CSB^+-Tree 인덱스를 이용한 DTD가 없는 Well-Formed XML 문서 저장 기법 *

신성현[0] 김진호
강원대학교 컴퓨터과학과
{shshin, jhkim}@mail.kangwon.ac.kr

Design for Storaging Well-Formed XML Data without DTD using CSB^+-Tree Index

Sung-Hyun Shin[0] Jin-Ho Kim
Dept. of Computer Science, Kangwon National University

요 약

최근 인터넷 상에서 정보 양이 증대함에 따라 DTD 기반의 XML을 이용하여 방대한 정보를 효율적으로 저장하고 검색하기 위한 많은 연구들이 진행되고 있다. 하지만 DTD의 문서를 작성에는 많은 노력이 필요하고, 문서 구조의 검사가 필요없는 간단한 문서만 사용하는데 있어서 DTD 작성은 불필요한 작업이 아닐 수 없다. 하지만 DTD 문서가 없을 경우에도 XML 기본 문법만 맞으면 문서를 파싱할 수 있고, 불필요한 네트웍 부담을 줄이기 위해 DTD없이 전송할 수 있다. 따라서 본 연구에서는 DTD가 없는 Well-Formed XML 문서를 구성하는 엘리먼트의 구조를 통해 정보를 추출하고, 주기억장치의 효율적인 저장 공간을 활용한 CSB^+-Tree 인덱스를 이용하여 Well-formed XML 문서를 저장하기 위한 기법을 제안한다.

1. 서론

최근 고도의 정보화 사회로 발전함에 따라 대량의 정보 관리와 데이터 요청에 빠르게 응답할 수 있는 시스템이 필요로 하게 되었다. 따라서 HTML과 SGML의 장점을 포함한 XML(eXtensible Markup Language)이 표준으로 등장하면서 효율적인 저장과 검색 관리 시스템에 대한 연구가 진행되고 있다[1][2].

XML은 DTD(Document Type Definition)를 통해 문서 자체에 문서의 구조를 기술하고 있다[3]. 그리고 문서의 구조를 사용자가 원하는 대로 정의할 수 있으며, 이러한 구조적인 특성은 다양한 형태의 데이터를 XML로 기술될 수가 있다.

하지만 DTD의 작성에는 많은 노력이 필요하다. 문서 구조의 검사가 필요없는 간단한 문서만 사용하는 사용자에게 있어서 불필요한 작업이 될 수 있다. 그래서 XML은 DTD를 임의적으로 선택할 수 있게 되었다. 간단한 문서는 DTD가 없을 경우에도 XML의 기본적인 구문 규칙만 따르면 문서를 파싱할 수 있도록 정의하고 있으며, 이러한 문서를 Well-Formed XML 문서라 한다. Well-Formed이라는 것은 어떤 문서가 하나의 XML 문서로 간주되는데 필요한 최소한의 필수 조건들의 집합을 의미한다[3]. XML 문서는 웹을 위해 설계된 것이며 Well-formed 문서가 의미있게 사용될 수 있는 상황이기만 하면, DTD를 굳이 전송하거나 문서에 포함시키지 않아도 된다. 따라서 네트웍 부담을 줄일 수 있다.

XML 문서를 효율적으로 저장하기 위한 기존의 연구로는 DTD가 존재하는 환경에서 XML 문서의 논리적인 구조 표현으로 기존의 데이터베이스에 저장 및 검색에 대한 연구[4]가 있으며 계층적 구조를 가진 DTD를 이용하여 XML 데이터베이스 시스템의 개발에 대한 연구[5]가 진행되고 있지만 DTD를 통해 XML 데이터를 대상으로 연구되었기 때문에 저장하려는 방법에 대해 연구되었기 때문에 DTD가 없는 XML 문서의 의미와 구조적 정보 표현에 대해서는 한계점을 나타내고 있다.

그래서 본 연구에서는 DTD가 없는 Well-formed 문서를 저장하는 방법으로 XML 문서를 구성하는 단위인 엘리먼트를 저장하기 위해 정보를 추출하고 이를 이용하여 주기적장치의 효율적인 저장 공간을 활용한 CSB^+-Tree 인덱스[6]를 이용하여 생성 저장하는 기법에 대해 제안한다.

본 논문은 2장에서 관련 연구에 대해 살펴보고 3장에서는 Well-formed의 기본 사항을 준수하여 인덱스에 생성 저장하는 기법에 대해 설명하고 4장에서는 결론을 맺는다.

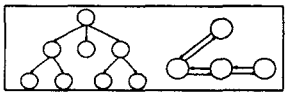
2. 관련 연구

관계형 데이터베이스 시스템에 XML 문서를 저장하기 위한 방법으로 번호 부여 기법(numbering Schema)[7]이 있으며 이 방법은 XML 문서의 계층 구조를 이용하여 부모와 자식 엘리먼트 간의 관계 정보를 알 수 있다. 또한 엘리먼트의 고유 식별자를 나타내는 ETID(Element Type ID), XML 문서에서 형제 엘리먼트들의 순서정보인 SORD(Sibling ORDer), 같은 타입의 엘리먼트 형제에 대한 순서정보인 SSORD(Same Sibling ORDer)로 구성하여 저장되는 연구도 있다[8]. 그러나 XML 문서를 저장할 경우 DTD가 반드시 존재해야 한다. 즉, DTD 문서를 통해 XML 문서의 엘리먼트 정보를 파악하여 XML 데이터를 저장한다. 따라서 DTD가 없는 환경에서는 Well-formed XML 데이터를 저장할 수 없는 문제점이 발생한다.

XML 문서의 구조에 대한 정보를 나타내는 인덱스는 대부분 그래프 형태로 구성되어 있다. 즉, DTD 기반의 XML 문서가 기본적으로 DTD 그래프 구조로 되어있다. 하지만 [그림 1]과 같이 그래프 구조는 하나의 부모 노드가 같은 레벨에 존재한 다수의 자식 노드를 가지므로, 리프 노드(leaf node)가 아닌 노드들은 다수의 포인터를 유지하고 있어야 한다. 기존 연구에서는

* 이 논문은 첨단정보기술연구센터(AITrc)를 통하여 한국과학재단의 지원을 받았음.

자식 노드의 포인터가 차지하는 공간을 절약하기 위해 첫 번째 자식 노드로의 포인터와 왼쪽, 오른쪽 형제 노드로의 포인터, 부모 노드의 포인터만을 유지하는 방법이 있다[9]. 이 방법은 자식 노드의 수와 관련 없이 하나의 노드는 4개의 포인터만 유지하면 된다. 하지만 노드 당 4개의 포인터가 존재하므로 여전히 저장 공간의 낭비가 생긴다.



[그림 1] 기존의 DTD 그래프

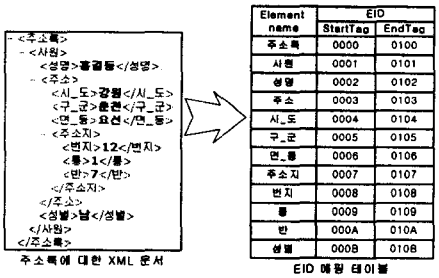
이러한 문제점을 해결하기 위해 저장 공간의 효율성을 높이기 위한 방법으로 CSB$^+$-Tree가 연구되었다[6]. CSB$^+$-Tree는 B$^+$-Tree 구조와 유사한 구조로 되어 있으며, 디스크 기반의 시스템에 페이지 구조와 같이 CSB$^+$-Tree 기반의 시스템에서는 기본 전송 단위가 캐쉬 라인이다. 기존의 B$^+$-Tree는 자식 노드 내의 키 값과 키 값에 대응되는 포인터들을 명시적으로 저장하기 때문에 저장 공간의 오버헤드가 크다. 하지만 CSB$^+$-Tree는 같은 레벨에 위치한 자식 노드들을 그룹으로 묶고 그 첫 번째 자식 노드의 주소 값만을 명시적으로 저장하기 때문에 키들을 위한 저장 공간을 더 많이 가질 수 있다. 그러므로 저장 공간에 따른 오버헤드가 감소하게 되고 캐쉬의 성능은 더욱 향상된다.

3. Well-formed XML 문서 저장 기법

3.1 XML 문서의 엘리먼트 식별 부여

XML 문서는 하나의 문서에 구조 정보와 내용을 포함하고 있다. 그래서 입력받은 XML 문서를 파싱 과정을 통해 엘리먼트의 구조를 분석한다.

XML 문서의 논리적 구조 정보를 통해 각 엘리먼트의 임의의 고유한 식별자(EID)를 [그림 2]와 같이 할당한다. [그림2]에서는 EID를 4바이트로 태그 정보, 엘리먼트의 순서 정보를 나타낸다. 상위 2바이트는 엘리먼트가 해당하는 태그가 시작 태그('00')인지 종료 태그('01')인지에 대한 정보를 나타내며, 하위 2바이트는 파싱 과정에서 ASCII 코드의 순서로 부여된 엘리먼트의 고유한 순서 정보를 표현한다.

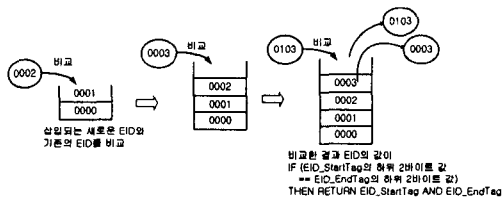


Element name	EID	
	StartTag	EndTag
주소록	0000	0100
사원	0001	0101
성명	0002	0102
주소	0003	0103
시_도	0004	0104
구_군	0005	0105
면_동	0006	0106
주소지	0007	0107
번지	0008	0108
통	0009	0109
반	000A	010A
성별	000B	010B

[그림 2] EID 부여의 예

3.2 스택을 이용한 EID 저장

Well-formed의 조건은 XML의 기본적인 문법을 준수하여야 하며 가장 기본적인 조건은 전체 XML을 구성하는 루트 노드는 단 하나이어야만 한다는 것이다. 이러한 XML 문서의 엘리먼트는 구조적인 특성 때문에 임의적으로 삽입 연산이 될 수 없으므로 엘리먼트를 인덱스에 저장하기 위해 임시 저장공간인 스택을 [그림 3]과 같이 이용한다.



[그림 3] 스택을 이용한 EID_StartTag와 EID_EndTag 값들의 비교

우선, 스택에 저장된 엘리먼트 EID를 저장하고 후에 들어오는 엘리먼트 EID를 비교하여 반환한다. 엘리먼트의 EID를 비교할 때는 EID의 하위 2바이트를 비교하여 같은 엘리먼트 태그임을 인지하고 시작태그와 종료태그의 여부를 나타내는 상위 2바이트를 비교한다. 이때, 전후의 비교하여 에러가 발생할 경우 XML 문서가 Well-formed의 조건에 위배되었음을 알 수 있다.

3.3 CSB$^+$-Tree 인덱스 구조

스택을 통해 엘리먼트를 저장하기 위한 인덱스는 주기억장치를 효율적으로 활용하기 위해 캐시를 고려한 B$^+$-tree(Cache Sensitive B$^+$-Tree)[6]를 변형하여 사용한다. 이 구조는 기존의 B+-Tree보다 주기억장치에 저장할 수 있는 키 값의 저장 개수를 증가시킬 수 있으며, 노드 분할 시 모든 자식 노드들의 포인터들도 변경을 시켜줘야 하는 오버헤드를 감소시킬 수 있다.

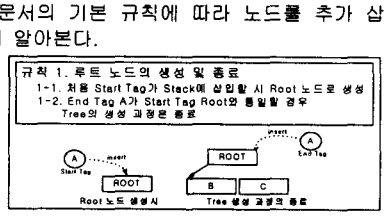
그러나 CSB+-Tree 인덱스 구조는 하나의 부모 엘리먼트의 다수의 자식 엘리먼트들이 저장되는 노드들을 하나의 그룹으로 묶고 그 첫 번째 자식 노드의 주소값을 명시적으로 표현하여 저장한다. XML 문서의 엘리먼트를 저장하기 위한 인덱스의 노드 구성은 다음과 같다.

- nEKey : 노드 내에 있는 키(엘리먼트)의 개수
- EKeyList[2d] : 키(엘리먼트의 UID)의 값
- FirstChild : 자식 노드 그룹에 있는 첫 번째 자식 노드의 주소값

CSB$^+$-Tree 인덱스를 이용하는 것은 이 명시적인 자식 노드의 첫 번째 자식 노드의 주소를 유지하는 포인터가 하나뿐이므로 주기억장치의 저장 공간의 유용성이 높다. 같은 레벨에 있는 노드 그룹이 모두 물리적으로 인접한 공간에 저장되므로 그룹 내의 나머지 자식 노드들은 FirstChild 주소에 옵셋 값을 더하여 주소를 알아낼 수 있다. 그래서 추가적인 XML 문서의 특정 엘리먼트가 삽입과 검색 연산에 대하여 유연하게 대처할 수 있다.

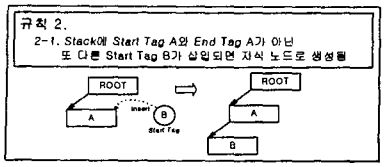
3.4 인덱스 저장 규칙

본 절에서는 CSB$^+$-Tree 인덱스를 이용하여 노드를 생성하고 XML 문서의 기본 규칙에 따라 노드를 추가 삽입하는 규칙에 대해 알아본다.

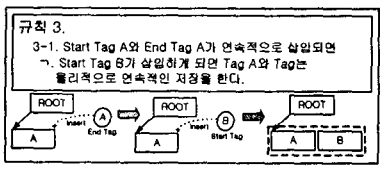


규칙 1의 1-1에서 언급된 내용은 인덱스가 생성할 때는 XML 문서에서 처음 파싱된 Start Tag로부터 노드가 생성된다. 예를 들어, 엘리먼트의 EID인 '0000'값이 스택에 삽입되면 인덱스의 루트 노드로부터 생성하게 된다. 또한 1-2에서는 시작 태그로부터 루트노드가 생성한 후 종료 태그의 값인 '0100'와 루트 노드의 하위 2바이트 값을 서로 비교하여 동

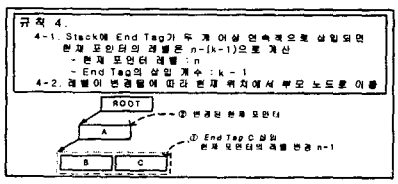
일하면 인덱스의 생성 과정은 종료가 되는 것이다.



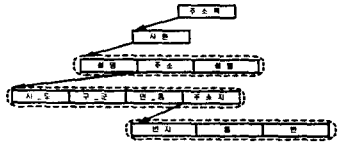
규칙 2에서는 스택에 저장된 엘리먼트의 EID 중에 Start Tag 값이 연속으로 저장하게 되면 인덱스에서는 후에 들어오는 EID의 Start Tag 값은 자식 노드를 생성하게 된다. 예를 들어, 엘리먼트의 EID의 STart Tag 값들이 연속적으로 '0000', '0001', '0002' 순으로 들어게 되면' '0000'은 루트 노드로 , '0001'은 자식노드로, '0002'은 손자노드로 각각 생성하게 된다.



규칙 3에서는 엘리먼트의 EID 중에 Start Tag와 End Tag가 연속적으로 스택에 삽입하게 되면 형제 노드로 생성되며 그룹으로 묶는다. 이때 물리적으로 인접하게 저장을 하게 된다. 예를 들어, EID의 Start Tag와 End Tag인 '0005'와 '0105', '0006'와 '0106'이 연속으로 삽입되면, 자식노드인 '0005'가 생성되며 형제노드인 '0006'이 물리적으로 인접하게 노드가 생성하게 된다.



규칙 4에서는 두 개 이상의 엘리먼트의 EID가 연속적으로 삽입할 때 현재 포인터의 위치 이동을 계산하는 방법을 나타내고 있다. 연속적으로 스택에 EID가 삽입될 경우 포인트의 위치를 이동시켜 다음 EID의 삽입시 대처할 수 있다. 예를 들어, 마지막으로 삽입된 노드의 값 '0006'의 위치에 포인터가 존재하고 있으며 포인터의 위치를 이동시키기 위해서는 n-(k-1)식으로 계산하여 포인터의 위치를 이동시킨다. 여기서 n은 현재 포인터가 위치하고 있는 레벨의 값을 나타내고 있으며, (k-1)에서 k는 EID 중의 End Tag의 개수를 나타내고 있다.



[그림 8] CSB+-Tree 인덱스를 이용한 엘리먼트 저장

위의 제안된 규칙에 따라 결과적으로 생성된 인덱스는 [그림 8]처럼 표현되었다. 이로써, 인덱스의 노드에 저장하기 위해 스택을 이용하였고, 위에 정의된 규칙을 적용시키면 인덱스의 삽입 연산에 대해 유연하게 대처할 수 있을뿐만 아니라, XML 문서가 하나 이상의 관련된 문서들도 간단히 통합 문서로서 저장할 수 있다.

VI. 결론 및 추후 연구

기존 연구에서는 XML 문서를 효율적인 관리 및 검색에 대한 내용으로 연구를 하였다. 하지만 기존 연구에서는 XML 문서를 저장 관리하기 앞서 DTD를 이용하여 문서를 구성하는 기본 단위인 엘리먼트의 정보를 추출하기 때문에 DTD가 없는 XML 문서의 데이터를 저장하기 위한 연구에 대해서는 희박하다. 또한 XML 문서에 대하여 인덱싱 기법을 논하지 않았거나 디스크 기반의 저장 시스템에서 이루어졌기 때문에 응답 처리의 속도를 높이기엔 한계점을 나타내고 있다.

본 논문에서는 DTD가 없는 Well-formed XML 문서를 구성하는 단위인 엘리먼트의 구조적인 특성을 이용하여 주기억장치의 저장 공간을 효율적으로 활용한 CSB^+-Tree 인덱스에 엘리먼트를 저장하는 기법을 제안하였다. 제안된 기법을 통하여 사용자가 요구하는 XML 데이터를 빠른 응답 처리를 통해 만족시켜 줄 수 있을 뿐만 아니라 삽입, 삭제, 검색 연산에 대해서도 유연하게 대처할 수 있다.

추후 연구로는 CSB^+-Tree 인덱스를 이용한 XML 문서를 효율적으로 저장할 수 있는 지에 대한 성능평가가 필요하며, XML 문서에서의 엘리먼트 뿐만 아니라 엘리먼트의 유형이나 속성들도 같이 추출하여 인덱스에 저장할 수 있는 설계 기법이 필요하다.

참 고 문 헌

[1] Jennifer Widom, "Data Management for XML," IEEE Data Engineering Bulletin Special issue on XML, Vol. 22, No. 3, pp. 44-52, Septemer 1999.

[2] Stefano Ceri, Piero fraternali and Stefano Paraboschi, "XML: Current Developments and Future Challenges for the Database Community," Proc. of the 7th Int. on EDBT, pp. 3-17, March, 2000.

[3] T. Bray, J. Paoli, and C. M. Sperberg-McQueen, " Extensible Markup Language (XML) 1.0 (Second Edition)," http://www.w3.org/TR/REC-xml, W3C Recommendation, October, 2000.

[4] Takeyuki Shimura, Masatoshi Yoshikawa, Shunsuke Uemura, "Storage and Retrieval of XML Document Using Objected-Relational Database," DEXA99, pp. 206-217, 1999.

[5] C. Kanne and G. Moerkotte, "Efficient Storage of XML Data," Proc. of Int. Conf. on Data Eng., 1998.

[6] Jun Rao, Kenneth A. Ross. " Making B^+-Trees Cache Conscious in Main Memory." In MOD 2000, Dallas, TX USA.

[7] Q. Li and Bongki Moon. "Indexing and Querying XML Data for Regular Path Expressions," Proc. of the 27th VLDB Conference, 361-370, September 2001.

[8] 박종관, 강형일, 송충범, 유재수, "XML 문서에 대한 효율적인 구조 기반 검색을 위한 색인 모델," 한국정보과학회, 2000 추계 학술 발표 논문집 pp. 18-20, 2000.

[9] Jae-Woo Chan. "An XML Document Retrieval System Supporting Structure- and Content-Based Queries," ER Workshops, 2001.