제22회 한국정보처리학회 추계학술발표대회 논문집 제11권 제2호 (2004. 11)

XML 문서의 효율적인 변경을 위한 XML 번호체계+

김영현, 강현철
중앙대학교 컴퓨터공학부
yhkim@dblab.cse.cau.ac.kr hckang@cau.ac.kr

XML Numbering Schemes for Efficient Update of XML Documents

Young-Hyun Kim, Hyunchul Kang
School of Computer Science and Engineering, Chung-Ang University

요 약

XML 의 중요성이 부각 되면서 XML 데이터의 효율적 관리 기법에 관한 연구가 활발히 수행되고 있다. XML 질의의 표준화 작업도 활발히 이루어져 현재 XQuery 가 유력한 표준으로 부각되었다. 그러나 XQuery 등이 완전한 XML 질의어가 되기 위해서는 변경 연산을 제공해야 하는데 XML 변경어의 표준화 작업이나 XML 변경 처리 기법에 대한 연구는 아직 미미한 실정이다. 본 논문에서는 e-Business 등 XML 데이터베이스 기반 웹 응용의 효율적 지원을 위한 XML 문서 갱신에 대한 기법을 다룬다. XML 문서의 효율적 갱신을 위한 RN(RmdEid, NextChildEid) 번호 체계, R(RmdEid) 번호 체계 을 제시한다. 어떤 XML 번호 체계를 사용하여 XML 문서를 저장하는 가는 XML 문서의 갱신 연산에 중용한 영향을 미친다. 이들 번호 체계들은 데이터베이스내의 테이블 스키마의 차이점으로 구분한 것이다. 이들 번호체계 각각에 대하여 XML 데이터베이스 시스템을 구현하여 이들 기능의 성능 및 공간 부담을 평가한 결과를 기술한다.

1. 서론

XML 의 중요성이 부각 되면서 XML 을 데이터 관리의 대상으로 하는 연구들이 많이 수행 되고 있다. E-Business 등 XML 데이터베이스 기반의 웹 응용은 웹에 산재해 있는 XML 문서들을 수시로 갱신하며 교환 작업을 한다. 이때 XML 문서의 갱신 연산을 효율적으로 한다면 성능 향상을 기할 수 있다.

본 논문에서는 XML 문서의 효율적인 갱신을 위한 번호체계에 대한 문제를 다룬다. 98 년에 W3C 의 XML 사양이 발표된 이래 XML 에 대한 연구들이 많이 수행되었으며 XML 데이터의 저장 및 질의 처리 기법에 대한 연구들은 많이 진척된 상태이다. 하지만, 본 논문에서 다루고자 하는 XML 문서의 효율적인 갱신에 대한 연구는 아직 미미한 실정이다. [4]에서는 XQuery 의 구문을 확장하여 다양한 XML 갱신 연산을 제시하고, XML 데이터가 관계 데이터베이스에 분할 저장되어 있을 경우 제시된 갱신 연산의 처리 문제를 다루었다. XML:DB 의 XUpdate Working Group 에서는 XML 변경어인 XUpdate 사양 [8]의 표준화 작업을 수행하고 있다.

E-Businnes 등 XML 데이터베이스 응용의 완전한 지원을 위해서는 문서 단위의 삽입 및 삭제 외에 문서 내 텍스트(PCDATA) 또는 속성값의 변경이나 엘리먼트 단위의 구조 변경 기능은 필수적이며, 이들을 효과적으로 구현하기 위한 다양한 연구와 성능 트레이드오프를 검토하는 작업이 중요하다. 특히, XML 문서에서는 대단위의 변경보다는 단말(leaf) 엘리먼트 단위의 변경 같은 소규모의 변경을 효율적으로 수행하는 기법이 빈도나 실용성 면에서 더 중요하다.

+ 본 논문은 한국과학재단 특정기초연구사업(R01-2003-000-10395-0) 지원으로 수행 되었음.

본 논문에서는 XML 문서의 삽입, 삭제, 변경 중에서 처리가 가장 복잡한 삽입 연산에 대해서만 다룬다. XML 문서에서의 삽입 과정은 질의를 통해 목적 엘리먼트를 찾은 후 새로운 엘리먼트를 실제로 삽입한 후 XML 문서를 데이터베이스에 저장 하기 위해 엘리먼트마다 할당한 엘리먼트 ID 의 고정화 작업을 끝으로 모든 삽입 과정을 마치게 된다. 이때 데이터베이스에 존재하는 XML 문서의 구조의 지침이 되는 정보의 매개체가 번호 체계이다. 효율적인 번호 체계를 가지면 데이터베이스 내에 존재하는 XML 문서의 계층 구조를 쉽게 파악 할 수 있을 뿐만 아니라 XML 문서 갱신 연산에 있어서도 강건하게 대처할 수 있다.

본 논문의 구성은 다음과 같다. 2 절에서는 본 논문에서 제시하는 번호 체계과 이를 사용한 XML 문서의 갱신에 대하여 기술 한다. 3 절에서는 이들의 구현과 성능 평가 결과를 기술한다. 마지막으로 4 절에서는 결론을 맺고 향후 연구 내용을 기술한다.

2. 번호 체계에 따른 XML 문서 갱신

XML Database 가 질의를 처리하는 과정에서 번호 체계는 System 내부에 존재하는 XML 문서의 구조를 설명해주는 지침이 된다. 입력으로 들어온 XML 문서 갱신 요청을 처리하기 위해 System 은 번호 체계를 통해 System 내부의 XML 문서를 검색한 후 목적 엘리먼트에 갱신 연산을 처리한다. 그리고, 갱신 연산의 결과로 번호 체계의 고정화 작업을 끝으로 질의 처리를 마치게 된다. 본 논문에서는 갱신 연산 후의 번호 체계의 고정화 작업의 정도에 따라 세가지 번호 체계를 제시한다.

2.1. RN-Scheme

그림 1 의 형식을 가진 XML 문서를 RN-Scheme(RmdEid, NextChildEid)으로 나타내면 그림 2 와 같은 트리 구조를 띄게 된다. 그림 2 의 각 노드의 괄호 안의 값은 (Eid, RmdEid, NextChildEid) 이다. Eid 는 preorder 순으로 XML 문서트리를 탐색 하면서 할당해 준 값이며, RmdEid 는 가장 오른쪽 자손의 Eid 에 해당하는 값이다. 그리고, NextChildEid 는 가상의 맨 오른쪽의 자식 노드의 Eid 이다. 그림 2 상태의 트리에서 그림 3 의 XML 문서와 같이 g 엘리먼트가 삽입되는 상황을 살펴 보면 RN-Scheme 의 갱신 연산 후의 번호 체계 고정화 작업이 어떻게 이루어지는지 알 수 있다. g 엘리먼트의 삽입이 이루어진 후의 트리인 그림 4 를 보면 삽입된 g 엘리먼트의 새로운 (Eid, RmdEid)의 값이 소수점을 가지게 되는 것을 확일 할 수 있다. 다음의 세가지 공식이 RN-Scheme 에서의 새로운 엘리먼트의 삽입이 이루어 지는 경우의 값을 할당해 주는 기법이다.

- g's (EID, RmdEid, NextChildEid) are set to (x,x,y)
  where x = (c's RmdEid + c's NextChildEid)/2
  and y = c's NextChildEid
- RmdEid of c is also
  set to (c's RmdEid + c's NextChildEid)/2
- NextChildEid of e is also
  set to (e's RmdEid + e's NextChildEid)/2

g 엘리먼트의 삽입 연산 과정은 (1) g 엘리먼트의 삽입, g 엘리먼트의 삽입으로 인한 (2) RmdEid 의 고정화 작업, (3) NextChildEid 의 고정화 작업 이렇게 3 단계의 과정으로 이루어 져있다. 삭제, 수정 연산의 경우에는 번호 체계에 전혀 영향을 주지 않기 때문에 고려 대상에서 제외 하겠다.

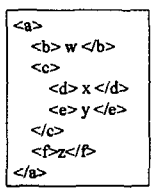

그림 1 XML 문서

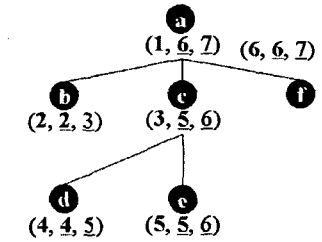

그림 2 XML 문서 트리

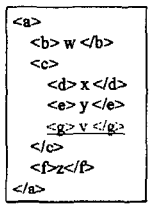

그림 3 XML 문서

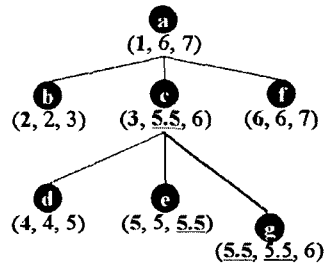

그림 4 XML 문서 트리

2.2. R-Scheme

R-Scheme 은 RN-Schem(RmdEid)에서 NextChildEid 를 제외한 번호 체계이다. NextChildEid 를 제외했기 때문에 번호 체계고정화 작업을 할 때, NextChildEid 에 대한 고정화 작업 부분이 사라진 셈이다. 하지만, 엘리먼트를 삽입할 경우 RmdEid 와 NextChildEid 를 사용하여 엘리먼트의 Eid 를 구하기 때문에 R-Scheme 에서 제외된 NextChildEid 를 찾는 루틴이 더해져서 삽입 연산이 이루어 지게 된다.

2.3. OS-Scheme

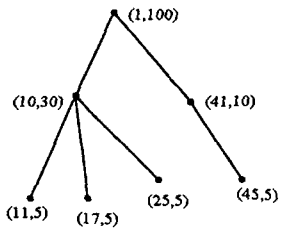

그림 5 Order Size Numbering Scheme

RN-Scheme 및 R-Scheme 과의 비교를 위해 [2][3] 의 OS-Scheme(Order, Size) 을 고려 하였다. OS-Scheme 은 RN-scheme, R-Scheme 과는 달리 OS-Scheme 은 번호 체계의 고정화 작업이 필요 없다. 그림 5 의 각 노드는

(Order, Size) 값으로 이루 졌다. Order 는 트리 노드의 순서를 나타내며, Size 는 각 트리 노드가 최대한으로 가질 수 있는 자식/후손의 개수를 의미 한다. 그래서, OS-Scheme 은 그림 5 와 같은 번호 체계를 사용하기 때문에 각 트리 노드의 Order 와 Size 값을 적절히 정해 주면, 엘리먼트 삽입 후에 번호 체계고정화 작업을 해주지 않아도 된다는 특징이 있다. 하지만, OS-Scheme 의 경우 Order 와 Size 의 값을 적절히 정해 주기 위하여 관련된 Order 와 Size 값들의 검색이 다소 많이 요구되는 문제점이 있다.

3. 구현 및 성능평가

XML 문서 갱신 연산을 전 절의 세가지 번호 체계에 대하여 Java 로 Windows 2000 Server 에서 구현하였다. XML Database system 은 XRel[1]을 기본으로 세가지 번호 체계에 대한 갱신 연산을 구현하였다. 성능 실험에 사용된 XML 데이터는 [4]의 셰익스피어 희곡 문서, [5]의 dblp 문서, [6]의 xbench 를 사용하여 만든 catalog 문서를 사용하였다.

- 셰익스피어 문서: 각각의 번호 체계가 문서의 수에 어떻게 반응하는지 알아 보기 위한 XML 문서
- DBLP 문서: 각각의 번호 체계가 문서의 크기에 어떻게 반응하는지 알아 보기 위한 XML 문서
- Catalog 문서: 삽입이 일어나는 문서 내부의 level(트리의 높이)을 다양하게 주어 각각의 번호 체계가 level 에 대해서 어떻게 반응하는지 알아 보기 위한 XML 문서

실험은 Pentium III 1GHz CPU 와 512MB 메모리를 장착한 시스템에서 DBMS 는 Oracle9i 를 사용하여 수행하였다. 평가 결과는 다음과 같다 (그림 6-8 참조).

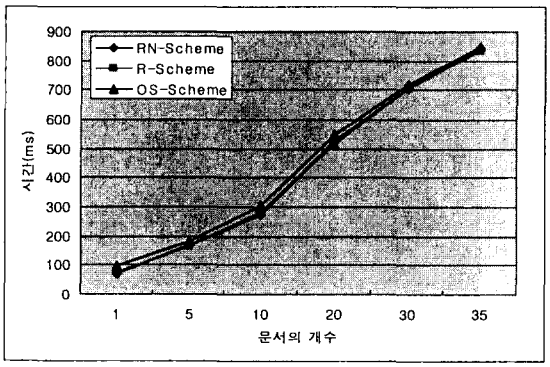

그림 6. XML 문서의 개수에 따른 삽입 시간

그림 6 을 보면 문서의 수가 증가함에 따라 모든 번호 체계의 삽입 연산 처리 시간이 증가하지만, 각 번호 체계의 별다른 시간 적 차이는 없다. 이때 사용한 셰익스피어 희곡 문서는 평균 600K 의 크기를 가진 XML 문서 였다.

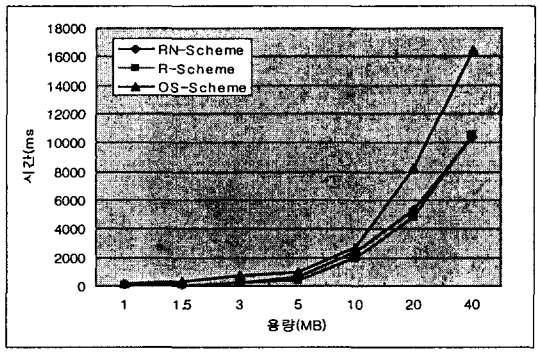

그림 7. XML 문서의 크기에 따른 삽입 시간

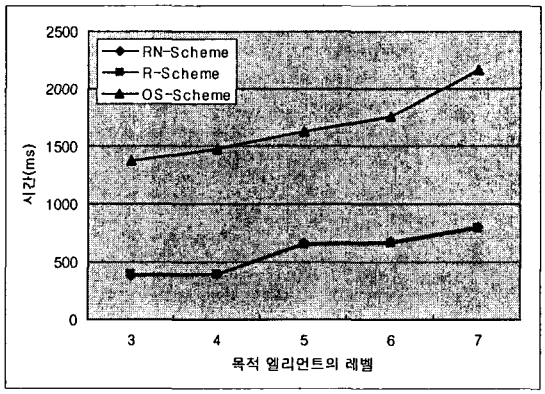

그림 8. 삽입 연산이 일어나는 엘리먼트의 레벨에 따른 삽입 시간

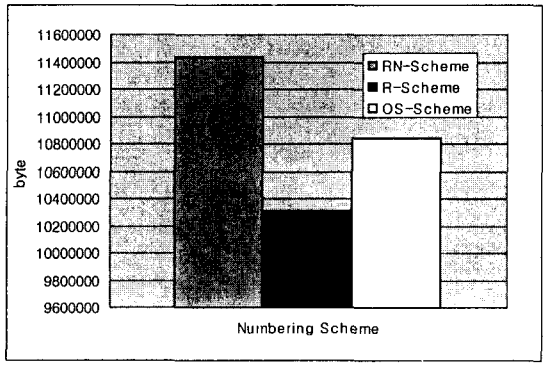

그림 9. 각 번호 체계의 RDB 내에서의 데이터 크기

그림 7 의 문서의 용량을 달리하여 실험한 결과를 보면 OS-Scheme 의 경우 10MB 이상의 XML 문서부터는 RN, R-Scheme 보다 삽입 연산의 성능이 떨어지는 것을 볼 수 있다. 이런 현상을 보이는 이유는 RN, R-Scheme 의 경우 RmdEid, NextChildEid 를 사용하여 삽입 되는 엘리먼트의 정보를 할당해 주지만, OS-Scheme 의 경우 삽입이 된 후의 번호 체계의 고정화 작업은 없지만, 삽입될 곳을 찾는 과정과 삽입되는 엘리먼트의 Order 나 Size 의 알맞은 값을 할당해 주는 부분에서 문서의 크기가 커질수록 부담이 더해

지기 때문이다. 그림 8 의 경우는 문서의 크기는 고정을 하고 삽입되는 엘리먼트의 부모가 되는 엘리먼트의 레벨을 달리하여 삽입 할 경우의 결과이다. RN, R-Scheme 의 경우 삽입 이후 고정화 작업시간이 레벨이 낮은 경우보다 레벨이 높은 경우가 더 높기 때문에 그림 8 과 같은 결과가 나왔다. 그리고, RN 과 R-Scheme 의 성능차이가 거의 보이지 않는 이유는 R-Scheme 의 경우 번호 체계의 고정화 작업을 한 단계 줄이기는 하였지만, NextChildEid 를 찾는 시간이 부가적으로 들기 때문에 RN, R-Scheme 이 성능의 차이가 거의 없는 것이다. 하지만, 그림 9 를 보면 7MB 의 Catalog 문서가 RDB 내에서 차지하는 공간의 크기를 나타낸 그림이다. R-Scheme 의 경우 RN-Scheme 의 NextChildEid 를 제거 하였기 때문에 RDB 상에서 차지하는 공간의 크기가 가장 적은 것으로 나왔다. OS-Scheme 의 경우는 Order 의 값이 순차적이지 않고 일정한 사이를 둔다. 따라서 문서의 크기가 커질수록 Order 의 값이 기하급수적으로 증가하기 때문에 RDB 상에서 Order 컬럼의 길이가 늘어나 R-Scheme 보다는 공간을 더 많이 차지하였다. 이상 실험의 결론을 말하면 R-Scheme 이 삽입시간 및 공간 양 측면에서 가장 효율적이다.

4. 결론 및 향후 연구

본 논문에서는 XML 데이터베이스 기반에서의 효율적인 문서의 갱신 연산을 지원해 주기 위한 번호 체계인 RN-Scheme 과 R-Scheme 을 제안하고 구현한 후 그 성능을 [2][3]의 OR-Scheme 과 비교, 평가하였다. 비교 결과 본 논문에서 제시한 R-Scheme 이 가장 우수한 것으로 나타났다.

향후 연구 과제는 다음과 같다. 본 논문에서는 단말 엘리먼트의 삽입 만을 다루었는데, 이를 임의의 XML 엘리먼트(즉, XML 서브트리)를 삽입할 때의 경우로 확장하는 연구가 필요하다.

참고문헌

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

[2] S. Chien et al., "Storing and Querying Multiversion XML Documents using Durable Node Numbers," Proc. 2nd Int'l Conf. on Web Information Systems Engineering, 2001, pp. 232-241.

[3] Q. Li and B. Moon, "Indexing and Querying XML Data for Regular Path Expressions," Proc. Int'l Conf. on VLDB, 2001.

[4] I. Tatarinov et al., "Updating XML," Proc. ACM SIGMOD Int'l Conf. on Management of Data, 2001, pp. 413 - 424

[5] http://metalab .unc.edu/bosak/xml/eg/shaks200.zip

[6] http://www.informatik.uni-trier.de/~ley/db/

[7] http://db.uwaterloo.ca/~ddbms/projects/xbench

[8] http://xmldb-org.sourceforge.net/xupdate