際は1行の配列で表現される。実際に格納されている値と図表記の違いについては、説明で補いながら解説する。

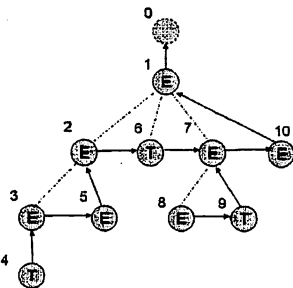


図1 深さ優先順にラベル付けされたXML木

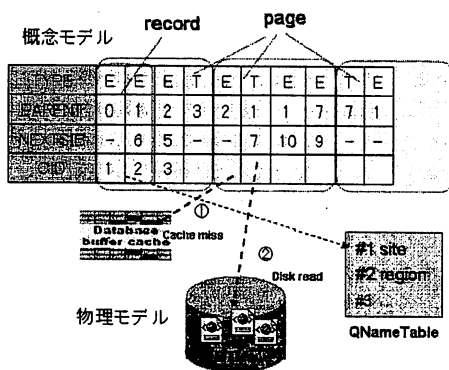


図2 DTMの概念図

表の構成

合計ノード数を n とすると図2のDTMは、 $4 \times n$ のフラットなINT配列として表現される。配列の添え字が1から始まると仮定すると、 N 行 M 列の実際の添え字は $(M-1) \times 4 + N$ で計算される。例えば、3列2行目の添え字は $(3-1) \times 4 + 2$ で10である。配列の添え字がノードハンドルとなる。このように提案システムにおいては、1ノードにつき4つのINT値、16バイトの表領域を利用する。

1つのノードが構成する4つの要素について、順に解説する。1列目は、ノード種別とノードの各種属性を表現する。下位3ビットで、7種類のノード種別を表現する。下位4ビット目はFIRST_ENTRYフラグ、下位5ビット目はLAST_ENTRYフラグである。それぞれ、左右に兄弟ノードを持つかどうかを判定するのに利用される。下位6ビット目のHAS_CHILDフラグでは、ノードが子要素を持つかどうかを表現する。続く0xFFCでマスクされる10ビットの領域は属性数を、0xFF0000でマスクされる8ビットの領域は名前空間宣言の数をそれぞれ表す。0x7000000でマスクされる3ビットの領域は、子要素数の概略を保持する予約スペースであり、残りの5ビットは将来の拡張の為に予約した領域である。2列目、3列目には親ノードの索引、弟(next-sibling)ノードの索引の値をそれぞれ保持する。4列目にはContentID(CID)を保持する。CIDは、DTM表とは別に管理する文字列のID、あるいはローカル要素名と名前空間を一意に表現する。XML文書中の文字列についてはチャンク化した上で、CIDを振って一意に管理する(この文字列管理モジュールを以下、StringChunkと呼ぶ)。チャンク化閾値を越える文字列(デフォルトでは512KB以上、設定可能)については、メモリ消費量を抑える為、メモリ内表現としてもLZ圧縮して管理する。QName(名前空間+ローカル名)については文書単位ではなく、Collectionと呼ぶ集合(ファイルシステムのディレクトリに相当)単位に一意に管理し、スペース効率を高めている(このQName管理モジュールをQNameTableと呼ぶ)。

表へのアクセス

表へのアクセスは、基本的に添え字を指定してノードに関連する数値を取得する操作APIを用いる。文字列値とQNameについては、それぞれノードハンドルからCID値を取得し、そのCID値をキーとして、それぞれStringChunk、QNameTableより取得する。XPathのAxis軸の評価(木構造のパターンマッチ)には、parent値、next-sibling値やノードの属性

値をみるなど、オフセット計算など数値演算をベースとする。問合せ処理機に対しては、論理的なデータ構造 (DTM) と操作方法が提供される。

4. DTMに基づくXMLストレージ

本節では、我々が開発したDTM(Document Table Model)に基づくXMLストレージ手法を説明する。3節で関係データベースにおける概念モデルに相当する部分を説明したのを踏まえ、ここでは、内部モデルについて説明する。

4.1 物理格納方法

3.2節で説明したように、我々のシステムではXML文書をDTMとして表現する。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(DTMsファイル)に対してページングを行うことにより実現している(図2参照)。

4.1.1 DTMの物理ファイル表現

レコードのサイズはデフォルトの設定では512列であり、一つのレコードには128(512/4)個のノードを管理する。最大長はINT長の4×512で2KBであるが、実際のレコードの物理格納においては、スペース効率を高める目的でレコードは圧縮されて保存される為、二次記憶上のレコードサイズは可変長である。以下、一次記憶上の論理的なレコードを論理レコード、論理レコードを二次記憶に格納したものを物理レコードと呼ぶ。

DTMsファイルの内部表現は図3のような形である。ファイルの先頭8byteで管理されるのは、記録済みのレコード長の合計値(Recorded)であり、Recorded値はRecordedブロックに記録される。このRecorded値が、Descriptorへのポインタを兼ねる。ここで、DescriptorはDTMsファイルのディクショナリである。レコードはRecordedブロックの後に追記する形で割り振られる。RecordedブロックとDescriptor間に空きスペースは存在しない。

ユーザが明示的に更新を指示した段階で、ファイルを管理するDescriptorブロックとRecordedブロックがファイルの内容と同期される。Descriptorは文書アクセス時に頻繁にアクセスされる為、文書アクセス中は一次記憶に保持し更新終了後(例えば、文書の追加時)にDTMsファイルとの同期を行う。このDescriptorの同期は、レコードの追加や削除があった事を判定するDirtyフラグがついている場合のみ行う。

ページング処理は、基本的にファイルシステムにおいて利用される技術に基づくが、一度に読み込むページングサイズはOSのページサイズには基づくよりも、

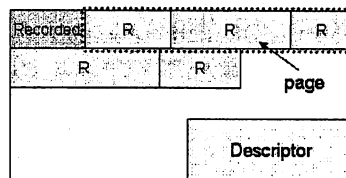


図3 DTMsファイルの内部表現

実際のXML文書アクセスにおける要求ページサイズに即したものである方がより効果的であると考え、Natix⁸⁾において、Kanneらは2K~32Kの固定長のページサイズを用いて、実験を行っているが、ページングサイズによってパフォーマンスに大きな影響が出ることを示している。

4.1.2 レコードの要求パターン

Natix⁸⁾の研究を踏まえて、補足実験として論理レコードの要求パターンの抽出を行った。この予備実験では、システムが用いる論理レコード長を512列とし、XMLベンチマークツールXMark¹⁷⁾のスケールファクタ1のXML文書(113MB)を用いた。

実際に問合せに検討したのはXMarkで用意されている20個のXQuery問合せを利用した。ここでは紙面の都合上、平均的な要求パターンであったXMark Q8と、特徴的であるXMark Q7の問合せにおけるレコード要求パターンを次に示す。尚、図中の縦軸は要求された論理レコードのアドレスを表す。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。ここで、論理レコードのアドレス割り振りは文書順に割り当てられているものとする。

文書順にアクセスされる例

図4は、XMark Q8のXQuery問合せを実行した際に、実際に要求されたレコード要求パターンを示す図である。起点となる文書ノードより、右肩上がりに文書順にアクセスされている。論文⁷⁾で報告されているように、XQuery問合せ中の構造関係の多くはchild軸へのアクセスであり、図中で右肩上がりへのアクセスに現れている。XMarkベンチマークにおける他の問合せについても多くが同様のアクセスパターンである。このアクセスパターンから、ページング戦略において構造結合の際に利用するページについては長期間メモリ上に保存しておくことが有効であることから、プライオリティ付きのバッファ管理が有効である可能性がある。

バッファ管理が難しい問合せ例

図5は、複数の//((descendant-or-self::node()/child::))を有する問合せである。

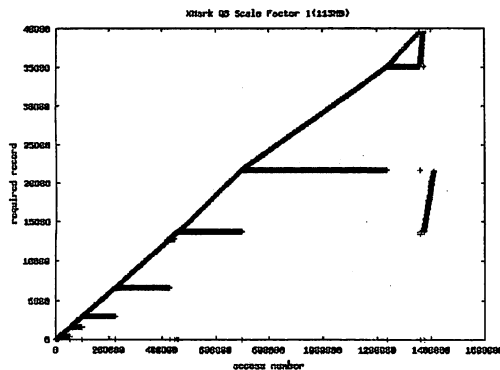


図 4 XMark Q8

XMark Q8 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>
```

この問合せにおいて、図 5 の x 座標の 1500000 近くから XML 文書に割り当てたほぼ全てのページを要求する急な右上が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("auction.xml")/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、急な傾斜アクセスパターンからみてとれるように、次々に新規ページが読み込まれていくこととなる。

XML ストレージ戦略に特化した研究においても、// のような問合せを効率的に管理するのは困難である。このような問合せについては、ネイティブ XML データベースにおいても逆経路索引等の既存の索引手法との連携が不可欠となる。また、索引が利用できないケースに対しては、3 回の count() を一度の読み込みで計算する最適化が有効である。

パス索引に関して言えば、多くの既存の研究実装において名前空間への対応が不十分である。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 においては拡張が必要である。これについては、Pal らが提案している手法¹⁸⁾における、コンパクトな逆経路式中の PathId について QName で割

り当てることで対応できると考える。

XMark Q7 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)
```

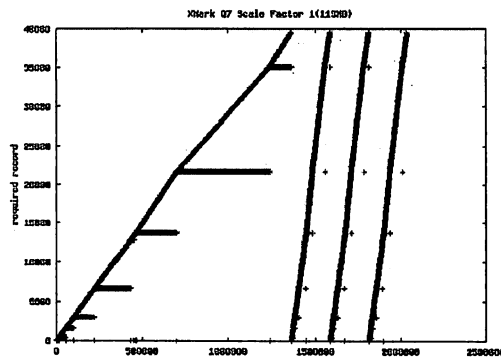


図 5 XMark Q7

4.1.3 バッファ管理

本節では、バッファ管理戦略について概要を説明する。まず、全ての取得記録は弱参照オブジェクトとして、論理記録アドレスと対に管理される。弱参照オブジェクトとは、GC 起動時に発見されれば必ず回収されるオブジェクトの事である。次に XML 問合せ処理に特化したバッファ管理戦略を図 6 を参照する形で説明する。

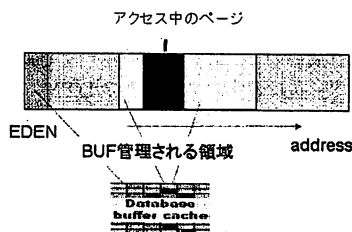


図 6 提案システムにおけるバッファ管理

現在の提案システムでは、基本的に先に説明した弱参照オブジェクトによる記録管理の他、DTMs ファイルの先頭ページを EDEN と呼ぶバッファに固定して管理する仕組みを利用している。これは、索引を利用できない場合の全ての XML 文書アクセスが

ルートノードからの巡回となるからである。索引等が利用できない場合に、先頭ページが読み込まれる事になる為、この領域については特別にバッファ管理している。

4.2 物理アクセス方法

本節では、二次記憶上に格納された論理表であるDTMに対してのアクセス手法を解説する。解説には図2を利用する。

ノードに対する情報の取得は、DTMの配列添え字に基づく数値演算である。二次記憶上に配置されたDTMsファイルに対する物理データアクセスはこの配列添え字に基づき、簡易ページャーを解してアクセスされる。提案手法における物理アクセスの手順を次に示す。

- (1) 添え字に対する実際の列データが一次記憶上に存在するかを調べ、存在する場合は二次記憶へのアクセスは行わない。
- (2) 一次記憶に該当する列データが存在しない場合、レコードの先頭の添え字(論理レコードアドレス)を計算する。
- (3) 図2のDescriptorより、その論理レコードアドレスに対する実際のファイル内での物理レコードアドレスを返す。Descriptorでは標準的なChain-Hashの実装を利用している。
- (4) レコードの先頭4ビットはレコード長を表す。この長さを見て、実際のデータブロックを取得する。取得したデータは弱参照オブジェクトとしてハッシュ表で管理される。この時、ページングサイズに基づいて続く論理アドレスのブロックをメモリ内に読み込む。この要求レコードよりも多くのレコードを一次記憶に読み込む機構は、簡易なプリフェッチ(先読み)として機能する。
- (5) 一次記憶のDTMに要求レコードが読み込まれると、後は通常のメモリ上のDTMに対するアクセスと同等と手続きが行われる。その後の手続きにおいて、要求ページがない場合には、手順1に戻る。

DTMに読み込んだページのページ(Purge)は、一次記憶に読み込まれたページ数が設定値を越えた際に行う。最後にアクセスされたノードの近いアドレスにあるノードが高い優先度で保持される。

5. 実 験

実験の目的

提案するDTMに基づくXMLストレージ手法の有

効性を測るために、XMLデータベースの標準ベンチマークツールであるXMark¹⁷⁾を用いて問合せ処理時間の計測を行った。100MBのテストXMLデータセットを用いて主記憶上で処理する場合と、二次記憶からページングしながら処理する場合の比較を行うことで、ページングに関連する負荷を計測する。さらに、主記憶上で処理することができない1GBのデータセットを用いて大規模XMLデータ処理における提案XMLストレージ手法の有効性の検証を行う。

比較対象

関係データベースを用いた大規模XMLデータベース環境や、XML分散処理システムなど多くの研究が提案されているが、実験で評価されているのは100MB程度までのデータサイズまでであり、評価されているクエリも単純なロケーションパスのみのものが多数を占める。論文²²⁾で、Bonczらは商用や研究システムを含めた既存XML問合せ処理エンジンについて、公表されている内容や独自の調査*によって性能の比較表をまとめている。この調査から、GBクラスのXMLデータ処理の計測値を提供しておりシステムとして公開されているものは、MonetDB²²⁾及びX-Hive²³⁾、及びBDB XML²⁴⁾である。X-Hive、BDBは共にプロプライエタリな商用パッケージである事と、MonetDBが相対的により優れた性能を出すと報告している²²⁾ことから、我々の提案システムの比較対象としてはMonetDBが適当である。しかし、ここで我々の複数の実験環境において、MonetDB/XQuery Version 0.10は1GBのXMLデータのロードに常に失敗するという問題が発生した。その為、MonetDB(とX-Hive)との比較には、Bonczらが論文で公表している値を、参考までに併記する形をとった。

実験環境

実験環境は、表1に示す通りである。ただし、MonetDBのベンチマークにおけるテスト環境は、OS:Linux 2.6.1, CPU: 1.6GHz Opteron, メモリ: 8GB, HDD: 8ATAドライブで構成されるRAIDシステム²²⁾である。計測に用いたのは、XMarkのスケールファクタ1(113MB)及び、スケールファクタ値10(1.11GB)のテストデータで、用意されている20個のXQuery問合せについて提案方式の評価を行った。

5.1 実験結果

実験結果は図7の通りである。計測値の単位は秒である。図7の表記としては、ERRはプログラムの問題により計測ができなかった項目、DNF(Did Not Finish)

* <http://monetdb.cwi.nl/XQuery/Overview/Benchmark/XMark/>

表 1 実験環境	
CPU	Intel Pentium D 2.8GHz
OS	Windows XP
メモリ	2GB (うち FREE 領域:1.4GB)
Java	Sun JDK 1.5.0.06
JVM option	-Xmx1g -Xms1g -Xss2m

は長時間問合せ結果が返ってこないために計測を中止した項目である。DTMs は二次記憶上に DTM を格納し、問合せを実行した場合のテスト項目、DTM は一次記憶上に DTM を構築し、問合せを実行した場合のテスト項目である。DTM の計測結果には、対象 XML データのパーズ時間と DTM の構築時間を含む。尚、1.11GB の XML データをディスク上に格納した際の実際のデータサイズは、DTMs ファイルが 245MB、StringChunk ファイルが 259MB、QNameTable ファイルが 3KB である。113MB の場合は、それぞれ 24MB、44.8MB、3KB である。

実験結果の考察

計測結果からは、Q7、Q11-Q12 の場合を除いては、ディスクからの DTM 断片の読み込み負荷による性能劣化が極端に発生していないことがわかる。

113MB のテストの Q15、Q16 に代表される一部の問合せにおいて、DTMs が DTM の処理性能を上回るという結果が出ている。これは必要とされる DTM ブロックの数が少ない、あるいは局所性が高いことが理由に挙げられる。Q15 及び Q16 に現れる Axis 軸の評価は *child ::* と *attribute ::* のみであり、これらは局所性の高い問合せであると考えられる。また、DTMs が予めシステムに XML データを格納済みであることに對して、DTM の場合は XML から一次記憶上に DTM を構築する為、この準備に時間がかかっている事が影響している。

Q7 は図 5 をみてきたように、DTMs のページを多くのページを順に要求する問合せである為、ページングコストが計測結果に現れている。このような // 問合せについては、索引の利用が欠かせない。また、Q11-Q12 の問合せが DNF となったのは、我々のシステムにおいて、Join の条件の抽出が充分ではなかったことが影響している。Q11-Q12 における Join を抽出できなかった事で計算量が低減されず、その計算量の大きさがページ読み込みに影響することで性能が低下していると考えられる。

表 2 が、Q11-Q12 に共通する Join を含む問合せ部分である。Join する際のキー^(a)が text 型であるのに対して、計算済みの部分^(b)は数値型であることが、Join 抽出における課題となっている。Q8 や Q9

表 2 Q11、Q12 より抜粋した問合せ	
for \$p in \$auction/site/people/person	
let \$l :=	
for \$i in \$auction/site/open_auctions/open_auction/initial	
where \$p/profile/@income ^(a) >	
5000 * exactly-one(\$i/text()) ^(b) ..	

は Join を含む問合せであるが、実際に Join を抽出し計算量を落としている為、性能の低下が起こっていない。Q11-Q12 についても Join を有効とする為に、型推論をおこない静的に型解決を行うことでこの問題を解決する必要がある。

XMark 1.11GB のケースにおいては、Q6、Q7、Q14、Q19 以外の問合せについては、データサイズが 10 倍となったことで性能の低下が顕著になったという事実は発生していない。これら Q6、Q7、Q14、Q19 の問合せについては、すべて // を含む問合せであったことが、読み込みページ数の増加とバッファ管理の効率に影響を与え、性能を低下させていると見られる。本稿の提案趣旨は、XML 物理格納手法に焦点を当てたものであり、現在索引を利用していないが、今後、索引を併用する手法を検討していく必要がある。一方で、child 軸アクセスに代表される局所性の高い問合せについては、性能の低下が起きていないことから、少なくとも 1GB クラスの XML データ処理においては索引を利用できない場合についても物理的に近接配置を行うことやページ管理を行うことで、データサイズに対して線形以上の性能向上が得られる場合があることがわかった。

6. ま と め

本論文では、DTM という論理表に基づく XML データ格納手法を提案した。実験により我々の初期実装において、1GB クラスの大規模 XML データ処理についても既存手法に対抗しうる実行性能を示した。局所性が高い問合せについて、物理的に近接配置し効率的にデータアクセスを行うことで性能に深刻な影響を与えずに実行できることが分かった。一方で、// のような局所性が低い問合せについては、逆経路索引¹⁸⁾等の既存手法の適用に課題を残した。また、バッファ管理手法とページングサイズについてはいくつか示唆を与えたが、実際に評価を行い最適な手法を見つけるのは今後の課題である。今後、チューニングを行うと共に検証を重ね、更新処理やテラバイトクラスのデータに対応するなど、手法を洗練させていく必要がある。

Query	xmark1001JIGED				xmark1013MB	
	DTMs	DTM			DTMs	DTM
Q1	9.765	-	1.3	9.9	2.210	2.643
Q2	11.843	-	1.8	33	2.609	2.549
Q3	11.843	-	11.5	25.1	2.629	2.656
Q4	11.782	-	4.5	16.1	3.549	2.578
Q5	8.547	-	0.8	20.7	2.375	2.5
Q6	47.464	-	0.01	176.1	5.547	2.547
Q7	66.454	-	0.1	276.4	7.644	2.703
Q8	24.172	-	9.6	49.1	3.656	2.765
Q9	8.547	-	11.8	61.6	2.375	2.765
Q10	8.703	-	62.8	33.44	2.718	
Q11	0.55	-	387.7	5.7	87.125	
Q12	0.55	-	121.1	5.7	20.625	
Q13	17.39	-	0.9	12.9	2.141	2.549
Q14	98.64	-	0.75	110.2	9.39	2.978
Q15	18.313	-	0.4	106	1.460	2.5
Q16	9.908	-	0.5	109	1.696	2.547
Q17	1.281	-	1.4	11.8	2.344	2.672
Q18	12.829	-	0.5	14.8	2.235	2.641
Q19	35.156	-	7	254.5	5.198	2.813
Q20	12.203	-	7	24.8	2.64	2.75

図 7 XMark ベンチマーク結果

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) の支援による。また、情報処理振興機構 (IPA) 平成 17 年度下期 未踏ソフトウェア創造事業の支援による。ここに記して謝意を表す。

参 考 文 献

- 1) OASIS Open Document Format for Office Applications (OpenDocument)
<http://www.oasis-open.org/committees/office/>
- 2) 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 年 1 月 17 日, 18 日.
- 3) 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 年 09 月.
- 4) XQuery 1.0: An XML Query Language
<http://www.w3.org/TR/xquery/>
- 5) XML Path Language (XPath) Version 1.0
<http://www.w3.org/TR/xpath>
- 6) System RX: One Part Relational, One Part XML, Kevin Beyer (IBM Almaden), SIGMOD 2005, pp. 347-358
- 7) N. Zhang, V. Kacholia, and M. T. Ozsu, "A Succinct Physical Storage Scheme for Efficient Evaluation of Path Queries in XML" In Proc. 20th International Conference on Data Engineering, Boston, MA, March 2004, pages 56-65.
- 8) Carl-Christian Kanne, Guido Moerkotte, "Effi-

cient Storage of XML Data," icde, p. 198, 16th International Conference on Data Engineering (ICDE'00), 2000.

- 9) Xiaofeng Meng, Daofeng Luo, Mong-Li Lee, Jing An: OrientStore: A Schema Based Native XML Storage System. VLDB 2003: 1057-1060
- 10) The Apache Xalan Project.
<http://xml.apache.org/xalan-j/>
- 11) dbXML <http://www.dbxmlgroup.com/>
- 12) Saxonica Saxon <http://www.saxonica.com/>
- 13) Markus L. Noga, Steffen Schott, Welf Lo"we, Lazy XML Processing, ACM DocEng'02, ACM Press, Nov 2002.
- 14) XML Query Use Cases
<http://www.w3.org/TR/xquery-use-cases/>
- 15) H. V. Jagadish, Shurug Al-Khalifa, Adriane Chapman, Laks V.S. Lakshmanan, Andrew Nierman, Stelios Paparizos, Jignesh M. Patel, Divesh Srivastava, Nuwee Wiwatwattana, Yuqing Wu and Cong Yu. TIMBER: A Native XML Database. The VLDB Journal, Volume 11 Issue 4 (2002) pp 274-291
- 16) Apache Xindice <http://xml.apache.org/xindice/>
- 17) 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, Amsterdam, The Netherlands, April 2001.
- 18) Shankar Pal, Istvan Cseri, Gideon Schaller, Oliver Seeliger, Leo Giakoumakis, Vasili Vasili Zolotov: Indexing XML Data Stored in a Relational Database. VLDB 2004.
- 19) Pathfinder: Relational XQuery Over Multi-Gigabyte XML Inputs In Interactive Time. Peter Boncz, Torsten Grust, Stefan Manegold, Jan Rittinger, and Jens Teubner. Technical Report INS-E0503. CWI, Amsterdam, March 2005.
- 20) Structural Joins: A Primitive for Efficient XML Query Pattern Matching, Shurug Al-Khalifa, H. V. Jagadish, Nick Koudas, Jignesh M. Patel, Divesh Srivastava, and Yuqing Wu, ICDE 2002.
- 21) Tian, F., DeWitt, D. J., Chen, J., and Zhang, C. 2002. The design and performance evaluation of alternative XML storage strategies. SIGMOD Rec. 31, 1 (Mar. 2002), 5-10.
- 22) MonetDB/XQuery: A Fast XQuery Processor Powered by a Relational Engine. Peter Boncz, Torsten Grust, Maurice van Keulen, Stefan Manegold, Jan Rittinger, and Jens Teubner. Proceedings of the 2006 SIGMOD Int'l Conference on Management of Data, Chicago, IL, USA, June 2006.
- 23) X-Hive XDB <http://www.x-hive.com/>
- 24) BDB XML <http://www.sleepycat.com/products/bdbxml.html>