XML 데이터의 역 인덱싱 기법

김종명°, 진 민
경남대학교 컴퓨터공학과

Inverted Indexing Method for XML Data†

JongMyoung Kim, Min Jin
Dept. of Computer Engineering, Kyungnam Univ.

요 약

관계데이터베이스를 이용한 XML 데이터저장방법에서 데이터가 삽입, 삭제, 갱신될 경우 인덱스를 재정의해야 하는 부담을 줄여주는 인덱스 기법을 제안한다. XML 데이터를 블록과 블록사이에 많아야 하나의 관계가 유지되도록 블록단위로 나누어 각 블록에 대해 Numbering 스킴을 적용하여 인덱스를 정의한다. 또한 정의된 인덱스를 이용하여 XML 질의 처리하기 *Parent-Child Block Merge Algorithm*과 *Ancestor-Descendent Block Merge Algorithm*을 제안한다.

1. 서론

XML 데이터가 갖는 데이터의 표현 능력과 교환의 용이성으로 XML 데이터의 사용은 지속적으로 증가하고 있다. 이러한 이유로 XML 데이터의 유지와 관리, 저장의 필요성이 요구되고 있다. 그러나 XML 데이터는 유연한 확장태크와 계층구조를 갖고 있어 정형화된 평면구조를 갖는 기존의 RDBMS에 저장하는 것은 쉬운 일이 아니다.

XML 데이터는 일반적으로 RDBMS에 텍스트형태로 저장되거나 관계데이터베이스의 스키마 구조로 변환되어 저장된다.

XML 데이터의 저장을 지원하는 한가지 방법으로 역 인덱스를 이용한 방법이 있다. Numbering 스킴을 이용한 일반적인 인덱스 방법은 XML 데이터의 삽입, 삭제, 갱신시에 인덱스가 재정의되어야 하므로 상당한 오버헤드를 요구한다.

† 이 연구는 정보통신부의 정보통신학술기초연구지원사업(정보통신연구진흥원) 연구비의 지원에 의한 것임.

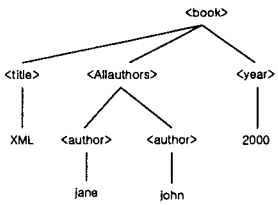
따라서 본 논문에서는 XML 데이터의 저장과 처리에 효율성을 높일 수 있는 블록단위의 역 인덱스 기법을 제안한다. XML 문서를 블록단위로 분할하고 관계데이터베이스의 스키마 구조로 변환하여 저장하는 블록분할 알고리즘과 정의한 인덱스를 이용하여 XML질의를 처리하는 합병조인 알고리즘인 *Parent-Child Block Merge Algorithm*과 *Ancestor-Descendant Block Merge Algorithm*을 제안한다. 블록분할 알고리즘으로 정의된 인덱스는 다른 XML 저장시스템보다 데이터 갱신에 따른 인덱스 데이터의 재정의에서 효율적이며 계층적인 구조 정보를 유지한다.

2. 관련연구

2.1 XML 데이터와 XML 질의

XML 데이터는 계층적인 구조를 가지므로 그림 1과 같이 트리구조로 표현될 수 있다. 이러한 트리 구조를 살펴보면 XML 데이터의 계층구조는 4가지의 특징으로 구분된다. 첫 번째로 Containment Property는 두 데이터의 관계가 Ancestor-Descendant인 경우를 의미한다. 두 번째로 Direct Containment

Property는 두 데이터의 관계가 Parent-Child일 경우를 의미한다. 세 번째로 Tight Containment Property는 두 데이터의 관계가 Direct Containment Property이면서 Descendant가 오직 하나만 있을 경우이다. 즉 Parent-Child관계이면서 하나의 Child를 가진 것을 의미한다. 네번째로 Proximity property는 두 데이터사이의 거리가 일정한 범위내에 있는 것을 의미한다.



[그림 1] XML 문서의 트리구조

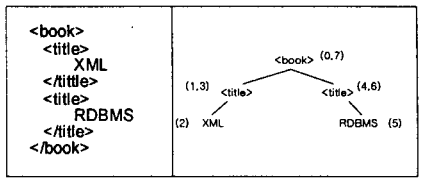
XML 질의는 일반적으로 XML데이터의 계층적인 구조에 기반을 두고 있는데 계층적인 구조는 크게 Parent-Child와 Ancestor-Descendant의 두 가지 관계로 표현된다. XML 질의의 한 종류인 Xpath는 XML 데이터 검색을 위해서 다음과 같은 형태를 가진다.[6, 10].

<book>/<Allauthors>//<author>

이 질의는 book 엘리먼트의 Child인 <Allauthors>와 Descendant의 관계인 <author>를 검색하는 것이며, <book>/<Allauthors>와 <Allauthors>//<author>을 검색하고 이것을 다시 조인하는 것과 같다. Xpath로 표현된 XML 질의에서 알 수 있듯이 XML 데이터의 계층적인 구조를 빠르게 검색하는 것이 XML 질의 처리에 있어 주요관건이 된다.

2.2 역 인덱스 기법과 XML 질의 처리

깊이우선순회의 Numbering 스킴[1, 3, 6, 8, 9]를 이용한 역 인덱스 방법은 XML 데이터의 계층구조와 질의 처리에 상당한 효율성을 제공한다. Numbering 스킴은 그림 2와 같이 숫자를 순차적으로 증가시키면서 계산하는 방법으로 XML 문서의 트리 구조에서 루트부터 시작하는 깊이우선순회방법을 사용하여 Numbering 스킴을 정의한다. 이렇게 정의된 인덱스를 가지고 서로 비교해서 데이터간의 계층구조를 검색하게된다.



[그림 2] XML 문서와 XML트리

문서에 Numbering 스킴을 사용한 인덱스 방법은 몇 가지로 구분된다. 0, 1, 2, 3, 4....같은 순차적인 숫자를 이용한 방법과 순차적이면서 개체사이에 일정한 구간을 두는 것으로 구분된다.

전자의 경우 XML 데이터를 Element와 Texts 구분 인덱스를 정의한다. ELEMENT(docno, begin:end, level)와 TEXTS(docno, wordno, level)로 구성된다[1]. 즉 문서번호, 개체의 시작 위치:끝 위치, 레벨로 표현되며 TEXTS는 시작과 끝이 동일하므로 하나로 표현된다. 합병조인알고리즘으로는 MPMGIN알고리즘[1], Tree-Merge Join 과 Stack-Tree Join 알고리즘[3]등이 있다.

후자의 경우 (order, size)의 쌍으로 구성되는데 order는 개체의 순서(DFS)를 의미하며 size는 개체의 특정한 범위를 나타낸다. order와 size의 범위는 특정한 규칙에 의해 정의된다. x가 y의 선조일 때 order(x) < order(y) ≤ order(x)+size(x)와 같은 규칙이 적용된다[6]. 합병조인알고리즘으로는 EE-Join, EA-Join, KC-Join 등이 있다.

3. XML 문서의 블록분할인덱스

3.1 XML 문서의 분할구조

기존의 Numbering 스킴를 사용한 XML 인덱스 기법은 XML 문서에 변경이 발생하게 되면 인덱스 재정의에 상당한 처리비용을 요구한다. 처리비용을 줄이기 위해서는 인덱스의 재정의를 특정한 부분으로 제한해야 한다. 이러한 문제를 해결하기 위해서 인덱스 재정의를 블록단위로 제한하는 블록분할인덱스를 제안한다.

블록분할인덱스는 XML 문서를 블록으로 나누고 블록과 블록사이는 많아야 하나의 관계를 갖도록한다. 또한 블록당 Numbering을 적용하여 인덱스를 구성하도록 한다. 그림 3은 블록분할알고리즘(Block Split Algorithm)을 나타낸것이다.

```
product BSA(XMLDocument XDoc , DesireNumber N)
begin
1 :  Locallevel Checklevel=-1;    // XML문서에서 개체 level
2 :  DesireNumber K=-1;           // N까지의 갯수를 제어
3 :  ObjectPoint LocalPoint;      // 개체의 위치변수 (시작시 처음위치를
                                     나타냄)
4 :  Objectlevel Firstlevel=0;    // Block의 처음개체 level을 저장(실제 보다
                                     1적다)
5 :  NumberingValue Number=-1;  // Block당 Numbering 하기 위한 값
6 :  BlockrObjectBalance ObjectBalanceCheck=0;  // Block당 Element 시작과
끝의 갯수 Check
7 :  BlockLevel BlockLevelNumber    // Block당 Level
8 :  LocalPoint=XDoc.FirstObject    // LocalPoint가 처음의 Object를 가려
                                     치도록 함.
9 :
10 : while (XDoc의 끝이 아니면){
11 :          new 블록의 설정;
12 :          Firstlevel=Checklevel;
13 :          BlockLevelValuel=-1;
14 :    do{
15 :      if (LocalPoint가 가리키는 Object가 시작 Element일 경우){
16 :          Checklevel++;   BlockLevelValue++;
17 :          Number++;
18 :          K++;
19 :          ObjectBalanceCheck++;
20 :          LocalPoint가 가리키는 Object를 출력(Block IDr, ObjectName,
              Checklevel(BlockLevelValuel), Number(BeginNumber));
21 :      }
22 :
23 :       if ( LocalPoint가 가리키는 Object가 Text 경유){
24 :          Checklevel++;   BlockLevelValue++;
25 :          Number++;
26 :          K++;
27 :          LocalPoint가 가리키는 Object를 출력(Block IDr, ObjectName,
              Checklevel(BlockLevelValuel), Number(BeginNumber));
28 :          Checklevel--;    BlockLevelValue--;

29 :       }
30 :
31 :       if (LocalPoint가 가리키는 Object가 Element닫는 Object){
32 :            Checklevel--;   BlockLevelValue--;
33 :            ObjectBalanceCheck--;
34 :            Number++;
35 :          if (K≠0) and (ObjectBalanceCheck >= 0)    //Block당 태그의
                                                  시작과 끝의 Balance 체크
36 :              ObjectName이 같은 것 중에 가장 최근의 레코드부터
                  Number(EndValue)없는 레코드에 삽입;
37 :
38 :       }
39 :       LocalPoint =LocalPoint ->Next;
40 :    } while ((K < N-1) and (Checklevel > Firstlevel))
41 :
42 :          while(LocalPoint가 가리키는 Object가 Element닫는 Object)
43 :            Checklevel--;
44 :            LocalPoint =LocalPoint ->Next;
45 :          }
46 :
47 :     Element의 End Value 값이 없는 것을 찾아 나중에 생성된 것부터
        찾아  Number값 다음부터 삽입(all);
48 :     Firstlevel의 개체의 level보다 한 단계 적은 최근의 개체의 위치를
        부모로 생성한다(처음 block 제외);
49 :   K=-1;
50 :   Number=-1;
51 :   ObjectBalanceCheck=0;
52 : }
    end
```

[그림 3] Block Split Algorithm

BSA(Block Split Algorithm)의해서 생성된 인덱스는 다음과 같은 스키마 형태를 가진다. Object name은 XML 문서의 개체 이름이며, Block ID는 각 블록을 구별하기 위한 블록식별자이다.

(Object name, Block ID, level, begin:end, Parent Block, Parent begin:end)

Level은 블록에서 개체의 레벨을 말하며, Begin:end는 블록에서 각 개체의 시작위치와 끝위치를 나타낸다. Parent Block은 블록과 블록사이의 관계에서 부모블록을 의미하며, Parent begin:end은 부모블록에 직접관계 있는 개체의 시작위치와 끝위치이다.

3.2 XML 문서의 변경에 따른 블록의 구조

블록화된 XML 문서에서 삽입, 삭제, 갱신 등의 작업이 일어날 경우의 블록구조를 살펴보면 다음과 같다.

① 삽입 : 가득찬 블록과 가득찬 블록사이의 삽입시에는 새로운 블록을 생성하여 블록사이의 관계를 표현하면 된다. 삽입하고자 하는 블록사이에 가득 차지 않은 블록이 존재하면 그 블록에 데이터를 삽입하고 인덱스를 재정의 하면 된다.

② 삭제 : 삭제하고자 하는 개체를 삭제하고 그 개체의 모든 자손들을 삭제해야 한다.

③ 갱신 : 해당 개체를 갱신하면 된다.

4. 합병조인알고리즘

XML 질의는 개체의 계층구조를 빠르게 검색하는 것이 핵심이다. BSA에 의해서 정의된 인덱스를 가지고 XML 질의를 처리하기 위해서 인덱스를 E-index, T-index로 나눈다[1, 3]. E-index의 경우 Element이름이 같은 것으로 구성하여 Block ID순서의 내림차순으로 정렬한다. 그 다음은 begin의 순서로 정렬한다. Block ID을 기준으로 정렬하는 것은 합병조인알고리즘에서 Block ID을 먼저 비교하기 때문에 처리시간을 단축 시킬 수 있기 때문이다. T-index는 텍스트 데이터를 함께 저장하며 E-index와 같이 Block ID과 begin의 순서로 정렬한다.

그림 4는 Parent-Child를 검색하기 하기 위한 합병조인 알고리즘 이며, 그림 5는 Ancestor-Descendant를 검색하기 하기 위한 합병조인 알고리즘이다.

이 두 알고리즘은 블록의 관계를 먼저 계산하므로 블록에 포함된 모든 데이터의 검색 없이도 데이터의 관계를 검색할 수 있다.

5. 결론

```
Procedure PCBMA(Plist, Clist)
Begin
1 :     set P-cursor beginning of Plist1
2 :     set C-cursor beginning of Clist2
3 :
4 :     for(P-cursor=Plist->firstNode : P-cursor !=NULL :    P-cursor
        =P-cursor ->NextNode)
5 :     {
6 :        for(C-cursor=Clist->firstNode : C-cursor !=NULL :    C-
        cursor =C-cursor ->NextNode)
7 :              if (P-cursor.Block ID == C-cursor.Block ID)
8 :                  {
9 :                      if ((P-cursor.begin < C-cursor.begin) && (P-
                         cursor.end > C-cursor.end) &&
10:                          (P-cursor.level == C-cursor.level+1)
11:                          OutputList;
12:
13:                      } else if {
14:                           if(P-cursor.Block ID == C-
                             cursor.Relation Block) &&
15:                               (P-cursor.begin == C-
                             cursor.Parentbegin) &&
16:                               (P-cursor.end == C-
                             cursor.Parentend) &&
17:                               (C-cursor.level == 0)
18:                           OutputList;
19:                  }
20:
21:           C-cursor=Clist->firstNode;
22:     }
end
```

[그림 4] Parent-Child Block Merge Algorithm

```
Procedure ADBMA(Alist, Dlist)
Begin
1 : set A-cursor beginning of Alist1
2 : set D-cursor beginning of Dlist2
3 :
4 : for(A-cursor=Alist->firstNode : A-cursor !=NULL :    A-cursor =A-
        cursor ->NextNode)
5 : {
6 :   for(D-cursor=Dlist->firstNode : D-cursor !=NULL :    D-cursor =D-
        cursor ->NextNode)
7 :     if (A-cursor.Block ID == D-cursor.Block ID)
8 :       {
9 :       if ((A-cursor.begin < D-cursor.begin) && (A-cursor.end > D-
                    cursor.end)
11            OutputList;
12:
13          } else if {
14          MarkID = D-cursor.Parent Block ID;
15:         while ( (not(A-cursor.Block ID ==   D-cursor.Parent Block
                    ID) or
16:           D-cursor.Parent Block != NULL) and (mark==미확인) )
17:             {   // 선조를 계속 검색함
18:           D-cursor.Parent Block=ParentBlock(D-cursor.Parent Block)
19:           }
20:
21:           if ((A-cursor.Block ID == D-cursor.Parent Block) or (mark
                    == 존재)){
22:             OutputList;
23:             ID를 mark 한다(존재);
24:                    }
25:           else{
26            ID 를 mark 한다(존재하지 않음).;
27:              }
28:       }
29:       }
30:       D-cursor=Dlist->firstNode;
31:             모든 mark를 미확인 상태로 한다,
32:   }
end
```

[그림 5] Ancestor-Descendant Block Merge Algorithm

본 논문에서는 BSA에 의해서 XML문서를 블록 단위로 나누어 Numbering 스킴를 정의하여 인덱스를 만들었다. 이렇게 만들어진 인덱스는 XML 문서의 계층적인 구조 정보를 유지하며, XML 문서의 데이터 갱신에 따른 인덱스 재정의에 상당한 효율성을 제공한다. 또한 XML 질의를 처리를 위해서 Parent-Child Block Merge Algorithm과 Ancestor-Descendent Block Merge Algorithm을 제안했다.

향후 연구계획으로는 좀더 나은 XML 질의 처리를 위해서 블록의 관계를 빠르게 검색할 수 있는 방법이 개발되어야 할 것이다..

[참고문헌]

[1] Chun Zhang, Jeffrey Naughton, David DeWitt, Qiong Luo, Guy Lohman, On Supporting Containment Queries in Relational Database Management Systems. In proceedings of ACM SIGMOD Conference on Management of Data 2001.

[2] D. Chamberlin, D. Florescu, J. Robie, J. Simeon, and M. Stefanescu. Xquery A query language for XML. W3C Working Draft. Available from http://www.w3.org/TR/xquery, Feb 2001.

[3] D. Srivastava, S. Al-Khalifa, H.V. Jagadish, N. Koudas, J.M. Patel, Y.Wu: Structural Joins: A Primitive for Efficient XML Query Pattern Matching. In proceedings of ICDE 2002.

[4] F. Tian, D. DeWitt, J. Chen, and C. Zhang. The Design and Performance Evaluation of Alternative XML Storage Strategies. Technical report, CS Dept., Universiy of Wisconsin, 2000.

[5] Kha, D. D., Yoshikawa, M., and Uemura, S. An XML indexing structure with relative region coordinate. In proceedings of 17th International Conference on Data Engineering April 02 - 06, 2001. pages 313-320.

[6] Q. Li, B. Moon. Indexing and querying XML data for regular path expressions. In proceedings of the 27th VLDB Conference, 361-370, 2001.

[7] Rizzolo and A. Mendelzon. Indexing XML Data with ToXin. In proceedings of Fourth International Workshop on the Web and Databases (WebDb 2001)

[8] S.-Y. Chien, V.J. Tsotras, and C. Zaniolo, "Version Management of XML Documents", In proceedings of International Workshop on the Web and Databases (WebDB 2000)

[9] Vincent Aguil'era, Sophie Cluet, Pierangelo Veltri, Dan Vodislav, and Fanny Wattez. Querying xml documents in xyleme. In to appear in the proceedings of the ACM-SIGIR 2000 Workshop on XML and Information Retrieval, Athens, Greece, july 2000.

[10] W3C Recommendation. XML Path Language (Xpath) 1.0. In http://www.w3.org/TR/xpath. 1999.