Nested Interval 을 이용한 XML 문서의 저장 및 질의 기법

송용호*, 나갑주*, 이상원*
* 성균관대학교 정보통신공학부
e-mail : {neilkr,factory,swlee}@skku.edu

Nested Interval Encoding with Continued Fractions for XML Storage & Retrieval

Yong-Ho Song*, Gap-Joo Na*, Sang-Won Lee*
*Dept. of Computer Engineering, Sungkyunkwan University

요 약

XML(Extensible Markup Language)이 데이터 표현(data representation)과 문서 교환(data exchange)의 표준으로 지정됨에 따라 데이터베이스(database, DB)에 XML 문서를 저장하고 질의하기 위한 연구가 활발히 진행되고 있다. 특히, 현재 주류를 이루고 있는 관계형 DB 에 저장하기 위한 XML 인덱싱(indexing) 기법에 대한 연구도 다양하게 진행되고 있다. 본 논문에서는 XML 문서를 관계형 DB 에 효율적으로 저장하고 질의하기 위한 방법으로서 기존의 트리(tree) 구조의 데이터를 관계형 DB 에 Nested Interval 인덱싱 기법을 적용하여 XML 문서를 저장하는 방법에 대해 연구한다. 기존의 저장 기법들의 경우 XML 문서를 효율적으로 질의하기 위한 인덱싱을 수행하기 때문에 입력 후 추가되는 노드(node), 혹은 노드 집합의 입력 시에는 전체 혹은 일부분의 XML 문서를 재-인덱싱 해야 하는 비효율이 있다. 그러나, Nested Interval 의 경우에는 재-인덱싱이 불필요하다. 본 논문에서는 기존의 트리 구조 데이터의 인덱싱 기법들에 대한 비교와 함께 Nested Interval 을 이용한 XML 문서의 인덱싱 기법에 대해 기술한다.

1. 서론

DB 에 XML 문서를 저장하고 질의하기 위한 연구들은 크게 XML 전용 DB 를 이용하여 XML 문서를 효율적으로 저장하고 질의하는 방법와 객체-관계형 DB 내에 XML 문서를 저장하는 방법, 그리고 순수 관계형 DB 내에 XML 문서를 저장하는 방법에 대한 연구로 나누어 볼 수 있다.

본 논문은 현재 DB 의 주류를 이루고 있는 순수 관계형 DB 에 XML 문서를 저장하는 기법에 대한 연구로 새로운 XML 문서의 삽입이나 기존 문서의 수정, 삭제 시 [1][2][3]등의 연구에서 불가피하게 발생하는 재-인덱싱 혹은 재-인코딩(encoding) 문제를 해결하고 효율적으로 질의하기 위한 방안에 대해 연구이다.

XML 문서는 트리 구조이므로, 관계형 DB 내에서 트리 구조에 대한 인코딩의 효율성에 따라 성능이 좌우된다. 본 논문에서는 [5][6][7]에서 소개된 트리 인코딩 기법인 Nested Intervals with Continued Fractions 을 이용하여 트리 구조인 XML 문서의 저장, 질의가 가능함을 검증한다.

구성은 다음과 같다. 2 장에서 관계형 DB 내에서 사용되는 대표적인 트리 인코딩 기법과 기존의 XML 저장 기법인 Xpath Accelerator 에 대해 소개하고, 3 장에서 본 논문에서 사용한 Nested Interval Tree Encoding with Continued Fractions 에 대해 설명한다. 4 장에서 구현 과정과 실험 결과를, 마지막으로 5 장에서 결론과 향후 연구 내용에 대해 기술한다.

2. 관련연구

본 논문은 XML 문서를 효율적으로 저장하고 관리하는 방안에 대한 연구이다. 그러나, XML 문서가 트리

구조이므로 우선 관계형 DB 내에 트리 구조의 데이터를 입력하는 연구들에 대해 기술한다. 또한, 최근 DB 에 XML 데이터를 저장하는 기법으로 소개된 XPath Accelerator[3]에 대한 연구에 대해서도 기술한다.

- Adjacency List Model [5] : 트리 구조의 데이터를 저장하기 위해 각 노드의 키값을 부여하고 현재 노드의 부모 노드를 동시에 저장하는 기법으로 특정 노드에서 부모 노드를 쉽게 구할 수 있는 장점이 있다.

- Materialized Path Model [5] : 현재 노드에서 부모 노드까지의 Path 저장하는 기법으로 XML 의 포함질의 혹은 노드 자체에 대한 검색에 장점이 있다.

- Nested Set Model [5] : Nested Set 모델은 각 노드를 (lft, rgt)값의 형식으로 정의하고 이를 DB 에 저장한다. Lft 와 rgt 의 경우 각 노드를 탐색하며 순차적으로 인덱싱한다. 이러한 인덱싱 기법으로 데이터를 저장할 경우 전체 노드의 개수는 루트 노드의 rgt/2 개이다. 또한, rgt-lft=1 인 노드는 단말 노드이고, 자식 노드(c_lft, c_rgt)는 항상 부모 노드(p_lft, p_rgt)에 대해 p_lft<c_lft, p_rgt>c_rgt 인 관계를 가지게 되므로 트리 데이터를 질의함에 있어 lft, rgt 컬럼을 비교하여 쉽게 구할 수 있는 장점을 지니고 있다.

- XPath Accelerator [3]: XML 문서를 관계형 데이터베이스에 저장하기 위해 제안된 인덱스 모델로 각 노드의 전위 탐색 값과 후위 탐색 값(pre order value, post order value)을 부여하는 인덱싱 기법이다. 그러므로, ancestor, following, preceding, descendant 의 좌표 평면을 얻을 수 있고 이를 이용하여 XML 데이터를 질의할 수 있다. XPath Accelerator 의 경우 위의 모델들과 다르게 XML 문서를 기준으로 연구된 기법으로 가장 효율적으로 XPath 에 대한 질의를 수행할 수 있다.

위에서 언급한 각 모델들과 기존의 [1][2]등의 경우, 이미 저장된 XML 데이터에 대해서는 장점을 지니고 있으나, 저장되는 XML 데이터의 엘리먼트들을 고정된 노드값으로 미리 인덱싱하기 때문에 새로운 노드의 삽입과 기존 노드의 수정 삭제 시에 모든 노드에 대한 재-인덱싱을 수행해야 하는 비효율이 있다.

3. Nested Interval Tree Encoding with Continued Fractions

본 절에서는 관계형 DB 에 XML 데이터를 저장하기 위한 Nested Interval Model[6]에 대해 기술한다. Nested Intervals 는 앞 절에서 언급한 Nested Set[5]을 일반화 시킨 구조이고, Materialized Path 로 쉽게 변환이 가능하다. 따라서, Nested Set 과 Materialized Path 의 장점을 모두 이용할 수 있다. [6]에서 제시된 Nested Interval 의 경우 노드의 깊이에 따라, 또는 자식 노드의 수에 따라 노드에 대응하는 값이 2 의 지수승으로 증가하기 때문에 대용량 XML 문서에 대해서는 적용이 어렵다. 본 논문은 이러한 문제점을 해결하기 위해 제시된 [7],[8],[9]의 기법들 중 XML 문서에 가장 적합한 형태인 Nested Interval Tree Encoding with Continued Fractions [8]기법으로 인코딩하고, 그 특성을 이용하여 노드 간의 관계를 수식을 통해 구함으로서 DB 의 접근을 최소화한다.

3.1 Basic Structure

Nested Interval 은 Nested Set 을 일반화 시킨 구조이므로 Nested Set 과 같이 하나의 노드를 (lft, rgt)의 구조로 표현한다. 그러나, Nested Set 모델과 다르게 각각의 값은 정수가 아닌 분수로 표현하므로 노드의 개수에 제한을 받지 않는다. 주요 특징을 살펴보면 부모 노드(plft,prgt)와 자식 노드(clft,crgt)의 관계는 plft<=clft and crgt<=prgt 와 같다. 따라서, 이러한 관계를 사용하여 부모와 자식 노드, 조상과 자손 노드를 쉽게 산출할 수 있다.

3.2 Nested Intervals Tree Encoding with Continued Fractions

[6]에서 제시된 기본적인 Nested Interval 의 경우 노드의 고유값은 노드의 좌표인 (lft, rgt)에서 lft+rgt 로 표현한다. 그러나, 이 경우 인코딩 특성상 노드의 레벨 혹은 노드의 sibling 값의 증가에 따라 고유값의 분모도 2 의 지수승으로 증가하는 단점이 있다. 그래서, 이러한 문제점을 해결한 [8]의 인코딩 기법을 사용한다. [8]의 인코팅 기법에서는 노드의 Materialized Path 의 인코딩 방법과, 각 노드의 값을 중심으로 수학적인 계산을 통해 노드 간의 관계를 구한다.

3.2.1 The Encoding

Nested Interval 에서는 a 와 b 는 $a \geq b \geq 1$ and GCD (a,b)=1 인 관계인 분수(a/b) 형태로 정의한다. 아래의 예처럼 특정 노드가 4913/1594 의 고유값을 가지면, Euclidean Algorithm 에 의해 Materialized Path 인 3.12.5.1.21 을 구할 수 있다.

$$4913 = 1594 * 3 + 131$$
$$1594 = 131 * 12 + 22,$$
$$131 = 22 * 5 + 21$$
$$22 = 21 * 1 + 1,$$
$$21 = 1 * 21 + 0$$

3.2.2 Continued Fraction

반대로 Materialized Path 로부터 고유값을 구하는 과정은 Simple Continued Fraction 을 사용한다. 아래의 예처럼 Materialized Path 3.12.5.1.21 에서 4913/1594 를 구한다.

$$3 + \cfrac{1}{12 + \cfrac{1}{5 + \cfrac{1}{1 + \cfrac{1}{21}}}} = \frac{4913}{1594}$$

3.2.3 Nested Intervals

위와 같이 노드는 하나의 고유값을 갖지만, Nested Interval 의 특성을 사용하기 위해서 각 노드의 Interval 을 구한다. Interval 은 다음과 같은 변환으로 얻어진다.

$$3+\cfrac{1}{12+\cfrac{1}{5+\cfrac{1}{1+\cfrac{1}{21+\cfrac{1}{x}}}}}=\frac{4913x+225}{1594x+73}$$

여기서 x 의 범위는 x∈[1, ∞]로 가정한다. 따라서, 주어진 노드의 Interval 은 (5138/1667, 4913/1594)의 범위를 가진다. 이 과정에서 interval 간의 분모, 분자 간의 차는 부모 노드의 고유값이다. 위의 예에서 5138-4913=225, 1667-1594=73 이므로 부모 노드는 225/73 이다. 자세한 증명은 [8]를 참조하기 바란다.

3.3 XPath Queries

XML 문서에 대한 질의는 주로 XPath[4]를 사용한다. 따라서, Nested Interval 을 이용하여 XML 문서를 저장하였을 경우 XPath 형태의 질의를 관계형 DB 에서 사용되는 질의문(Structured Query Language, SQL)로 표현이 가능해야 한다. 이 절에서는 대표적인 XPath 에 Nested Interval 을 적용하는 방법에 대해 기술한다.

[3]에서 제시하는 XPath Accelerator 의 경우 새로운 노드의 입력과 기존 노드의 수정, 삭제 시에 모든 노드를 다시 인코딩 해야 하는 비효율이 있다. 또한, 질의 시 노드 간의 관계는 2 차원의 평면과 같이 각 면에 대해 조건절을 포함해야 한다. 이로 인해, XPath Accelerator 에서는 노드 간의 관계를 구할 경우, 무한대의 범위에 대해 검사를 해야 하는 문제점이 있다. 즉, XPath Accelerator 에서는 성능 향상을 위해 조건절의 검색 범위를 줄이는 것이 주안점이다. 그러나, 본 논문에서 사용한 Nested Interval 의 경우는 각 노드 간의 관계는 조건절의 검색 범위가 interval 을 통해 제한이 되므로 XPath Accelerator 보다 나은 성능을 낸다.

3.3.1 child node & parent node

부모 노드를 구하기 위해서는 먼저 Materialized Path 를 구하고, 이를 활용하여 부모 노드를 산출한다. 예를 들어 노드의 특정값이 4913/1594 이면 앞 절에서 설명한 Euclidean Algorithm 을 활용하여 3.12.5.1.21 의 Materialized Path 를 구하고, 이를 이용하여 각 단계의 sibling number 를 2*2 형태의 행렬로 변환한 후 이들 간의 곱으로 부모 노드의 값을 구한다. 자세한 증명은 [8]를 참조하기 바란다. 또한, 자식 노드를 구하기 위해서는 현재 노드의 값에 새로운 자식의 sibling 을 2*2 행렬의 형태로 곱할 경우에 가능하다. 아래의 예와 같이 3.12.5.1.21 의 각 sibling 을 행렬의 (1,1)에 위치하여 곱을 구할 경우, 결과 행렬의 (1,1)과 (2,1)의 값인 4913/1594 가 노드의 고유값이고, (1,2)와 (2,2)의 값인 225/73 이 현재 노드의 부모 노드이다.

$$\begin{bmatrix}3 & 1\\ 1 & 0\end{bmatrix}\cdot\begin{bmatrix}12 & 1\\ 1 & 0\end{bmatrix}\cdot\begin{bmatrix}5 & 1\\ 1 & 0\end{bmatrix}\cdot\begin{bmatrix}1 & 1\\ 1 & 0\end{bmatrix}\cdot\begin{bmatrix}21 & 1\\ 1 & 0\end{bmatrix}=\begin{bmatrix}4913 & 225\\ 1594 & 73\end{bmatrix}$$

또한, 현재 노드값에 다음 자식 노드의 sibling 을 2*2 의 행렬 형태로 곱을 할 경우, 자식 노드의 고유값을 통해 원하는 자식 노드를 구할 수 있다.

3.3.2 ancestor node & descendant node

Ancestor 노드와 descendant 노드의 경우 Nested Interval 의 정의에 따라 쉽게 구할 수 있다. descendant 노드의 경우 특정 노드의 interval 값 사이에 고유값을 갖는 노드들이다. Ancestor 노드의 경우는 그 반대이다. 예를 들어, 특정 노드 a 의 interval 값을 (ax, ay)라 하고, descendant 노드의 interval 값을 (bx, by)라 하면, 모든 (bx, by)노드는 ax<bx<by<ay 의 관계가 성립된다. 반대로 (bx, by)의 ancestor 노드를 구할 경우에도 위의 관계를 이용하여 질의할 수 있다. 두 노드 간의 관계는 Nested Interval 의 정의를 나타내고 있는 [5],[6]를 참고하기 바란다.

3.3.3 following node & preceding node

Following 노드와 preceding 노드를 구하는 과정은 3.3.1, 3.3.2 의 과정을 이용한다. 즉, following 노드는 부모 노드의 (ax, ay)와 following 노드 (bx, by)와의 관계가 ax<ay<bx<by 인 노드이고, preceding 노드는 bx<by<ax<ay 인 노드이다.

3.3.4 following-sibling & preceding-sibling

현재 노드의 값의 분모, 분자에 부모 노드의 값을 더하면 following-sibling 노드이고, 빼면 preceding-sibling 노드이다. 왜냐하면, 각 sibling 노드들 간의 차가 현재 노드의 분자, 분모에 각각 부모 노드의 분자, 분모 만큼의 차를 가지므로 쉽게 산출할 수 있다.

4. Implementation and Experiment

본 절에서는 상용 관계형 DB 에 Nested Interval 인덱싱 기법 구현하는 과정과 XML 질의 시 대표적으로 사용되는 Xpath 질의를 SQL 로 변환하는 과정을 설명한다.

구현을 위한 환경은 펜티엄 4 2.0GHz, 1GB 램, 60G 하드 디스크의 하드웨어적인 환경과 Window XP Professional 운영체제와 관계형 DBMS 는 Oracle10g 를 사용하였다. 구현에 사용한 XML 문서는 [12]의 셰익스피어 희곡의 XML 문서이다.

4.1 XML document parsing and Table Schema

XML 문서를 저장하기 위하여 JAVA SAX Parser[8]를 이용하여 Materialized Path 로 파싱(parsing)하고, 파싱된 문서는 XML_DOCS 와 XML_DATA 테이블에 저장한다.

```
CREATE TABLE XML_DOCS( docID INT NOT NULL,
 doc_name VARCHAR2(30) NOT NULL,
 nodeID INT NOT NULL, node VARCHAR2(100) NOT NULL,
 attr VARCHAR2(10) NOT NULL, numer INT NOT NULL,
 denom INT NOT NULL, p_numer INT,
 p_denom INT, depth INT,
 CONSTRAINT xml_docs_pk PRIMARY KEY(docID,nodeID));

CREATE TABLE XML_DATA( docID INT NOT NULL,
 nodeID INT NOT NULL, value VARCHAR2(4000),
 CONSTRAINT xml_data_pk PRIMARY KEY (docID,nodeID));
```

4.2 Mapping functions

앞 절에서와 같이 Materialized Path 와 Nested Interval 간의 일대일 대응 관계이다. 따라서, 각 노드의 저장 시 JAVA Procedure 함수와 PL/SQL 함수를 이용하여 Materialized Path 를 Nested Interval 형태로 입력한다. 아래는 입력용 함수 그룹, 질의용 함수 그룹 그리고 노드 간의 관계를 구하는 함수 그룹이다.

- **Storing function group** : x_numer, x_denom, y_numer, y_denom
- **Path function group** : path, path_numer, path_denom, sibling_number
- **Relation function group**: parent_numer, distance, child_numer, …

4.3 Data inserting

아래 질의문은 위의 함수를 이용한 XML 데이터의 입력 SQL 이다.

```
INSERT INTO XML_DOCS (docID, doc_name, nodeID, node,
              attr, numer, denom, p_numer, p_denom)
    VALUES (8, 'hamlet.xml', 2, 'TITLE', 'false',
           path_numer('2.2'), path_denom('2.2'),
           path_pnumer('2.2'), path_pdenom('2.2'));

INSERT INTO XML_DATA (docID, nodeID, value)
    VALUES (8, 2,
       'The Tragedy of Hamlet, Prince of Denmark');
```

아래 표는 임의의 중간 노드들을 각각 다른 크기로 삭제한 XML 데이터를 입력한 다음, 추가적인 노드의 삽입 시의 수행 시간이다. 표를 보면, 추가되는 노드의 삽입 과정에서 재-인덱싱 과정이 불필요하므로 삽입되는 노드의 개수에 따라 수행 시간은 일정하게 증가한다.

노드 개수	100	500	1000	2000	3000	4000
입력시간(ms)	96	510	997	1992	3013	4101

4.4 Converting queries expressing Xpath

XML 데이터의 질의를 할 경우 주로 XPath 를 이용하여 XML 에 대한 질의를 수행한다. Nested Interval 을 XML 저장과 retrieval 에 적용하면, XPath 형태의 질의를 SQL 로 변환이 가능하다. 아래는 SQL 로 변환된 자손 노드를 구하는 질의문과 following-sibling 노드를 구하는 질의문이다.

```
<Query for descendant node>
• //ACT/ descendant::LINE
SELECT count(n2.node) FROM xml_docs n1, xml_docs n2
 WHERE n1.node='ACT' and n2.node='TITLE'
   AND n1.docID=n2.docID
   AND n1.numer/n1.denom > n2.numer/n2.denom
   AND (n1.numer+n1.p_numer)/(n1.denom+n1.p_denom)
     < (n2.numer+n2.p_numer)/(n2.denom+n2.p_denom) ;
[Elapsed: 00:00:00.10 Sec]

<Query for following node>
• //ACT/following::LINE
SELECT COUNT(n2.node)
  FROM xml_docs n1, xml_docs n2
 WHERE n1.node='ACT'
   AND n2.node = 'LINE'
   AND n1.docID = n2.docID
   AND n1.p_numer/n1.p_denom < n2.numer/n2.denom ;
[Elapsed: 00:00:00.73 Sec]
```

위의 질의문에서 조건절들은 Interval 에 의해 검색 범위를 제한하고, 함수를 이용한 수학적 연산으로 노드를 검색한다. 따라서, 대용량 데이터에 대해서도 일정한 수준의 성능을 보장한다.

5. 결론 및 향후 연구

본 논문에서는 관계형 DB 에 트리 구조의 데이터를 저장하는 기법인 Nested Interval 기법을 XML 데이터의 저장 및 질의 모델에 적용하였다. Nested Interval 기법은 관계형 DB 를 이용한 XML 데이터 저장 기법들 모두에서 나타나는 문제점인 새로운 노드의 삽입 시, 기존 노드의 수정(이동) 및 삭제 시 불가피하게 발생하는 재-인덱싱의 문제를 해결한다. 또한, Continued Fraction 의 형태로 노드를 저장하면, 각 노드 간의 관계를 수식을 통해 쉽게 구할 수 있다. 또한 XPath 형태의 질의 시에도 각 노드 간의 관계를 간단한 함수 호출을 하여 DB 의 접근 없이 원하는 노드의 값을 구하므로 소요 비용을 최소화할 수 있다.

향후 연구 과제로는 본 기법의 보다 객관적인 비교 분석을 위한 다양한 벤치마킹을 시도해 보아야 할 것이다. 또한, 현재 XPath 형태의 질의를 SQL 로 변환해야 하는 부분을 자동화하기 위해 XPath 파서(Parser)의 개발도 이루어져야 할 것이다. 마지막으로 객체-관계형 DB 에서 확장형 인덱스를 이용하여 본 기법을 적용하는 방안에 대한 연구도 이루어져야 할 것이다.

참고문헌

[1] D.Floresce, D.kossman, "Storing and Querying XML data Using a RDBMS", IEEE Data Engineering Bulletin, Vol.22, No 3, 1999

[2] I. Tatarinov et al., "Storing and Querying Ordered XML Using a Relational Database System", Proc. ACM SIGMOD Int'l Conf. on Management of Data, 2002

[3] Torsten Grust, "Accelerating XPath Location Steps", ACM SIGMOD, Madison, June, 2003

[4] W3C, XML Path Language(XPath), Version 1.0, W3C Recommendation, November 1999

[5] CELKO.J, "Joe Celko's Trees & Hierarchies in SQL for Smarties", Morgan Kaufmann, 2004

[6] TROPASHKO, V. 2003a. Trees in SQL:Nested Sets and Materialized Path. http://www.dbazine.com/tropashko4.shtml

[7] TROPASHKO, V. "Nested Intervals with Farey Fractions.", eprint arXiv:cs/0401014, 2004

[8] http://www.saxproject.org/

[9] The Plays of Shakespeare in XML. (http://www.xml.com/pub/r/396)