DEIM Forum 2009 C7-4

XML 文書に対する索引分散管理システムにおける索引分割手法

小野光太郎† 樋口 健† 都司 達夫†

† 福井大学工学研究科 〒 910–8507 福井県福井市文京 3–9–1
E-mail: †{ono,higuchi,tsuji}@pear.fuis.fukui-u.ac.jp

あらまし XML 文書に対する索引分散管理システムは，XML 文書中の要素間の親子関係に対する索引である PBT (Parent B$^+$Tree) と CBT (Child B$^+$Tree) を非共有メモリ型並列計算機上で分散管理することで，効率的な検索と低コストな更新の両立を目指すものである．同システムの性能は索引の分割，割当方法（索引分割手法）に大きく左右されるが，既存の索引分割手法は XML 文書の構造を考慮したものではない．そのため，XML 文書の検索を考慮したクラスタリングスキームを索引分割手法に適用することで検索効率の向上が期待できる．本研究では 1-Index などの構造索引で用いられるノード間の後方双模倣性を新たな索引分割手法の提案とその有効性の検証を行う．
キーワード XML 文書検索，索引分散管理システム，並列処理

An Index Partitioning Scheme for the Distributed Index System for XML Documents

Kotaro ONO†, Ken HIGUCHI†, and Tatsuo TSUJI†

† Graduate School of Engineering, University of Fukui Bunkyo 3–9–1, Fukui-city, Fukui, 910–8507 Japan
E-mail: †{ono,higuchi,tsuji}@pear.fuis.fukui-u.ac.jp

Abstract The distributed index system for XML documents constructs two kinds of indexes, PBT (Parent B$^+$Tree) and CBT (Child B$^+$Tree). They are divided and managed on shared-nothing parallel computer. By using the distributed indexes and processing queries in parallel, we will expect good performance for search, with keeping low update cost. In the system, the index partitioning scheme is important for efficient search, because the performance of the system is changed by the index partition. However, existing index partitioning schemes does not regards the structure of XML documents, so we will expect that a index partitioning scheme which regards the structure of XML documents gives more speed up for search. In this paper, we propose a index partitioning scheme based on the backward-bisimilarity, and we will examine the effectiveness of this scheme.
Key words XML document search, distributed index system, parallel processing

1. はじめに

今日，XML [1] 文書によるネットワークを経由したデータの交換や共有は一般的なものとなっており，それらを高速に検索することへの要求は大きい．この要求を満たすため，多くの XMLDB は XML 文書に対する索引付けをサポートしている．

これまでに多くの索引付け手法が提案されているが，古典的なものとして XML 文書中の要素間の親子関係に対する索引付けがある．これは OODB におけるマルチインデックス手法に相当し，更新は高速だが検索は低速である．これに対して 1-Index [4] などの構造索引は高速な検索が可能だが，XML 文書が更新されたときに大規模な更新を要する場合がある．

XML 文書に対する索引分散管理システム [3] は，要素間の親子関係に対する索引である PBT（Parent B$^+$Tree）と CBT（Child B$^+$Tree）を非共有メモリ型並列計算機上で分散管理することで，更新コストを低く保ったまま検索効率の向上を目指すものである．同システムにおいては索引の分割方法が検索効率に大きく影響するが，これまで主に用いられてきたハッシュに基づく索引分割手法や横分割法は XML 文書の構造を考慮したものではない．そのため，XML 文書の構造を考慮したクラスタリングスキームを索引分割手法に適用することで検索効率の向上が期待できる．

1-Index などの構造索引で用いられる XML ノード間の後方双模倣性に基づく分割は，同索引の検索性能の高さが示すように，XML 文書の検索に適したクラスタリングスキームといえる．本研究では XML ノード間の後方双模倣性に基づく新たな索引分割手法の提案とその有効性の検証を目的とする．

2. XPath

XPath（XML Path Language）[2] は XML 文書中の位置の指定を主な目的とする言語であり，XML 文書に対する簡易な問い合わせ言語として広く普及している．本節では XPath の最も一般的な式であるロケーションパスと軸（axis）について簡単に説明する．

2.1 ロケーションパス

ロケーションパスはその名が示すとおり，XML 文書中の位置を指定する式である．ロケーションパスはひとつ以上のロケーションステップを「/」で区切ったものである（省略構文を用いて記述した場合は一部例外がある）．ロケーションステップは軸，ノードテスト，述語で構成され，このうち述語は省略できる．ロケーションパスの先頭に「/」がある場合には絶対パス，なければカレントノードからの相対パスとなる．

2.2 軸

軸は XML 文書の木構造における方向を表すものであり，以下の 13 種類がある．

（1） child
（2） descendant
（3） parent
（4） ancestor
（5） following-sibling
（6） preceding-sibling
（7） following
（8） preceding
（9） attribute
（10） namespace
（11） self
（12） descendant-or-self
（13） ancestor-or-self

後述する XML 文書に対する索引分散管理システムではこれらのうち（1）から（8）までを用いる．

3. 親子関係に対する索引

本節では親子関係に対する索引を用いた XML 文書の検索方法について説明する．

3.1 親子関係に対する索引の概要

本研究で用いる親子関係に対する索引について説明する．

親子関係に対する索引では XML 文書中の要素に対して一意に定まる EID（Element ID）を割り当て，それらをキー及びデータとして用いる．EID は TID（Tag ID，タグに対して一意に定まる）と IID（Instance ID，タグが同じ要素に対して一意に定まる）で構成され，EID から要素のタグに関する情報を得ることができる．

作成する索引は，子要素ノードから親要素ノードへの参照を保持する PBT（Parent B^+Tree）と，親要素ノードから子要素ノードへの参照を保持する CBT（Child B^+Tree）のふたつである．このうち CBT はデータとして子要素ノードの EID の他に子要素の NO（兄弟要素の何番目かを表す番号）を保持し，これを元に検索結果を文書順にソートすることができる [3]．

3.2 親子関係に対する索引を用いた軸の検索

親子関係に対する索引を用いて XPath の軸に基づく検索を行う方法について説明する．

（1） child
CBT による検索で子要素ノードの EID 集合を得る．

（2） descendant
CBT による検索を再帰的に繰り返すことで子孫要素ノードの EID 集合を得る．

（3） parent
PBT による検索で親要素ノードの EID を得る．

（4） ancestor
PBT による検索を再帰的に繰り返すことで祖先要素ノードの EID 集合を得る．

（5） following-sibling
以下の手順で後方兄弟要素ノードの EID 集合を得る．
（a） PBT による検索で親要素ノードの EID を得る．
（b） 親要素ノードの EID を用いた CBT による検索で（検索キー自身を含む）兄弟要素ノードの EID 集合を得る．
（c） 兄弟要素ノードの EID のうち NO が検索キーの NO より大きいものを後方兄弟要素ノードの EID とする．

（6） preceding-sibling
以下の手順で前方兄弟要素ノードの EID 集合を得る．
（a） PBT による検索で親要素ノードの EID を得る．
（b） 親要素ノードの EID を用いた CBT による検索で（検索キー自身を含む）兄弟要素ノードの EID 集合を得る．
（c） 兄弟要素ノードの EID のうち NO が検索キーの NO より小さいものを前方兄弟要素ノードの EID とする．

（7） following
以下の手順で後方要素ノードの EID を得る．
（a） ancestor 軸の検索で祖先要素ノードの EID 集合を得る．
（b） 祖先要素ノードの EID 集合を用いて following-sibling 軸の検索を行う．
（c） following-sibling 軸の検索結果を用いた descendant 軸の検索で後方要素ノードの EID 集合を得る．

（8） preceding
以下の手順で前方要素ノードの EID を得る．
（a） ancestor 軸の検索で祖先要素ノードの EID 集合を得る．
（b） 祖先要素ノードの EID 集合を用いて preceding-sibling 軸の検索を行う．
（c） preceding-sibling 軸の検索結果を用いた descendant 軸の検索で前方要素ノードの EID 集合を得る．

4. XML 文書に対する索引分散管理システム

本節では XML 文書に対する索引分散管理システム (以下，同システム) について説明する．同システムの概略図を図 1 に示す．

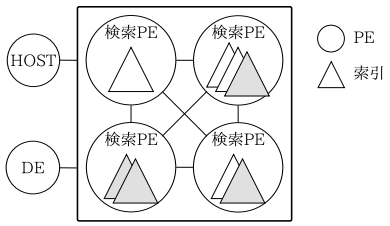

図 1　XML 文書に対する索引分散管理システム

同システムは複数の Processor Element（以下，PE）で構成されている．各 PE は資源を共有しない独立した計算機であり，相互に通信が可能であるという状況を想定する．また，各 PE での処理は単一プロセスによる逐次処理とする．

4.1 PE

PE には HOST，検索 PE，DETECTOR の 3 種類がある．以下でそれぞれの概要を説明する．

- HOST

  HOST は同システムの外部インターフェースなどの役割をもつ．主に行う処理として以下が挙げられる．
  - 外部から処理要求を受け付け RID（Request ID）を割り当て，検索 PE に処理要求，DETECTOR に送受信数通知を送信する．
  - 検索 PE から検索結果を受け取り，集計処理を行う．
  - DETECTOR から検索結果の送信数を受け取り，受信数と比較して最終的な終了判定を行う．
- 検索 PE

  検索 PE は実際に索引を格納し，検索，更新処理などを行う．検索 PE で行う 1 回の検索が検索処理の 1 step に相当する．
- DETECTOR

  DETECTOR は検索処理の各 step の終了判定と HOST で行う最終的な終了判定の補助などを行う．DETECTOR は各 RID の各 step について以下を保持する．
  - 受信予定数
  - 受信数
  - （検索 PE への）検索処理要求の送信数
  - （HOST への）検索結果の送信数

4.2 メッセージ

同システムの検索処理では以下のメッセージが PE 間で送受信される．

- 検索処理要求

  HOST から検索 PE，または検索 PE から検索 PE に送信される．少なくとも RID、step、検索命令（CBT 検索，PBT 検索，CBT&PBT 検索のいずれか）、軸、EID、1 step 前の検索で用いた EID（following-sibling 軸検索などで用いる）が含まれる。
- 送受信数通知

  HOST から DETECTOR，または検索 PE から DETECTOR に送信される．少なくとも RID、step、検索処理要求の受信数、検索処理要求の送信数、検索結果の送信数が含まれる．
- 検索結果

  検索 PE から HOST に送信される．少なくとも RID、検索結果の EID 集合が含まれる．
- 検索結果の送信数

  DETECTOR から HOST に送信される．少なくとも RID、検索結果の送信数が含まれる．

4.3 検索処理

同システムにおける検索処理の概要を説明する．

（1） HOST が以下の処理を行う．
- （a） 外部から検索処理要求を受け取り，RID を割り当てる．
- （b） 対応する検索 PE に検索処理要求を送信する．
- （c） 対応する DETECTOR に送受信数通知（検索処理要求の送信数）を送信する．

（2） 各 PE が以下の処理を繰り返す．

- 検索 PE
  - （a） 検索処理要求を受信する．
  - （b） 検索処理を行う．
  - （c） 検索処理が続行されるならば対応する検索 PE に検索処理要求を送信する．
  - （d） 検索結果があれば HOST に検索結果を送信する．
  - （e） 対応する DETECTOR に送受信数通知（検索処理要求の送受信数と検索結果の送信数）を送信する．
- DETECTOR
  - （a） 送受信数通知を受信する．
  - （b） 受信した送受信数通知の RID，step に対応する保持情報に，送受信通知の各種送受信数を反映する．
  - （c） 受信した送受信数通知の RID，step に対応する保持情報において，受信予定数と受信数が等しいならば以下の処理を行う．
    - 検索処理要求の送信数が 0 でなければ，それを次の step の受信予定数に設定する．
    - 検索処理要求の送信数が 0 であれば，検索結果の送信数（全 step の合計値）を HOST に送信する．
- HOST

  受信したメッセージに対応した処理を行う．
  - 検索結果の送信数を受信したならばそれを保持する．
  - 検索結果を受信したならば以下の処理を行う．
    - ＊ 検索結果の送信数をまだ受け取っていない，または検索結果の送信数と受信数が等しくないならば検索結果を保持する．
    - ＊ 検索結果の送信数と受信数が等しいならばその検索を終了する．

5. XML 文書に対する索引分散管理システムにおける索引分割手法

前節で述べた XML 文書に対する索引分散管理システムは，分割された索引が各検索 PE に配置されていることを必要とする．また，処理要求の送信先となる検索 PE を決定する方法も必要としている．本節では，これらの方法を定める XML 文書に対する索引分散管理システムにおける索引分割手法について説明する．

5.1 索引分割手法の概要

XML 文書に対する索引分散管理システムにおける索引分割手法（以下，索引分割手法）の概要を図 2 に示す．

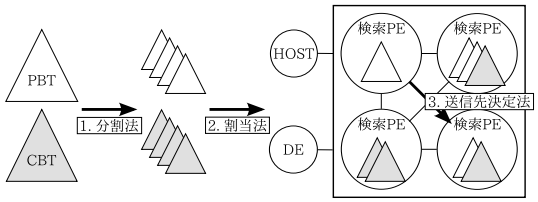

図 2 XML 文書に対する索引分散管理システムにおける索引分割手法

図 2 に示すように，索引分割手法は以下の 3 つの要素で構成されている．

（1） 分割法
　索引（PBT と CBT）を分割する方法．

（2） 割当法
　分割した索引を検索 PE に割り当てる方法．

（3） 送信先決定法
　処理要求の送信先となる検索 PE を決定する方法．

次小節以降で各索引分割手法について説明する．

5.2 既存の索引分割手法

5.2.1 ハッシュに基づく索引分割手法

ハッシュに基づく索引分割手法は EID のハッシュ値を用いて分割，割当，送信先決定を行うものである．

- 分割法・割当法・送信先決定法
  EID のハッシュ値に基づき分割，割当，送信先決定を行う．

5.2.2 横 分 割 法

横分割法は XML 文書の木構造においてレベルごとに分割を行い，ひとつのレベルにひとつの検索 PE を対応させるものである．

- 分割法・割当法
  要素ノードのレベルごとに分割，割当を行う．
- 送信先決定法
  直前の検索結果のレベルと要求された検索の種類（PBT 検索/CBT 検索）から取得できる．

5.2.3 横分割/ハッシュによる索引分割手法

前述した横分割法では，ひとつのレベル（に対応する索引）はひとつの検索 PE が管理することになる．そのため，ひとつのレベルに大量の要素ノードが存在する場合，そのレベルを管理する検索 PE に負荷が偏り，検索効率の低下や，極端な場合には処理の失敗を招く可能性がある．

これを回避するため，一定以上の要素をもつレベルについてはハッシュに基づく索引分割手法を用いて管理することが考えられる．これが横分割/ ハッシュによる索引分割手法である．

5.3 提案する索引分割手法

5.3.1 後方双模倣性に基づく索引分割手法

後方双模倣性に基づく索引分割手法は 1-Index（後方双模倣性に基づく構造索引）の索引ノードに対応する形で分割を行うものである．

- 分割法
  XML データに対する 1-Index を作成し，1-Index の索引ノードに対応する形で PBT，CBT を分割する（図 3）．

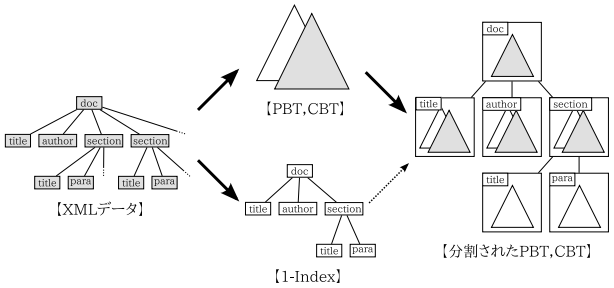

図 3 後方双模倣性に基づく分割法

- 割当法
  1-Index の索引ノードは大抵の場合検索 PE より多くなるため，各検索 PE には複数の索引が割り当てられる．同じ検索 PE に割り当てる索引の選び方は多岐に及ぶが，本研究では索引ノード ID のハッシュ値が等しいものを同じ検索 PE に割り当てるとする．
- 送信先決定法
  前提として，HOST が完全な 1-Index，各検索 PE が部分的な 1-Index（自分に対応する索引ノードとその親，子ノード）をもつものとする．これらの 1-Index の索引ノードは XML ノードへの参照ではなく対応する検索 PE の ID をもつ．以下のように送信先の検索 PE を決定できる．
  - PBT 検索の場合
    現在の索引ノードの親ノードに対応する検索 PE
  - CBT 検索の場合
    現在の索引ノードの子ノードのうち，検索結果のノードとタグが一致する索引ノードに対応する検索 PE

予想される利点としては，少数の索引からまとめて検索結果が得られるため，無駄な通信や検索処理が抑えられ，検索効率の向上が見込める．一方で更新時に 1-Index の構造が変化し，直接更新の対象となっていない索引についても大幅な変更を要する場合があり，更新コストの面では不利になると予想される．

検索結果が文書順にソートされている必要がない場合には，HOST が保持する 1-Index を検索に用いることができる．この場合検索処理の手順は以下のようになる．

（1） HOST が 1-Index を用いて検索を行い，検索結果の索引ノード ID 集合を対応する検索 PE に送信する．

（2） 検索 PE が索引ノード ID 集合を受信し，それらに対応する PBT に格納されている全てのキーを検索結果として HOST に送信するか，または検索を続行する．

5.3.2 後方双模倣性/ハッシュに基づく索引分割手法

前述した後方双模倣性に基づく索引分割手法では，ひとつの索引ノード（に対応する XML 要素ノード）はひとつの検索 PE が管理することになる．このため横分割法と同様に，ひとつの索引ノードに大量の XML 要素ノードが対応する場合，その索引ノードを管理する検索 PE に負荷が偏り，検索効率の低下や処理の失敗を招く可能性がある．

これを回避するため，一定以上の XML ノードをもつ索引ノードについては，対応する XML 要素ノードをハッシュに基づく分割手法で管理するものが後方双模倣性/ハッシュに基づく索引分割手法である．

6. 評 価 実 験

前節で説明した各索引分割手法に基づき検索処理時間の比較実験を行った．

6.1 実 験 条 件

実験条件について説明する．

6.1.1 実 験 環 境

実験環境として表 1 に示す PC18 台で構成された PC クラスタを用いた．

表 1 PC クラスタノード

HP dc5000 SF			
CPU	Pentium4	ディスク容量	40 GB
クロック速度	2.8 GHz	OS	Fedora Core 3
メモリ容量	1 GB		
通信速度	1 Gbps（実測値 638.36 Mbps）		

PE の構成は HOST 1 台，検索 PE 12 台，DETECTOR 5 台とした．

6.1.2 処 理 対 象

処理対象のファイルとして，XMark プロジェクトで公開されている XML 文書ファイルの生成ツール xmlgen により scaling factor を 1 として生成したファイルを用いた．生成したファイルの情報を表 2 に示す．

表 2 処理対象ファイル

xmlgen 生成ファイル（scaling factor : 1）			
要素数	1,666,315	タグ数	74
最大ファンアウト	25,500	深さ	12
ファイルサイズ	116,517,344 bytes		

6.1.3 クエリセット

a） QS1

child 軸の検索性能評価用クエリセットである．以下を 10 回繰り返し，合計 100 件のクエリを送出する．2 と 7，3 と 8，4 と 9，5 と 10 はそれぞれ同じクエリである．

```
1.  /site/people/person/profile/interest
2.  /site/closed_auctions/closed_auction/\
      annotation/description/parlist/\
      listitem/parlist/listitem/text/emph
3.  /site/regions/australia/item/name
4.  /site/categories/category/name
5.  /site/catgraph/edge
6.  /site/open_auctions/open_auction/bidder/\
      increase
7.  /site/closed_auctions/closed_auction/\
      annotation/description/parlist/\
      listitem/parlist/listitem/text/emph
8.  /site/regions/australia/item/name
9.  /site/categories/category/name
10. /site/catgraph/edge
```

b） QS2

descendant 軸の検索性能評価用クエリセットである．以下の 5 件のクエリを送出する．

```
1.  /site/regions//item
2.  /site/categories//name
3.  /site/people//name
4.  /site/open_auctions//seller
5.  /site/closed_auctions//seller
```

c） QS3

parent 軸の検索性能評価用クエリセットである．以下を 20 回繰り返し，合計 100 件のクエリを送出する．

```
1.  (/site/regions/africa/item/mailbox/mail/\
      from)[1] をカレントノードとして
    ../../../../../..
2.  (/site/categories/category/name)[1]
    をカレントノードとして../../..
3.  (site/people/person/address/zipcode)[1]
    をカレントノードとして../../../../..
4.  (/site/open_auctions/open_auction/bidder/\
      personref)[1] をカレントノードとして
    ../../../../../../..
5.  (/site/closed_auctions/closed_auction/\
      annotation/author)[1] をカレントノードとして
    ../../../../..
```

d） QS4

ancestor 軸の検索性能評価用クエリセットである．以下を 20 回繰り返し，合計 100 件のクエリを送出する．

```
1.  (/site/regions/africa/item/mailbox/mail/\
      from)[1] をカレントノードとして ancestor::*
2.  (/site/categories/category/name)[1]
    をカレントノードとして ancestor::*
```

```
3. (site/people/person/address/zipcode)[1]
   をカレントノードとして ancestor::*
4. (/site/open_auctions/open_auction/bidder/\
     personref)[1] をカレントノードとして
   ancestor::*
5. (/site/closed_auctions/closed_auction/\
     annotation/author)[1] をカレントノードとして
   ancestor::*
```

e） QS5

following-sibling 軸の検索性能評価用クエリセットである．以下を 20 回繰り返し，合計 100 件のクエリを送出する．カレントノードを指定する各クエリの末尾の述語で指定されている番号は，検索結果を文書順にソートしたとき中央に位置するノードを指すものである．

```
1. (/site/categories/category)[500]
   をカレントノードとして following-sibling::*
2. (/site/catgraph/edge)[500]
   をカレントノードとして following-sibling::*
3. (/site/people/person)[12750]
   をカレントノードとして following-sibling::*
4. (/site/open_auctions/open_auction)[6000]
   をカレントノードとして following-sibling::*
5. (/site/closed_auctions/closed_auction)[4875]
   をカレントノードとして following-sibling::*
```

f） QS6

preceding-sibling 軸の検索性能評価用クエリセットである．以下を 20 回繰り返し，合計 100 件のクエリを送出する．カレントノードを指定する各クエリの末尾の述語で指定されている番号は，検索結果を文書順にソートしたとき中央に位置するノードを指すものである．

```
1. (/site/categories/category)[500]
   をカレントノードとして preceding-sibling::*
2. (/site/catgraph/edge)[500]
   をカレントノードとして preceding-sibling::*
3. (/site/people/person)[12750]
   をカレントノードとして preceding-sibling::*
4. (/site/open_auctions/open_auction)[6000]
   をカレントノードとして preceding-sibling::*
5. (/site/closed_auctions/closed_auction)[4875]
   をカレントノードとして preceding-sibling::*
```

g） QS7

following 軸の検索性能評価用クエリセットである．以下の 1 件のクエリを送出する．

```
1. /site/people/following::*
```

h） QS8

preceding 軸の検索性能評価用クエリセットである．以下の 1 件のクエリを送出する．

```
1. /site/people/preceding::*
```

6.1.4 索引分割手法と検索方法

実験に用いた索引分割手法および検索方法を示す．

a） ハッシュ

ハッシュに基づく索引分割手法で分割した索引を用いて検索を行う．

b） 横分割/ハッシュ（閾値）

横分割/ハッシュに基づく索引分割手法を用いて分割した索引を用いて検索を行う．括弧内はハッシュに基づく索引分割手法に切り替える XML 要素ノード数を表す．

c） 後方双模倣性/ハッシュ（閾値）

後方双模倣性/ハッシュに基づく索引分割手法を用いて分割した索引を用いて検索を行う．括弧内はハッシュに基づく索引分割手法に切り替える XML 要素ノード数を表す．

d） 1-Index（閾値）

HOST において 1-Index を用いた検索を行う．索引は後方双模倣性/ハッシュに基づく索引分割手法を用いて分割したものを用い，括弧内はハッシュに基づく索引分割手法に切り替える XML 要素ノード数を表す．

6.2 child 軸検索の性能比較

各索引分割手法および検索方法に基づきクエリセット QS1 を実行した総検索処理時間を図 4 に示す．x 軸は索引分割手法/検索方法，y 軸は総検索処理時間を示す．

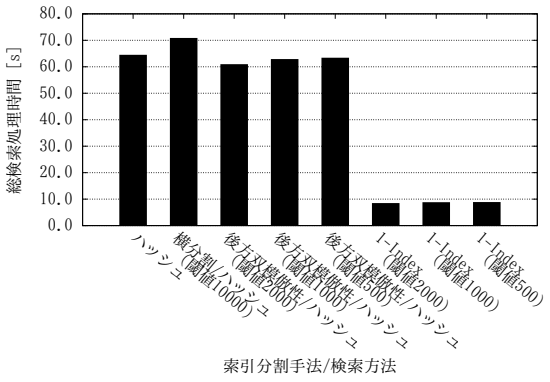

図 4 各索引分割手法に基づく QS1 の総検索処理時間

ハッシュに基づく索引分割手法は既存の索引分割手法では最も高速な結果となっている．

横分割/ハッシュに基づく索引分割手法は最も低速という結果になった．今回の実験で用いた閾値 10000 ノードにおいては，12 レベルのうち 9 レベルがハッシュに基づく索引分割手法で管理されており，より大きな閾値をとった場合にはさらに悪化することが確認されている．検索時に一部の検索 PE に長時間処理が集中することが確認されており，特定レベルの検索がボトルネックになっていると考えられる．

後方双模倣性/ハッシュに基づく索引分割手法は，閾値の異

なる 3 つ全てにおいて既存手法をわずかに上回る結果となった．また，閾値により検索性能が変化しており，今回の実験では 2000 ノードが最も高速という結果になっている．

1-Index を用いた検索は PBT，CBT を用いた通常の検索よりもはるかに高速であることが確認されたが，前述したように検索結果は文書順にソートされていないため，用途による使い分けが必要となる．

6.3 descendant 軸検索の性能比較

各索引分割手法および検索方法に基づきクエリセット QS2 を実行した総検索処理時間を図 5 に示す．x 軸は索引分割手法/検索方法，y 軸は総検索処理時間を示す．

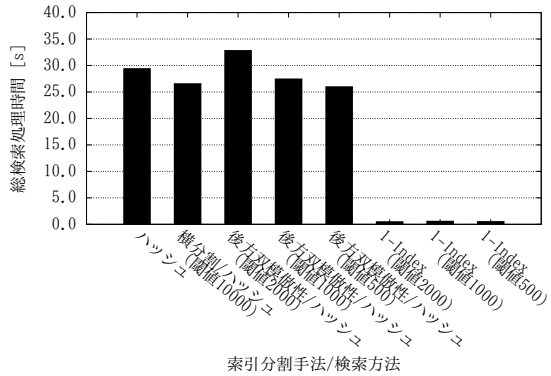

図 5　各索引分割手法に基づく QS2 の総検索処理時間

既存手法では横分割/ハッシュに基づく索引分割手法が高速な結果となった．QS2 の各クエリをそれぞれ単独で検索した場合，ハッシュに基づく索引分割手法の方が勝るものが多いことがわかっている．しかし QS2-3 と QS2-4 を同時に検索した場合，ハッシュに基づく索引分割手法は極端に検索性能が悪化することが確認されており，これが QS2 全体の検索性能の低下につながっていると考えられる．

後方双模倣性/ハッシュに基づく索引分割手法は閾値 500 のものが既存手法を上回った．閾値により検索時間が大きく変化しており，閾値 2000 のものは最も低速となっている．

1-Index を用いた検索は QS1（child 軸）と同様，通常の検索よりはるかに高速な結果となった．

6.4 parent 軸検索の性能比較

各索引分割手法および検索方法に基づきクエリセット QS3 を実行した総検索処理時間を図 6 に示す．

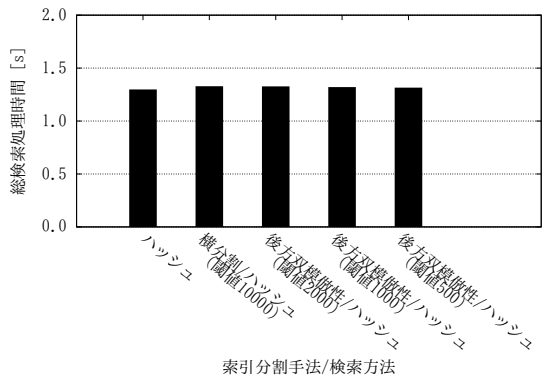

図 6　各索引分割手法に基づく QS3 の総検索処理時間

どの索引分割手法および検索方法についてもほぼ同じ検索処理時間となっている．parent 軸検索の場合，いずれの索引分割手法を用いても検索結果を 1 件得て次の検索 PE に送信するという処理の繰り返しになり，処理内容に差がほとんどないためと考えられる．

6.5 ancestor 軸検索の性能比較

各索引分割手法および検索方法に基づきクエリセット QS4 を実行した総検索処理時間を図 7 に示す．

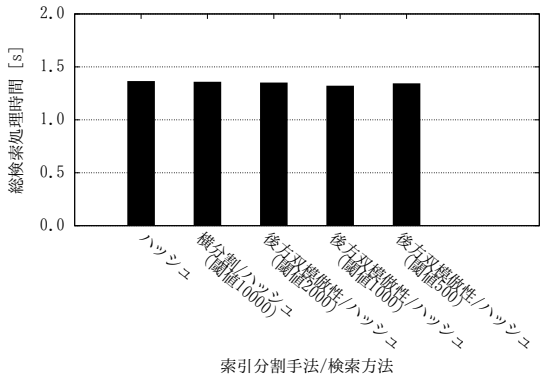

図 7　各索引分割手法に基づく QS4 の総検索処理時間

どの索引分割手法および検索方法についてもほぼ同じ検索処理時間となっている．parent 軸と同様に，いずれの索引分割手法を用いても処理内容に差がほとんどないためと考えられる．

6.6 following-sibling 軸検索の性能比較

各索引分割手法および検索方法に基づきクエリセット QS5 を実行した総検索処理時間を図 8 に示す．

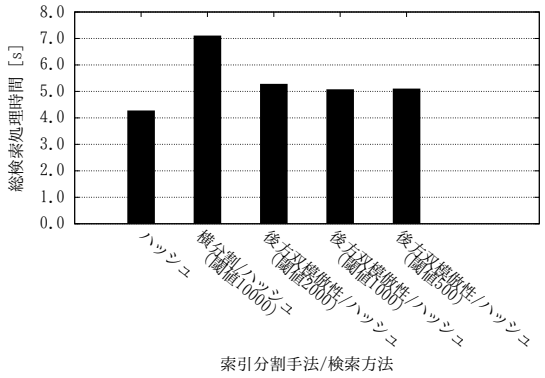

図 8　各索引分割手法に基づく QS5 の総検索処理時間

ハッシュに基づく索引分割手法が最も高速という結果になった．

横分割/ハッシュに基づく索引分割手法が低速なのは，処理が行われるレベル 1 と 2 が横分割法で管理されており，実質的にふたつの検索 PE だけで処理を行っているためと考えられる．

後方双模倣性/ハッシュに基づく索引分割手法はハッシュに基づく索引分割手法に劣る結果となった．処理が行われるレベル 1 と 2 においては両手法の索引の分割，割当状況にほぼ差がないことから，実験結果における両手法の差は送信先決定法のコスト差から生じていると考えられる．後方双模倣性に基づく索引分割手法の送信先決定法は検索方向が子方向のときにより大きなコストがかかることから，QS2（parent 軸），QS3（ancestor 軸）と異なり明確な差が生じたと考えられる．

6.7 preceding-sibling 軸検索の性能比較

各索引分割手法および検索方法に基づきクエリセット QS6 を実行した総検索処理時間を図 9 に示す.

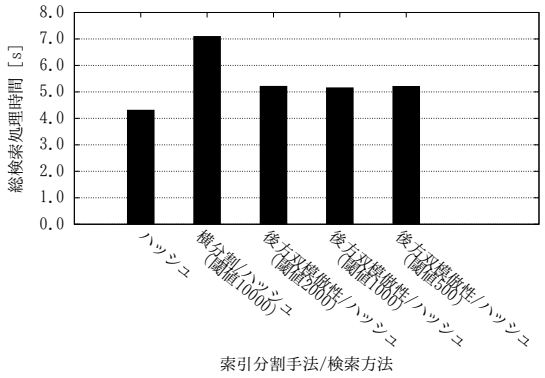

図 9　各索引分割手法に基づく QS6 の総検索処理時間

結果の傾向は following-sibling 軸と同様である．QS5 と QS6 は処理対象となる XML 要素ノードが同じであり，検索結果数もほぼ等しいため当然の結果といえる．

6.8 following 軸検索の性能比較

各索引分割手法および検索方法に基づきクエリセット QS7 を実行した総検索処理時間を図 10 に示す.

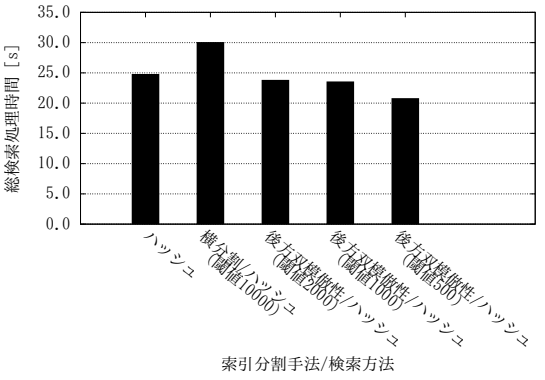

図 10　各索引分割手法に基づく QS7 の総検索処理時間

既存手法についてはハッシュに基づく索引分割手法が横分割/ハッシュに基づく索引分割手法を上回っている．

後方双模倣性/ハッシュに基づく索引分割手法は全ての閾値で既存手法を上回っている．QS7 のクエリの処理の大半は descendant 軸と同様のものであるため，この結果は妥当なものといえる．

6.9 preceding 軸検索の性能比較

各索引分割手法に基づきクエリセット QS8 を実行した総検索処理時間を図 11 に示す.

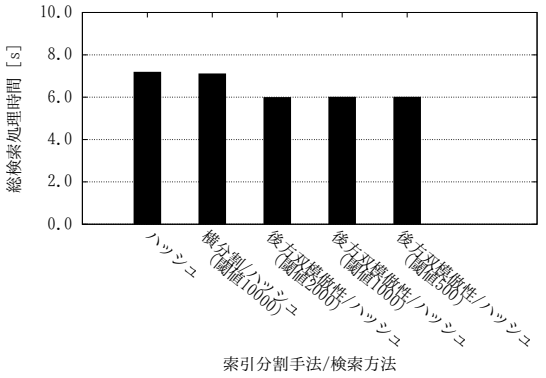

図 11　各索引分割手法に基づく QS8 の総検索処理時間

既存のふたつの手法はほぼ同じ時間となっている．

後方双模倣性/ハッシュに基づく索引分割手法はどの閾値もほぼ同じ時間となっており，全て既存手法を上回った．QS7 と同様に QS8 の処理も大半が descendant 軸検索と同様であることから，この結果は妥当といえる．

7. おわりに

本研究では XML 文書の構造を考慮した索引分割手法として、後方双模倣性に基づく索引分割手法と後方双模倣性/ハッシュに基づく索引分割手法を提案し、実装および既存手法との比較評価を行った。後方総模倣性/ハッシュに基づく索引分割手法を用いることにより，child 軸，descendant 軸，following 軸，preceding 軸における検索性能の向上が確認できた．following-sibling 軸と preceding-sibling 軸に関しては若干の性能低下を示したが，最も頻繁に使用されると予想される child 軸と，処理コストの高い descendant 軸，following 軸，preceding 軸において性能が向上したことから，同索引分割手法は検索性能の向上に有効であると考えられる。また 1-Index を利用した検索は、通常の索引分散管理システムを用いた検索と比べてはるかに高速であることが確認された。

性能低下が予想される更新についてはまだ検証を行っておらず，今後の課題となっている．

文　献

[1] Extensible Markup Language (XML), URL:http://www.w3.org/XML/

[2] XML Path Language (XPath) Version 1.0, URL:http://www.w3.org/TR/xpath

[3] 高橋 貴之，索引分散管理システムを用いた文書順を考慮した XML 文書検索，平成 18 年度福井大学院修士論文

[4] T. Milo and D. suciu, Index structures for path expressions, *ICDT*, 1999