제22회 한국정보처리학회 추계학술발표대회 논문집 제11권 제2호 (2004. 11)

XML 문서의 효과적인 색인방법을 위한 Order-Array 의 사용

김영*, 안찬민*, 박상호*, 박선*, 이주홍*, 전석주**
*인하대학교 컴퓨터정보공학과
**서울교육대학교 컴퓨터교육과
e-mail:{youngjin, ahnch1, parksangho, sunpark}@datamining.inha.ac.kr
juhong@inha.ac.kr, chunsj@snue.ac.kr

An Efficient Indexing Method For XML Documents Using Order-Array

Young Kim*, Chan-Min Ahn*,Sang-Ho Park*, Sun Park*, Ju-Hong Lee*, Suk-Ju Chun**
*School of Computer Science and Engineering, Inha University
**Dept. of Computer Education, Seoul National University of Education

요 약

최근 XML 은 전자상거래에서 의학, 국방, 법률 등의 전문분야에 이르기까지 많은 분야에서 활용되고 있으며, 데이터의 양 또한 방대해지고 있다. 따라서 대량의 XML 문서들을 효과적으로 저장하고 빠르게 검색할 수 있는 많은 인덱싱 기법들이 연구되고 있다. 최근의 인덱싱 기법들 중 Numbering Scheme 을 기반으로 한 인덱싱 기법들은 대부분의 검색에 우수한 성능을 보이나 하위노드의 수가 늘어나면 검색 오버헤드가 커질 수 있으며, 대량의 XML 문서의 추가 삽입 및 구조가 다른 XML 문서의 삽입시에 인덱스와 데이터 값의 재조정에 따른 많은 비용이 발생하게 된다. 이에 우리는 Numbering Scheme 을 기반으로 하지만, 각 노드별로 노드범위(Node-Range)와 Order-Array 를 추가하여 검색성능을 향상시키고 대량의 XML 문서의 삽입 및 구조가 다른 XML 문서의 삽입시에 발생되는 문제를 해결하고자 한다.

1. 서론

최근 인터넷상의 데이터의 표현 및 교환의 표준으로 사용되고 있는 XML(eXtensible Markup Language) [1]은 전자상거래[2]에서부터 의학, 국방, 법률, 메신저 등에 이르기까지 인터넷 전 분야에 걸쳐 활용되고 있다. 특히 전문 분야의 XML 문서들은 그 크기가 방대해지고, 사용자의 질의 또한 복잡해졌다. 따라서 수많은 XML 문서를 효율적으로 저장하고, 복잡한 XML 질의[3,4]를 빠르게 처리하기 위한 많은 인덱싱 기법들이 제안되어 왔다.

최근에 제안된 인덱싱기법들 중 많은 수가 루트(Root)에서 리프(Leaf)까지의 심플패스(Simple Path)를 인코딩하여 처리하거나, 노드별로 <preorder, postorder>를 부여하여 검색하는 Numbering Scheme 을 기반으로 처리한다. 그러나 패스 기반의 인덱싱기법들은 심플패스의 질의에는 좋은 검색 성능을 보이나, 조상-후손관계의 질의나, 최하위노드의 질의시엔 성능이 떨어진다.

Numbering Scheme 기반의 인덱싱기법은 질의의 <preorder, postorder>를 분석하여 처리하며 RDBMS(B+ Tree)를 기반으로 한다. 대부분의 검색에서 좋은 성능을 보이나, XML 패스의 하위노드의 수가 많아지게 되면 인덱스의 성능이 떨어지게 된다. 또한 대량의 XML 문서가 추가로 삽입되거나, 구조가 다른 XML 문서가 삽입되어 이웃노드(Sibling-Node)나 하위노드가 추가된다면 인덱스의 재조정이 일어나며 최악의 경우엔 데이터테이블 전체의 <preorder, postorder>가 재조정될 수 있다.

이러한 문제들을 해결하기 위하여 본 논문에선 Numbering Scheme 을 기반으로 각 노드별로 노드범위(Node-Range)와 중복노드(Duplicate-Node)를 두어 XPath

질의를 빠르게 분석하며, Order-Array 를 추가하여 실제 검색할 데이터테이블의 범위를 줄인다. 또한 데이터테이블을 최하위노드별로 유지하므로 삽입, 삭제에 드는 비용을 효과적으로 줄일 수 있다.

본 논문의 구성은 다음과 같다. 2 장에서는 기존의 인덱싱 기법들을 소개하며 장.단점을 기술한다. 3 장에서는 본 논문의 인덱싱 기법을 소개한다. 3.1 은 본 논문의 기반이 되는 Numbering Scheme[5]을 소개하고, 3.2 에서는 우리가 제안한 인덱싱 기법과 검색 예제를 기술하며, 3.3 에서는 삽입, 삭제방법을 기술한다. 4 장에서는 우리의 인덱싱 기법과 기존의 인덱싱 기법과 비교실험 하며, 5 장에서 결론 및 향후 연구과제에 대해 기술한다.

2. 관 련 연 구

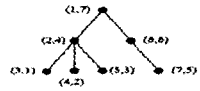
대용량의 XML 문서에서 사용자의 질의를 빠르게 검색하기 위한 많은 인덱싱 기법들이 제안되어 왔다. 대표적인 인덱싱 기법에는 Lorel's Index[6], Inverted Index 를 이용한 기법[7], XISS[8], Index Fabric[9], Stack Tree[10], BLAS[11] 등이 있다. 그 중 최근의 인덱싱 기법인 Index Fabric 은 XML 트리의 모든 노드와 데이터를 Patricia Trie[12]로 인코딩(Encoding)하여 패스 검색시의 연산을 줄였으며, 레이어(Layer)를 사용하여 트리의 바란스(Balance)를 맞추었다. 심플패스의 연산시엔 아주 뛰어난 성능을 보이나, 중간노드와 최하위노드의 검색, 조상-후손관계의 조인연산시에 매우 효율이 떨어진다. 이를 보완하기 위하여 Refined-Path 의 개념을 두어 자주 사용되는 패스를 인덱스에 추가하는 방법을 사용하였으나, DBA 가 정의해야 하므로 사용자의 다양한 질의를 처리하기엔 부적합하며 많을 경우엔 인코딩의 효과가 떨어지게 된다.

Numbering Scheme 기반의 XISS 는 데이터의 삽입시에 유연한 확장을 위하여 노드별 <preorder, postorder> 대신에 Order 와 후손들의 범위인 Size 로 레이블링(Labeling)하였다. 사용자의 XPath 질의시 해당되는 Order 와 Size 를 계산하여 RDBMS(B+Tree)에 구현된 XML 데이터를 검색한다. 심플패스, 조상-후손관계조인연산, 중간노드등의 대부분의 검색에 좋은 성능을 보이나 노드의 수가 너무 많거나 중복노드가 많을 경우엔 검색성능이 떨어질 수 있다. 또한 대용량의 XML 문서의 추가 삽입이나, 구조가 다른 XML 문서의 삽입시에 이웃노드나 하위노드가 추가될 경우 최악의 경우엔 인덱스 뿐만 아니라 데이터의 Order 와 Size 도 재조정 될 수 있다. 최근의 인덱싱 기법 BLAS 는 복잡한 질의처리의 효율을 높이기 위하여 D-Labeling 과 P-Labeling 을 사용하였다. D-Labeling 은 Numbering Scheme 과 같은 방법으로 모든 노드를 preorder 와 postorder, level 로 관리하며, P-Labeling 은 최하위노드(Suffix Node)를 효율적으로 처리하기 위하여 비율(Proportion)로 나누어 관리한다. 사용자의 복잡한 질의가 주어졌을 경우 질의를 분해하여 D-Label 과 P-Label 을 이용하여 조상-후손관계와 최하위노드를 처리하여 검색의 범위를 줄인다. 그러나, 구현이 매우 복잡하고 위의 XISS 와 같은 문제가 발생할 수 있다.

3. 인 덱 싱 기 법

3.1. Numbering Scheme

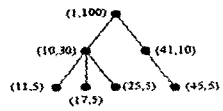
XPath 질의는 조상-후손관계의 조인연산('//')이 빈번하게 발생하게 된다. 이러한 연산을 처리하기 위해 기반이 되는 방법 중 하나가 Dietz's Numbering Scheme[5] 이다.



[그림 1] Dietz's Numbering Scheme

XML 의 모든 노드는 $<d_1, d_2, d_3>$의 레이블을 가지며, 만약 노드 m, n 이 있을 경우에 m 이 n 의 후손(Descendant)이면 $n.d_1 < m.d_1$ AND $n.d_2 > m.d_2$를 만족하고, m 이 n 의 조상(Ancestor)이면 $n.d_1 > m.d_1$ AND $n.d_2 < n.d_2$를 만족하며, m 이 n 의 자녀(Child)이면 $n.d_3+1 = m.d_3$을 만족한다. 만약 n 과 m 이 조상-후손관계가 아니라면 $n.d_2 < m.d_1$ OR $n.d_1 > m.d_2$의 중첩방지(Non-Overlap) 조건을 만족하게 된다. 즉 $<d_1, d_2, d_3>$ 은 <preorder, postorder, level> 을 나타내게 된다.

XISS 에서는 데이터의 삽입을 유연하게 하기 위하여 [그림 1]의 방법을 확장하였다.



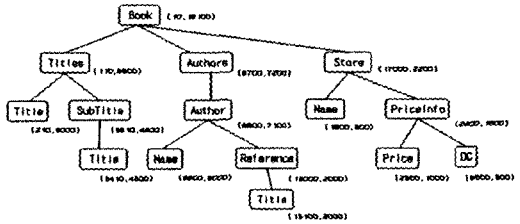
[그림 2] Numbering Scheme 의 확장 예

[그림 2]와 같이 preorder 대신 order 로, postorder 대신 Size 로 변경하였고, Size 에는 하위노드에 포함될 데이터의 수 + 확장범위가 들어가게 된다. 노드 m 과 n 이 있고 m 이 n 의 부모(Parent)라면 다음의 조건을 만족하게 된다.

Order(m) < Order(n) AND
Order(n) + Size(n) <= Order(m) + Size(m)

3.2. 인 덱 싱 제 안

우리의 인덱싱 방법은 질의의 결과로 나온 preorder 와 postorder 를 데이터인덱스(B+ Tree)에서 비교하지 말고, XPath 질의를 분석하여 나온 범위의 데이터를 바로 결과로 사용하자는 것이다.

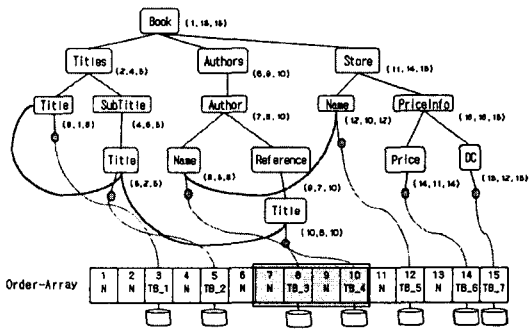


[그림 3] XISS 의 Numbering Scheme 구현 예

[그림 3]과 같이 구성된 XML 데이터가 있을 때 XPath 질의 "/Book//Author"가 주어졌을 경우 질의를 분석하여 최종적으로 Order (9800) 보다 크고, 9800 + Size (7100) 보다 같거나 작은 값을 가진 XML 데이터를 데이터테이블에서 검색하게 된다. 즉

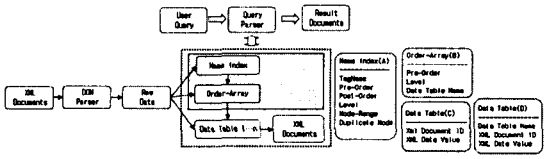
```
SELECT xmlDoc_id, xmlValues
FROM Xiss_ValueTable
WHERE Order > 9800
  AND (Order + Size <= 16900)
```

본 논문의 방법은 이러한 검색을 하지 않고, XML 데이터의 삽입시 최하위노드별로 데이터테이블을 유일하게 생성한다. 그리고 노드범위(Node-Range)와 Order-Array 를 사용하여 XPath 질의에 해당하는 데이터테이블을 검색한다.



[그림 4] Order-Array 를 사용한 예

본 논문은 "/Book//Author"의 질의의 범위로 <7,10>을 얻는다. 그리고 Order-Array 의 7 부터 10 사이에 위치한 데이터테이블을 개방함으로써 검색이 완료된다. 본 논문의 인덱싱구조는 다음 [그림 5]와 같다.



[그림 5] 인덱스의 구성

[그림 5]의 Name Index (A)는 노드별로 preorder, postorder, level, Node-Range, Duplicate-Node 등의 정보로 구성된다. XPath 질의시에 질의를 분석하여 preorder 와 Node-Range 를 얻은 후에 Order-Array 의 해당 범위로 이동하여 검색결과를 얻는다. Node-Range 의 계산방법은 다음과 같다.

```
Node : n, Order-Array 의 범위 : x
[NodeRange의 계산]
if n.pos = ROOT then
   n.noderange = n.postorder
else
   if n.rightsibling is true then
      n.noderange = n's rightsibling.preorder - 1
   else
      n.noderange = n's parent.noderange
   end if
end if
[Node-Range를 이용한 Order-Array 의 데이타테이블의 범위계산]
n.preorder <= x and x <= n.noderange and x.data_flg = 'Y'
```

Name Index (A)의 Duplicate-Node 는 조상-후손('//')관계의 연산시에 나올 수 있는 중복된 노드들을 연결하는 포인터로 중복노드가 존재할 시에는 빠르게 이동하며 조상의 preorder 와 postorder 를 비교하여 질의의 범위를 계산할 수 있다. [그림 5]의 Order-Array(B)는 모든 노드를 preorder 순으로 데이터테이블의 존재유무와 데이터테이블의 이름을 보관한다. [그림 5]의 Data Table (C)는 XML 문서의 유일한 ID 와 데이터값이 보관된다. 이와 같이 질의의 결과범위에 해당되는 데이터테이블을 Order-Array 에서 바로 검색하여 오픈하는 구조로 되어 있으며, 사용자의 값에 대한 조건연산, 예를 들어 "/Book//Author[.='Peter Norton']이라는 질의가 주어졌을 경우엔 오픈된 데이터테이블의 XML Data Value 를 비교하여 처리한다. 그러나 질의결과의 범위가 넓거나, 중복노드가 존재하여 많은 데이터테이블을 오픈 할 경우 RDBMS 에 많은 오버헤드를 초래할 수 있으므로 Data Table (D)와 같이 단일 데이터테이블에 Data-Table-Name 을 Cluster-Index 로 지정하여 처리하는 방법도 제안한다. 패스의 최하위 노드가 많을 경우엔 성능이 향상된다.

예) [그림 4]에서의 XPath 질의의 예
/Book/Titles/Title : 루트부터 Title 까지 검색한 후 범위 <3,3>을 얻고, Order-Array 의 위치 3 의 데이터테이블을 오픈한다.
/Book/Titles : 루트부터 Titles 까지 검색한 후 범위 <2,5>를 얻고, Order-Array 의 위치 2 부터 5 까지의 데이터테이블을 오픈한다.
/Book//Author : 루트부터 Author 까지 검색한 후 범위 <7,10>을 얻고, Order-Array 의 위치 7 부터 10 까지의 데이터테이블을 오픈한다.
/Book//Title[='경제학개론'] : 루트로부터 Title 까지 검색한 후 중복노드 포인터로 이동하면서 Book 의 preorder 와 postorder 에 포함되는 Title 의 범위 <3,3>, <5,5>,<10,10>을 얻은 후 Order-Array 의 3,5,10 번의 데이터테이블의 XML Data Value 와 비교하여 결과를 리턴한다.

3.3. 데이터의 삽입, 삭제

본 논문에서 제안한 방법은 데이터테이블을 Order-Array 를 통해 별도로 관리하므로, 기존의 방법에서와 같이 데이터테이블의 preorder 와 postorder 의 재조정작업이 필요가 없다.

삽입방법은 XML 문서구조에 따라 다르게 적용된다. XML 문서의 구조가 동일하면 단순히 패스를 따라 이동하여 Order-Array 에 연결되는 데이터테이블에 항목을 추가한다. 이 경우 삽입시의 preorder 와 postorder 의 재조정이 필요가 없다. XML 문서의 구조가 다르면, Name Index 와 Order-Array 를 조정한다. 그러나, 추가된 데이터테이블의 이름이 Order-Array 에 유일하게 생성되므로 별도의 오버헤드는 없다.

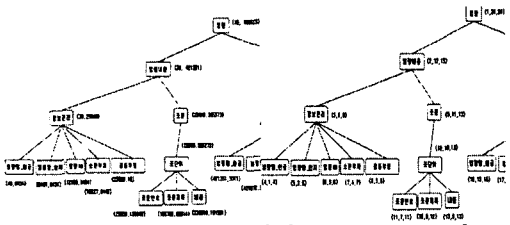
데이터의 삭제시에는 Order-Array 를 통해 데이터테이블로 이동한 후 항목을 삭제한다. 만약 데이터테이

블이 Empty 가 되면 Order-Array 의 Data_Flg 와 Data Table Name 을 지운 후 해당 데이타테이블을 삭제하고 Name Index 와 Order-Array 를 재조정 한다.

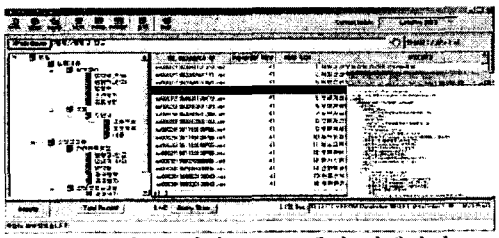
4. 성 능 실 험

본 실험은 Intel CPU 4 1.6G, OS : Windows 2000 Server, Main Memory 512M, HDD 60G 의 컴퓨터에서 하였다. RDBMS 로는 Sql*Server 2000, 언어는 Visual Basic 을 사용하여 테스트 하였다. 실험 데이타는 국내에서 사용되는 법률 XML 실데이타의 일부로서 테스트 하였으며, 총 9,495 개의 XML 문서, 498,858 개의 데이타로 구성되며 Size 는 400 M Byte 이다.

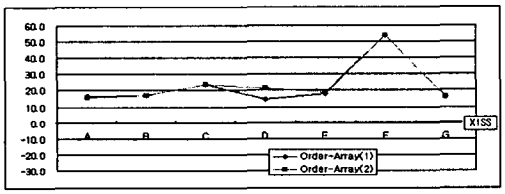
실험은 XISS 와 우리가 제안한 [그림 5]의 Order-Array 의 Data Table (C)와 (D)를 모두 사용하여 비교하였다.



[그림 6] 실험데이타의 XML 노드 구성 (XISS vs Order-Array)



[그림 7] 실험 프로그램의 실행화면



A : /법령/법령내용/정보관리/법령명_한글
B : /법령/법령내용/정보관리
C : /법령/법령내용/법령명_한자
D : /법령//소관부처
E : //소관부처
F : /법령//관련법령정보/법령명_한글[.='여권법']
G : //내용

[그림 8] XPath 의 질의 실험 결과

실험결과 XISS 보다 조금 나은 성능평가가 되었지만, 패스의 최하위노드가 많을 시 좋은 성능향상이 있었다. Order-Array 의 Data Table(C)와 (D)의 격차는 거의 유사하였으나 질의에 여러 종류의 조인연산이 들어간 경우엔 단일 데이터테이블에서 검색하는 Data Table(D)의 성능이 높다.

5. 결 론 및 향 후 연 구 과 제

본 논문에서는 XML 문서의 효과적인 인덱싱 기법을 제안하였으며 기존의 인덱싱 기법에서의 검색, 삽입시의 오버헤드를 최소화 하는데 중점을 두었다. 본 논문에서 제안된 인덱싱 기법은 Numbering-Scheme 을 기반으로 하지만, 노드범위(Node-Range)와 Order-Array 를 추가하여 XPath 질의시 데이터테이블의 preorder 와 postorder 를 비교하지 않고 빠르게 검색한다. 또한 대용량 XML 문서의 추가삽입, 구조가 다른 XML 문서의 삽입, 삭제 시에도 효과적으로 처리한다. 그러나, 다양한 패스의 복잡한 조인연산이 들어간 경우 많은 데이터테이블의 조인연산이 필요하므로 성능이 떨어질 수 있다. 향후 이와 같은 복잡한 XPath 질의에도 효과적으로 검색할 수 있는 방법을 지속적으로 연구할 것이다.

참고문헌

[1] David Hunter 외, Beginning XML, Wrox Press,2000

[2] Hirosho Oshikawa, Manabu Ohta, "An Active Web-based Distributed Database system For E-Commerce" Proceedings of the International Workshop on web Dynamics, with the 8th ICDT'01. London, UK,3,2001.

[3] Jonathan Robie, Joe Lapp, David Schach, "XML Query Language (XQL)" http://www.w3.org/TandS/QL/QL98/-xql.html

[4] A.Deutsch,M.Fernandez,D.Florescu,A.Levy,D.Suciy, "X-ML-QL: A Query Language For XML", http://www.w3.-org/TR/1998/Note-xml-QL-19980819/

[5] Paul F.Dietz. Maintaining order in a linked list. In Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing,pages 122-127, San Francisco, California, May 1982.

[6] S.Abiteboul et al. The Lorel Query Language For Semistructured Data, Int, J. on Digital Libraries 1(1): 68-88. 1997.

[7] 민경섭,김형주,"상이한 구조의 XML 문서들에서 경로질의처리를 위한 RDBMS 기반 역 인덱스 기법", 정보과학회 논문지 30 권 제 4 호 420~428.

[8] Quanzhong Li, Bonki Moon, "Indexing and Querying XML Data For Regular path expression",Proceedings of the 27th VLDB Conference, 361-370, Roma, Italy, 2001.

[9] Brian F.Cooper, Neal Sample, Michael J.Franklin, GisliHjaltason, Moshe shadmon, "A Fast Index For Semi-structured Data", Proceedings of the 27th VLDB Conference, Roma, Italy, 2001.

[10] Shu-Yao Chien, Zografoula Vagena, Donghui Zhang, Vassillis J.Tsotras, Carlo Zaniolo, "Efficient Structural Joins On Indexed XML Documents", Proceedings of the 28th VLDB Conference, 2002

[11] Yi Chen,Susan B.Davidson,Yifeng Zheng,"BLAS : An Efficient XPath Processing System", SIGMOD 2004 June 13-18, 2004

[12] Donald Knuth. The Art of Computer Programming, Vol.III, Sorting and Searching, Third Edition. Addison Wesley, Reading, MA,1998.