제26회 한국정보처리학회 추계학술발표대회 논문집 제13권 제2호 (2006. 11)

RDBMS에 기반한 XML 문서의 경로 저장과 숫자 매칭 기법

봉하익, 황병연
가톨릭대학교 컴퓨터공학과
e-mail:{drvong, byhwang}@catholic.ac.kr

A Path Storing and Number Matching Technique of XML Documents Based on RDBMS

Ha-Ik Vong, Byung-Yeon Hwang
Dept. of Computer Engineering, The Catholic University of Korea

요 약

최근 XML(eXtensible Markup Language) 사용의 증가로 인해 다량의 대용량 XML 문서가 이용되고 있음에 따라, 효율적인 문서 관리를 위한 XML문서의 데이터 모델과 저장 스키마를 어떻게 구현할 것인가에 대한 연구가 활발히 진행되고 있다. 이에 본 논문에서는 관계형 데이터베이스를 기반으로 한 XML문서에 대한 효율적인 저장, 검색 및 관리 기법으로 노드의 텍스트 값이나 속성 값이 존재하는 경로만을 저장하고, 노드 표현에 따라 고유 노드명 식별자(Node Expression Identifier)를 부여함으로써 부여된 노드 식별자를 매칭하는 숫자 매칭(Number Matching)기법을 제안한다. 그리고 이를 입증하기 위해 XPath 질의들에 대한 처리 성능을 기존 방법과 비교함으로써 제안한 방법의 효율성을 제시한다.

1. 서론

1996년 W3C(World Wide Web Consortium)에서 XML(eXtensible Markup Language)[1]이 제안된 이후 그 사용이 증가하였으며, 이로 인해 XML 데이터의 양도 빠른 속도로 늘어나고 있다. 이를 뒷받침해주기 위한 효율적인 접근 방법의 필요성이 커지면서 여러 연구가 발표되고 진행되어 왔다. 특히, 문서상의 트리 구조가 복잡해지면서도 깊이(depth)가 깊은 대용량의 XML 문서에 대한 효율적인 저장, 검색, 관리를 위한 XML 문서 관리 시스템의 연구가 활발히 진행되고 있다.

이에 본 논문에서는 관계형 데이터베이스를 기반으로 한 대용량 XML 문서에 대한 효율적인 저장 및 질의 처리 기법으로, 노드의 텍스트 값이나 속성 값이 없는 경로를 저장하지 않음으로써 사용자가 요구하는 경로만 저장하는 방식을 제안한다. 또한 기존의 데이터베이스 저장 또는 검색 스키마에서 사용하던 경로에 대한 문자열 매칭(String Matching)기법[2,3]을 숫자 매칭(Number Matching)기법으로 변환한다. 이 숫자 매칭 기법은 노드의 지리적 위치 또는 상하 및 위치에 의존하여 고유 번호를 붙이는 Ordered Encoding 기법[4]이 아닌, 노드명칭에 따라 고유 번호를 부여하는 노드 식별자 개념을 적용하였다.

제안하는 두 가지 방법은 문자열 경로가 저장되는 테이블 저장 공간의 축소와 매칭 시간의 절약을 통해 질의 처리 성능을 향상시키는 효과를 보이며, 동시에 트리 구조 내의 갱신 작업을 수행했을 때 발생할 수 있는 노드의 Renumbering이 없어진다.

본 논문의 구성은 다음과 같다. 2장에서는 관련 연구에 대해 소개하고, 3장에서는 본 논문에서 제안하는 XML문서의 데이터 모델 및 저장 스키마에 대해서 설명한다. 4장에서는 성능 분석에 대해 서술하며, 5장에서는 결론 및 향후 연구과제에 대해 기술한다.

2. 관련 연구

Model-mapping 접근법을 기초로 하여 관계형 데이터베이스 스키마를 구현한다면, 주요 문제점들 중 하나는 트리 모형 내의 기본 구조를 (객체) 관계형 스키마로 어떻게 매핑(mapping)시키느냐 하는 것이다. 지금까지 몇 가지 접근법들이 제시되어 오고 있다. 예를 들면, edge를 저장하는 방식[5], 경로와 영역(region)의 결합이라는 방식으로 XML 트리 구조를 표현한 XRel[2], 경로와 노드식별자로 표현하는 XParent[3] 등이 있다. 또한 노드에 순서화된 숫자를 부여하는 방식을 사용하여 노드간의 관계 파악을 정의하는 Ordered Encoding 기법[4]이 있다.

2.1 XRel

XRel은 XML 문서의 트리 구조 내에서 인스턴스 내의 루트노드를 제외한 루트로부터 각 노드까지의 모든 경로들을 열거했으며, 관계형 속성들내의 경로 표현들 그 자체를 저장시켰다. 또한 모든 가능한 경로 표현들이 하나의 문자열로써 데이터베이스에 저장되기 때문에 문자열 매칭(String Matching)이라는 방식으로 처리할 수 있다. 그러나 XRel은 문자열로 된 경로 표현식을 데이터베이스에 저장시키면서 자료 값을 갖지 않는 경로까지 저장시킴으로써 실제 사용자가 검색 시에 사용하지 않는 경로까지 저장시킨다. 이에 제안 방식은 자료 값을 갖는 경로만 저장시키며, 문자열의 경로를 숫자 경로로 저장시키는 방식을 제안한다.

2.2 Order Encoding

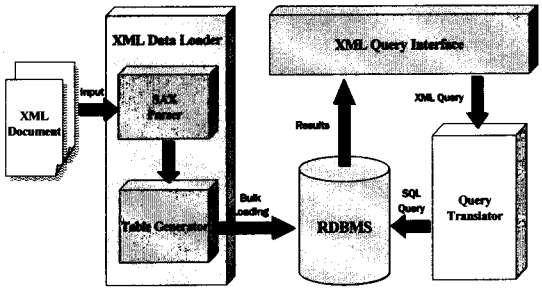
XML 문서의 각 노드의 순서를 코드화하는 방법으로 제안된 이 방법은 전역 순서 코딩(Global order encoding), 지역 순서 코딩(Local order encoding), 듀이 순서 코딩(Dewey order encoding), 같은 이름을 갖는 형제노드 순서 코딩(Same sibling order encoding)을 포함하고 있다. 전역 순서 코딩은 문서 내에서 노드의 절대적인 위치를 나타내는 숫자를 할당하는 방법으로 질의 위주의 작업에 적합하며, 지역 순서 코딩은 형제노드들 사이에서의 관계적 위치를 나타내는 숫자를 할당하는 방법으로 갱신 위주의 작업에 적합하다. 그리고 듀이 순서 코딩은 문서의 루트로부터 노드까지의 경로를 나타내는 벡터 표현을 할당하는 방식으로 질의와 갱신이 혼합된 작업에 적합하다. 같은 이름을 갖는 형제노드 순서 코딩 기법은 같은 이름의 형제 노드에 대해 부분적인 순서를 부여하는 방식이고 이는 다른 기법들과 결합하여 보완적으로 사용될 수 있는 방식이다.

그러나 Order Encoding 방식은 모든 노드마다 고유의 노드 순서를 부여함으로써, 노드가 추가 입력되었을 때에는 순서에 영향을 미치는 형제 노드 또는 자식 노드의 노드 순서를 갱신해야 하는 문제가 생긴다.

3. 제안하는 데이터 모델 및 저장 스키마

XML 문서는 트리로 볼 수 있으며, 단말노드는 자료 값 즉, 텍스트에 해당되고 내부노드는 XML 엘리먼트에 해당된다. 그러나 일반적인 XML 문서에서는 단말노드만이 자료 값을 갖는 형태를 벗어나 내부노드도 자료 값을 갖는 문서가 존재하므로 내부노드 역시 자료 값을 가진 문서를 바탕으로 한다.

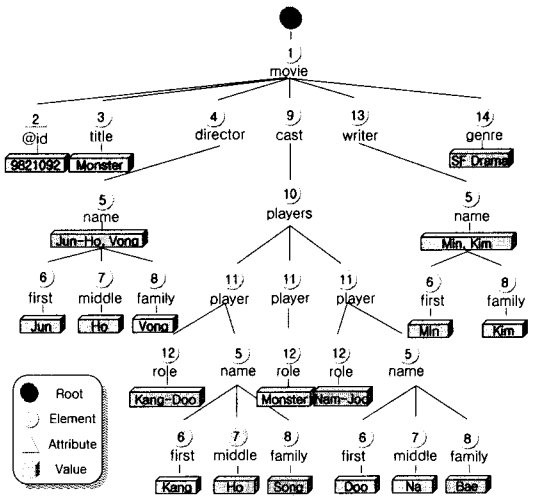
또한 XML 트리에 대한 분해 및 저장 방법은 경로를 저장하는 방식을 사용하며 XML 질의 모델로써 XPath(XML Path Language)를 사용한다. <그림 1>은 RDBMS에 기반하여 본 연구에서 제안한 관리 시스템 구조를 나타낸다.



<그림 1> XML 관리 시스템의 기본 구조

3.1 XML 문서의 데이터 모델

본 논문에서는 XML 문서를 표현하기 위하여, 각 노드의 순서 및 위치가 아닌 명칭에 따라 번호를 부여하는 데이터 모델을 제시한다. <그림 2>는 제안 방식을 바탕으로 한 XML 문서의 한 예이며, 이 방식은 노드명이 반복 사용됨에 따라 번호 역시 중복 사용됨을 알 수 있다.



<그림 2> XML 트리 모형의 예

이 때 같은 노드명(번호)을 가짐과 동시에 노드간의 순서 역시 같을 경우, 중복되는 동일한 경로 표현마다 각각의 고유 Index값을 부여하여 동일 경로를 구분한다.

제안하는 데이터 모델은 Tatarinov[4]가 제안한 모든 노드마다 고유한 순서를 부여하는 순서화된(ordered) XML 데이터 모델과 비교될 수 있다. 이 Ordered Encoding 방식은 부모-자식 관계 및 조상-후손 관계 그리고 형제관계를 고려하여 순서를 부여한다. 그러나 제안 방식은 노드간 순서를 고려하지 않았기 때문에 갱신 작업이 발생했을 때 노드명에 새로운 번호를 부여하거나 또는 노드명이 기존의 것과 동일하다면 기존 번호를 부여함으로써, 영향 받는 다른 노드의 번호를 다시 부여하지는 않는다.

3.2 데이터베이스 저장 스키마

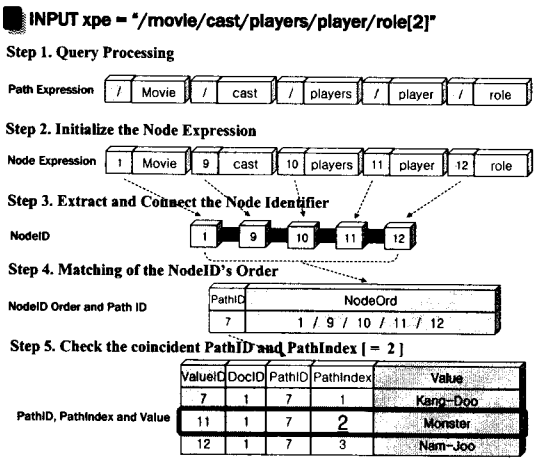
본 논문에서는 경로를 저장함에 있어서 모든 가능 경로를 저장하는 것이 아니라, 자료 값을 갖는 경로만 저장하는 방식을 사용한다. 이는 XML 문서를 검색하는 사용자가 실제로 텍스트 값이나 속성 값이 없는 사항에 대해서는 검색을 하지 않거나, 검색을 하더라도 null 값을 결과로 보이기 때문에 불필요한 저장 공간의 낭비를 막기 위함이 그 목적이다.

또한 데이터 타입의 크기에 따라서 메모리의 크기와 형태가 결정되는데, 기존 방식이 다수의 문자열을 경로 테이블에 저장시켰다면 제안 방법은 숫자를 경로 테이블에 저장시킴으로써 경로 테이블에 할당되는 메모리의 양을 줄이며, 이는 검색 시간을 절약하는 효과를 나타낸다.

즉, 경로를 표현할 때 경로에 해당하는 노드명을 모두 열거하여 저장하고 검색 시 이를 매칭하는 문자열 매칭(String Matching)방식 (예: movie/cast/players/player/role) 대신 경로에 해당하는 노드 표현 식별자를 저장하여 매칭하는 숫자 매칭(Number Matching)방식(예: 1/9/10/11/12)을 채택하여 Path 테이블의 NodeOrd에 저장한다.

예로 들은 경로 표현이 문자열 매칭 방식으로 저장된다면 30Byte를 차지하게 되는 반면 숫자 매칭은 12Byte만 차지하게 된다. 그리고 이 경로에 해당하는 자료 값은 'Kang-Doo', 'Monster'와 'Nam-Joo'로써 서로 다른 세 개의 경로가 같은 표현형을 갖고 있으며, Value 테이블에 동일한 경로마다 Index를 부여한 PathIndex라고 하는 동일 경로 구별자를 통해 구별할 수 있다.

<그림 3>은 앞에서 예로 들은 "movie/cast/players/play/role"에서 2번째 순서의 자료 값을 찾는 결과로 'Monster'를 추출하는 과정을 나타낸 것이다.



<그림 3> 경로 질의 검색 과정

이를 바탕으로 한 노드 타입별 관계형 저장 스키마는 다음과 같다.

Document(DocID, DocExp)
Node(NodeID, NodeExp)
Path(PathID, NodeOrd)
Value(ValueID, DocID, PathID, PathIndex, V_Text)
Attribute(AttrID, DocID, PathID, PathIndex, A_Text)

여기서 DocID, NodeID, PathID, ValueID, AttrID는 문서, 노드명, 경로, 자료 값, 그리고 속성 값의 고유 식별자이며 DocExp와 NodeExp는 해당 문서명과 노드명의 표현형이다. NodeOrd는 숫자로 표현된 경로 표현식, PathIndex는 동일 경로 구별자이며 V_Text와 A_Text는 자료 값과 속성 값이다.

3.3 질의 처리 알고리즘

제안 방법이 수행하는 질의 처리 알고리즘은 <그림 4>와 같다. 질의어에서 문자열을 노드명 식별자로 전환하고 Path 테이블에 저장된 노드명 식별자 표현과 일치하는 PathID를 찾는다<그림 5>. 이어서 Value 또는 Attribute 테이블에 저장된 해당 노드를 찾는다<그림 6>.

```
Procedure SearchNodes(PEQ : Path Expression of Query)
returns matched : set of nodes
begin
   //replace PEQ by NodeIDs' Expression
   begin
        replace PEQ by NodeID that is matched in Node Table
        if(PEQ has "//" at the beginning) then
            replace "//" by "%/";
        elesif(PEQ has "//" in the middle or its end) then
            replace "//" by "/%/";
        else store replced ones in replacedNodeOrdExp;
   end

   // find PathIDs that are matched with NodeOrd stored
      in Path Table
   initialize searchedPathIDset;
   begin
        if( replaced ) then searchPathID;
        store PathIDs in searchedPathIDset;
   end

   // find nodes with the equivalent PathID, Index and
      Value in TEXT Table
   initialize nodeSet;
   for(i=0; searchedPathIDset[i] != NULL; i++) do
   begin
       if (PEQ has Index) then
          nodeSet += findMatchedValue_IX(matchedPathIDset[i], Index);
       elseif (PEQ has Text value) then
          nodeSet += findMatchedValue_TEXT(matchedPathIDset[i], txt_value);
       elseif (PEQ has Attribute value) then
          nodeSet += findMatchedValue_Attribute(matchedPathIDset[i], attr_value);
       else nodeSet += findMatchedValue_EQ(matchedPathIDset[i]);
   end
   return nodeSet;
end
```

<그림 4> 질의 처리 알고리즘

```
Procedure searchPathID(replacedNodeOrdExp)
returns matched(PathID) in Path Table
begin
        SELECT PathID
        FROM Path p1
        WHERE p1.NodeOrd LIKE ' "+ replacedNodeOrdExp +"'
end
```

<그림 5> Path 테이블에서 해당 ID를 찾는 알고리즘

```
Procedure findMatchedValue_EQ(pathID)
returns matched(DocID, ValueID)s in Text Table
begin
    SELECT DocID, ValueID
    FROM Text t1
    WHERE t1.PathID = pathID
end

Procedure findMatchedValue_IX(pathID, Index)
returns matched(DocID, ValueID)s in Text Table
begin
    SELECT DocID, ValueID
    FROM Text t1
    WHERE t1.PathID = pathID AND t1.PathIndex = Index
end

Procedure findMatchedValue_TEXT(pathID, text_Value)
returns matched(DocID, ValueID)s in Text Table
begin
    SELECT DocID, ValueID
    FROM Text t1
    WHERE t1.PathID = pathID AND t1.value LIKE '%"text_Value"%';
end

Procedure findMatchedValue_Attribute(pathID, attr_Value)
returns matched(DocID, ValueID)s in Text Table
begin
    SELECT DocID, ValueID
    FROM Text t1
    WHERE t1.PathID = pathID AND t1.value LIKE '%"attr_Value"%';
end
```

<그림 6> Text 테이블에서 해당 노드를 찾는 알고리즘

4. 실험 및 성능 평가

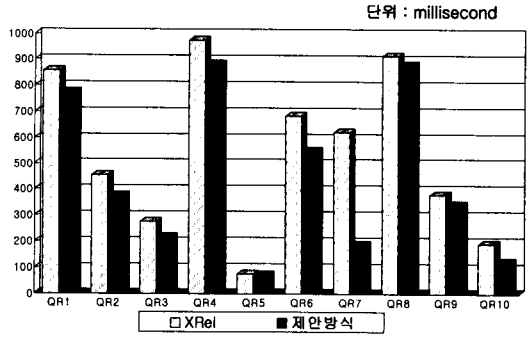
본 실험에서 사용한 운영체제는 Window XP, 하드웨어는 Pentium 4-1.72GHz, 1GB RAM이며, 데이터베이스 시스템으로 MS SQL-Server 2000을 사용하였다. 실험용 데이터로는 37개의 XML 문서를 갖고 있는 'Bosak Shakespeare collection'을 사용하였으며, 비교 평가를 위한 검색 기법으로 XRel[2]을 사용하였다.

비교 분석을 위해 <그림 7>과 같이 아래의 질의를 사용한다.

QR	Query Expression
QR1	/PLAY/FM/P
QR2	/PLAY/PERSONAE/PGROUP/GRPDESCR
QR3	/PLAY/PROLOGUE/SPEECH/LINE
QR4	/PLAY/INDUCT/SCENE/SPEECH/LINE
QR5	/PLAY/PROLOGUE/TITLE[4]
QR6	/PLAY/ACT/SCENE/SPEECH/[SPEAKER='LEONATO']
QR7	/PLAY//SUBHEAD
QR8	//INDUCT/SCENE//LINE
QR9	//EPILOGUE//LINE
QR10	//PLAYSUBT

<그림 7> 경로 질의

위의 질의에 대한 결과는 다음 <그림 8>과 같다.



<그림 8> 경로 질의 처리 시간

<그림 7>에 나타난 질의문은 QR1-4번이 부모-자식 관계에 대한 질의, QR5-6번이 특정 인덱스를 갖는 질의 그리고 QR7-10번이 조상-후손 관계에 관한 질의이다. 실험 결과는 <그림 8>에서 보이는 것처럼 제안한 방식이 기존 방식보다 우수하게 나타났다.

	저장되는 경로 수	Path Table의 문자열 크기	Node Table의 문자열 크기
XRel	57개	1308Byte	없음
제안 방식	40개	439Byte	127Byte

<그림 9>경로 검색시 사용되는 Table 정보

이는 <그림 9>에서 알 수 있듯이 제안 방식이 기존 방식보다 저장되는 경로 수가 적고, 문자열로 저장되는 것보다 고유 번호로 저장되는 것이 경로 테이블의 크기를 줄이는 효과를 나타내기 때문이다.

물론 기존 연구는 검색시 노드 정보를 불러올 노드 테이블을 사용할 필요가 없이 경로 테이블만 검색하면 되지만, 제안 방식은 노드 테이블도 사용해야 한다. 그러나 앞에서 밝힌 제안 방식의 장점들이 이 점을 보완해주고 있어 여러 질의에서 성능 개선을 나타냈다.

5. 결론 및 향후 연구 과제

본 논문에서는 XML 문서의 효율적인 질의처리를 위한 경로 기반 저장 기법의 보완점을 제시하였다. 제안 방법에서는 자료 값이나 속성 값을 갖는 경로만 저장시키고, 경로 저장 및 검색시 문자열이 아닌 숫자 매칭을 사용함으로써 저장 공간을 축소시키고 검색 시간을 절약하는 효과를 나타냈다. 또한 이에 대한 기존 연구와의 비교 실험을 통해 우수한 검색 성능을 보임을 입증하였다.

그러나 역색인 기법이나 Numbering scheme에 기반한 경로 조인 기법들과의 비교 연구를 통해 제시 방법이 얼마나 효율적인 것인가에 대한 고찰이 필요하다.

향후에는 조상-후손 및 형제 관계를 효과적으로 처리할 수 있는 방법과 좀 더 효율적인 경로 축약 및 검색 기법에 대하여 연구할 것이다.

참고문헌

[1] "Extensible Markup Language(XML) 1.1", W3C Candidate Recommendation, 2002

[2] M. Yoshikawa and T. Amagasa, "XRel: A Path-Based Approach to Storage and Retrieval of XML Documents using Relational Databases," ACM Transactions on Internet Technology, Vol. 1, No. 1, pp. 110-141, 2001

[3] H. Jiang, H. Lu, W. Wang, and J. X. Yu, "Path Materialization Revisited: An Efficient Storage Model for XML Data," In Proc. of the 13th Australasian Database Conference(ADC), pp. 85-94, Melbourne, Australia, Jan. 2002

[4] I. Tatarinov, S. D. Viglas, K. Beyer, J. Shanmugasundaram, E. Shekita, and C. Zhang, "Storing and Querying Ordered XML Using a Relational Database System," In Proc. of ACM SIGMOD Conf., 2002

[5] D. Florescu and D. Kossmann, "Storing and Querying XML Data Using an RDBMS," IEEE Data Engineering Bulletin, Vol. 22, No. 3, pp. 27-34, 1999