韓國컴퓨터情報學會 論文誌
第16卷 第4號, 2011. 04.

2011-16-4-4-1

동적 XML 데이터 관리를 위한 트리 분해 기반의 소수 레이블링 기법

변 창 우*

A Prime Number Labeling Based on Tree Decomposition for Dynamic XML Data Management

Chang-Woo Byun*

요 약

갱신 연산의 허용으로 동적 XML 데이터의 처리 효율성의 요구가 증대하면서 새로운 동적 XML 레이블링 기법들이 연구되어 왔다. 동적 XML 레이블링 기법의 핵심적인 해결 사항으로는 조상-자손-형제 관계 결정, 레이블 저장공간의 절약, 빠른 응답시간, 갱신에 의한 레이블 재작성의 최소화이다. 대표적인 동적 레이블링 기법으로 소수 기반 레이블링 기법이 있다. 소수 기반 레이블링 기법은 소수의 특성을 이용하여 조상-자손 관계를 쉽게 결정한다. 또한 새로운 엘리먼트를 삽입할 때도 기존 노드의 레이블을 재작성하는 비용이 발생되지 않는 장점을 갖고 있다. 하지만 소수를 많이 사용하면 레이블의 값이 상당히 커지게 되는 레이블 오버플로우 문제가 발생된다. 본 논문에서는 레이블 오버플로우 문제를 효과적으로 줄이는 새로운 방법을 소개한다. 제안하는 방법의 핵심 개념은 트리 분해이다. 레이블 오버플로우가 발생하면 트리를 하부 트리들로 분해하고 레이블은 각 하부 트리에 한해서 부여하는 것이다. 실험을 통해 트리 분해 기반의 소수 기반 레이블링 기법의 효과를 보인다.

▸ 키워드 : 동적 XML, 소수 기반 레이블링, 레이블 재작성, 트리분해

Abstract

As demand for efficiency in handling dynamic XML data grows, new dynamic XML labeling schemes have been researched. The key idea of the dynamic XML labeling scheme is to find ancestor-descendent-sibling relationships and to minimize memory space to store total label, response time and range of relabeling incurred by update operations. The prime number labeling scheme is a representative scheme which supports dynamic XML documents. It determines the ancestor-descendant relationships between two elements by a simple divisibility test of labels. When a new element is inserted into the XML data using this scheme, it does not change the label values of existing nodes. However, since each prime number must be used exclusively, labels can become significantly large. Therefore, in this paper, we introduce a novel technique to effectively reduce the problem of label overflow. The suggested idea is based on tree decomposition. When label overflow occurs, the full tree is divided into several sub-trees, and nodes in each sub-tree are separately labeled. Through experiments, we show the effectiveness of our scheme.

▸ Keyword : Dynamic XML, Prime number labeling, Relabeling, Tree composition

• 제1저자 : 변창우 • 교신저자 : 변창우
• 투고일 : 2011-02-07, 심사일 : 2011-02-18, 게재확정일 : 2011-02-21
* 인하공업전문대학 컴퓨터시스템과(Dept. of Computer Systems and Engineering, Inha Technical College)
※ 이 논문는 2009학년도 인하공업전문대학 교내연구비지원에 의하여 연구되었음.

www.kci.go.kr

Ⅰ. 서 론

XML(eXtensible Markup Language)[1]의 구성요소인 엘리먼트, 속성, 텍스트 데이터를 트리의 노드로 모델링을 하는 트리 모델로 변형하여 XML 데이터를 저장하는 방법이 일반적이다[1-2]. 대표적인 연구로는 XML 트리의 각 노드에 레이블(label)을 부여하고 레이블 정보를 이용하여 조상, 자손, 형제 관계를 파악할 수 있는 색인(index) 기반의 인코딩 기법으로 레이블링 방법에 대한 연구가 있다[3-10].

XML 문서가 갱신(삽입, 삭제, 갱신 등)되는 환경을 고려하지 않은 정적 레이블링 기법(static labeling scheme)은 갱신을 허용하는 동적인 XML 데이터 환경에 그대로 적용하는 데는 다수의 기존 노드들의 레이블을 재작성해 주어야 하는 문제점이 있어 무리가 따른다[3].

조상, 자손, 형제 관계 파악의 용이성, 저장 공간의 최소성, 기존 노드의 레이블 재작성의 최소화를 목표로 연구된 동적인 XML 데이터 환경에서의 레이블링 기법으로는 범위 기반 레이블링 방법[3], 접두사 기반 레이블링 기법[4-6], 소수 기반 레이블링 방법[7-9], 섹터 기반 레이블링 방법[11] 등이 있다. 그러나 기존 연구들의 갱신 비용은 저장 공간 및 레이블 재작성에 여전히 부담을 주고 있다.

본 논문에서는 트리 분해 기반의 효율적인 소수 레이블링 기법을 제안하며 다음과 같은 장점을 갖고 있다.

- 소수 레이블링 기법 사용으로 조상, 자손, 형제 관계를 파악하는 처리 시간의 최소화
- 트리 분해를 통해 레이블 크기의 최소화에 따른 저장 공간의 절약
- 갱신 부분의 레이블링 용이성 및 기존 다른 노드의 레이블 재작성(relabeling) 최소화

본 논문의 구성은 다음과 같다. 2장에서 관련연구를 소개하고 3장에서 트리 분해 기반의 소수 레이블링 기법을 제시하며 조상, 자손, 형제 관계를 쉽게 파악할 수 있는 방법 및 갱신 환경에서 제안하는 레이블링 기법의 처리과정을 설명한다. 4장에서는 논문에서의 제안 방법에 대한 실험결과를 토대로 평가를 설명하고, 5장에서 결론을 기술한다.

Ⅱ. 관련 연구

1. 정적 레이블링 기법

XISS(XML Indexing and Storage System)에서는 레이블로 (order, size)를 사용한다[3]. order는 XML 데이터 내에서 엘리먼트의 순서이고 size는 자손으로 가질 수 있는 엘리먼트들에 할당될 수 있는 범위이다. A와 B 노드의 레이블이 (orderA, sizeA), (orderB, sizeB)라 할 때 orderA < orderB < orderA + sizeA 를 만족하면 A는 B의 조상 노드이다. Size 값을 이용해 새로 삽입되는 데이터에 기존 데이터의 레이블을 재작성하지 않고 레이블을 할당할 수 있기 때문에 동적 XML 환경에 대한 고려를 했다고 할 수 있다. 그러나 해당 위치에 새로운 데이터를 삽입하고자 할 때, 미리 할당된 여분의 범위가 모두 소진되면 다수의 기존 노드들에 대한 레이블 재작성이 필요하게 되는 문제점이 발생한다.

2. 동적 레이블링 방법

2.1 접두사 기반 레이블링 방법

LSDX는 숫자와 문자의 조합을 레이블에 사용한다[5]. LSDX의 레이블은 XML 트리 내에서 해당 노드의 레벨(level) 정보와 부모 노드의 레이블과 해당 노드의 셀프 레이블, 이렇게 3차원 인덱스로 구성된다. [그림 1]에서 1a.b는 0a를 갖는 노드의 자식노드임을 알 수 있고, 1a.b와 1a.z는 같은 부모를 가지며 셀프레이블인 b와 z 중 b가 z보다 앞서기 때문에 1a.b가 1a.z보다 전임자 노드임을 알 수 있다. [그림 1]에서 새로운 노드 A가 삽입되면 1a.b 레이블을 갖는 노드 앞에 기준이 되는 노드의 셀프 레이블 b보다 빠른 순서를 갖는 문자 a를 붙여 레이블링 함으로써 조상, 자손, 형제 관계 파악은 그대로 적용될 수 있다. LSDX는 깊이가 깊어질수록 레이블 길이의 증가 및 저장 공간의 문제점이 있으며, 새로운 노드가 지속적으로 동일한 자리에 삽입될 경우 레이블의 길이가 길어지는 문제점뿐만 아니라 [그림 1]에서 B와 C 노드가 동일한 레이블을 갖게 되어 충돌이 발생함을 알 수 있다.

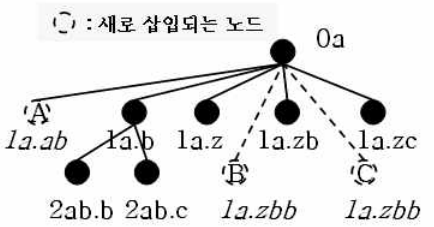


그림 1. LSDX 레이블링 기법
Fig. 1. LSDX labeling

www.kci.go.kr

2.2 소수 레이블링 방법

소수 기반 레이블링 방법은 소수의 특성을 사용한다[7]. XML 트리의 각 노드에 셀프 레이블로 소수(prime number)를 할당하고 각 노드의 실제 레이블은 부모 노드의 실제 레이블과 자식 노드의 셀프레이블의 곱이다. [그림 2]에서 노드 u와, v의 레이블은 2와 10이다. 이때 "label(v) mod label(u) = 10 mod 2 = 0"이기 때문에 u는 v의 조상 노드이다.

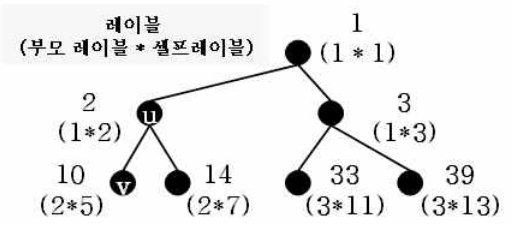


그림 2. 소수 레이블링 기법
Fig. 2. Prime number labeling

이와 같은 방법은 XML 문서의 크기가 커질수록 더 큰 소수가 필요하고 더 큰 두 소수의 곱이 생성되기 때문에 저장 공간의 확장성에 문제를 안고 있다. 또한, 삭제 노드에 사용되었던 소수를 재사용하지 못하는 단점이 있다.

그래프 모델을 기반으로 소수 레이블링 기법에 대한 연구도 있다[8]. 부모-자식-형제 관계를 쉽게 알 수 있도록 다음과 같은 3차원의 인덱스 구조를 제안하고 있다:

L(v) = (P(v), Ca(v), Cp(v))

P(v)는 셀프 레이블, Ca(v)는 조상 노드들의 Ca(v')와 자신의 셀프 레이블의 곱, Cp(v)는 부모 노드들의 셀프 레이블 곱을 나타낸다.

두 노드 u와 v에 대한 조상-부모-형제 관계는 다음과 같은 계산을 통해 알 수 있다.

- u가 v의 조상 관계: Ca(v) mod P(u) = 0
- u가 v의 부모 관계: Cp(v) mod P(u) = 0
- u와 v의 형제 관계: gcd(Cp(u), Cp(v)) ≠ 1, gcd는 최대공약수

그러나 임의의 노드의 깊이가 깊거나 너비가 넓은 경우 미리 정한 크기에 대해 오버플로우가 발생하는 문제가 있다. 결국 저장 공간의 확장성, 삭제 노드에 사용되었던 소수를 재사용하지 못하는 단점이 있다[9].

III. 트리 분해를 통한 하향식 레이블링 기법

1. 하향식 레이블링 및 트리 분해

앞으로의 모든 절에 이용되는 노드의 레이블 표현은 다음과 같은 2차원 인덱스 표현법을 사용한다(v' 노드는 v 노드의 부모):

L(v) = (L1(v), L2(v))

- L1(v)는 v가 단말노드인 경우는 상호 배타적 소수 할당
- L2(v)는 부모의 L2(v')와 자신의 L1(v)의 곱(단, 루트 노드의 L2 값은 2로 할당)

기본적인 레이블 할당 방법은 [그림 3]처럼 XML 트리에 하향식으로 루트노드부터 단말노드로 이동하고, 동일 레벨에서는 왼쪽에서 순차적으로 할당하는 것을 가정한다. 임의의 노드 x의 L2 값은 부모노드들의 L2 값에 자신의 L1 값의 곱이기 때문에 급격한 L2값의 증가가 나타나고 결국 오버플로우가 발생한다. 만약 L2의 레이블 크기를 1바이트로 가정했을 때 [그림 3]에서 노드 N7~N15는 오버플로우가 발생한다.

이와 같은 레이블 오버플로우 문제를 해결하기 위해 트리 분해 기법을 제안한다. 트리 분해란 오버플로우가 일어났을 경우 오버플로우가 일어난 노드부터 새로운 가상의 루트노드([그림 4]의 v2, [그림 5]의 v3)를 두어 새로운 서브트리로 두는 것을 말한다. 또한 각 서브트리에서의 레이블 할당 방법은 동일하다.

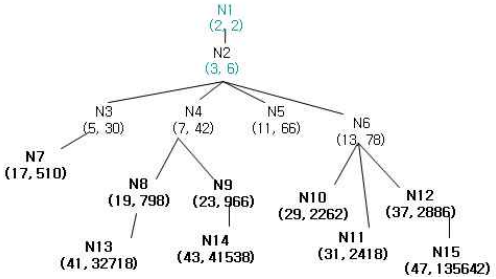


그림 3. 하향식 XML 트리 레이블링 및 레이블 오버플로우
Fig. 3. Top-down XML tree labeling and label overflow

www.kci.go.kr

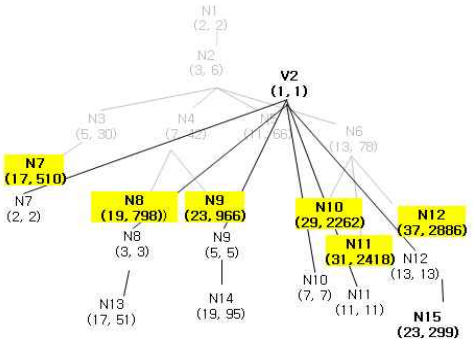


그림 4. 일차 트리분해에 의한 서브트리
Fig. 4. Sub-tree by the the first tree decomposition of Fig. 3

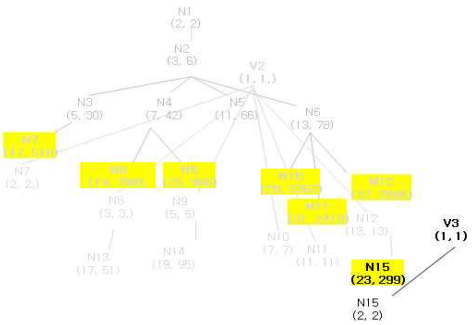


그림 5. 이차 트리분해에 의한 서브트리
Fig. 5. Sub-tree by the second tree decomposition of Fig. 4

각 노드에 레이블이 할당된 XML 트리를 LABEL 테이블로 저장한다. [그림 6]은 [그림 5]의 XML 트리를 나타내고 있다. SubG 컬럼은 서브트리의 가상노드를 나타내고, L1, L2, Node 컬럼은 노드의 레이블 L1, L2의 값 및 노드 이름을 나타낸다.

SubG	L1	L2	Node	Usable
1	2	2	N1	Y
1	3	6	N2	Y
1	5	30	N3	Y
1	7	42	N4	Y
1	11	63	N5	Y
1	13	78	N6	Y
2	2	2	N7	Y
2	3	3	N8	Y
2	5	5	N9	Y
2	7	7	N10	Y
2	11	11	N11	Y
2	13	13	N12	Y
2	17	51	N13	Y
2	19	95	N14	Y
3	2	2	N15	Y

그림 6. 그림 5에 대한 LABEL 테이블
Fig. 6. LABEL table of Fig. 5

이와 같은 상황에서 문제가 되는 것은 서브트리들 간의 연결고리이다. 서브트리들 간의 연결 노드를 bridge 노드라 하고 이들에 대한 정보를 BRIDGE 테이블로 저장한다. [그림 4]에서 bridge 노드는 N7, N8, N9, N10, N11, N12 이고, [그림 5]에서는 노드 N15이다. [그림 7]의 PVsubG컬럼은 bridge 노드의 부모노드의 서브트리의 가상노드, PVL1, PVL2컬럼은 부모노드의 L1, L2의 값, CVsubG컬럼은 bridge 노드의 서브트리의 가상노드, CVL2컬럼은 bridge 노드의 L2의 값을 의미한다.

PV_{subG}	PV_{L1}	PV_{L2}	CV_{subG}	CV_{L2}	Node
1	5	30	2	2	N7
1	7	42	2	3	N8
1	7	42	2	5	N9
1	13	78	2	7	N10
1	13	78	2	11	N11
1	13	78	2	13	N12
2	13	13	3	2	N15

그림 7. bridge 노드들에 대한 BRIDGE 테이블
Fig. 7. BRIDGE table of bridge nodes

2. 관계 파악

관계 파악은 포괄적으로 임의의 한 노드가 주어졌을 때 그것의 조상 노드들, 부모노드, 자손노드들, 자식노드들, 형제노드들을 파악하는 것을 의미한다. 본 절에서는 지면상 조상(부모) 관계 파악하는 방법에 대해 설명한다. 자세한 설명에 앞서 세 개의 함수를 소개한다.

- SUBG(n):임의의 노드 n에 대한 SubG 값 출력 함수
- L1(n):임의의 노드 n에 대한 L1 값 출력 함수
- L2(n):임의의 노드 n에 대한 L2 값 출력 함수

임의의 노드에 대한 조상노드를 파악하는 방법은 그 노드가 속한 서브트리에서 조상노드들을 탐색하고 그 후 상위의 서브트리에서 조상노드를 탐색해야 한다. 논문에서 제안하는 조상 관계 파악을 위한 SQL 질의 템플릿은 [그림 8]과 같다.

테이블 T1은 임의의 노드 n이 속한 서브트리에서의 자신을 포함한 조상노드들에 대한 정보를 나타낸다. 테이블 T3은 테이블 T1에서 부모노드 중 bridge 노드가 있는 경우를 파악하고 이 정보를 이용하여 조상노드를 파악하게 된다. 그 결과는 테이블 T2에 저장된다. 테이블 T1과 테이블 T2의 합집합에서 테이블 T1에 포함되어 있던 자신에 대한 정보를 빼면

www.kci.go.kr

최종적인 조상노드들에 정보가 출력된다.

노드 N8의 조상 관계 파악을 위해 SUBG(N3)=2, L1(N13)=17, L2(N13)=51을 조상 관계를 위한 SQL 질의 템플릿에 삽입하여 수행하면 [그림 9]처럼 테이블 T1, T3, T2, T4, T5가 생성되고, 합집합과 차집합을 통해 최종적으로 조상노드는 N8, N4, N2, N1임을 알 수 있다.

부모노드를 파악하는 템플릿은 조상 관계 파악을 위한 템플릿을 변형하면 되기 때문에 생략한다.

```
T1: Select Node, SubG, L2
    From Label
    Where SubG = SUBG(n) and L2(n) % L2 = 0;
Union
T2: Select Node, SubG, L2
    From Label, (select PVSubG, PVL2
                 from bridge, T1
                 where CVSubG = T1.SubG and CVL2 = T1.L2) T3
    Where Label.SubG = T3.PVSubG and T3.PVL2 % Label.L2 = 0
Union
T4: Select Node, SubG, L2
    From Label, (select PVSubG, PVL2
                 from bridge, T2
                 where CVSubG = T2.SubG and CVL2 = T2.L2) T5
    where label.subG = T5.CVSubG and T5.PVL2 % Label.L2 = 0
Minus
Select Node, SubG, L2
From Label
Where SubG = SUBG(n) and L2 = L2(n);
```

그림 8. 조상 관계 파악을 위한 SQL 질의 템플릿
Fig. 8. SQL template for finding ancestor nodes

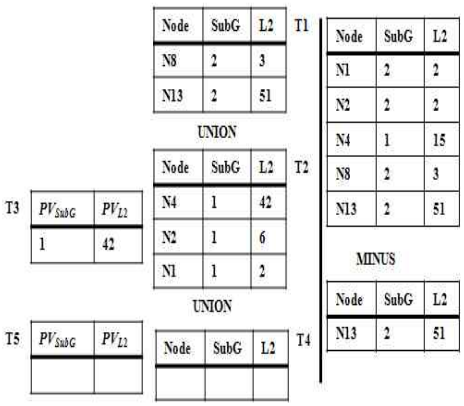


그림 9. 노드 N13의 조상 관계 파악
Fig. 9. Example of finding Ancestor nodes of node N13

3. 데이터 갱신에 대한 영향 분석

[그림 10]의 삭제 SQL 질의 템플릿을 통해 삭제되는 노드뿐만 아니라 그 자식노드들을 파악하여 LABEL 테이블의 컬럼 L2(0으로 할당), Node(NULL 값 할당)의 값을 변경하여 컬럼 L1의 소수를 재사용한다. 추가적으로 삭제되는 노드 중에 bridge 노드가 있는 경우는 테이블 BRIDGE에 그 삭제를 반영한다.

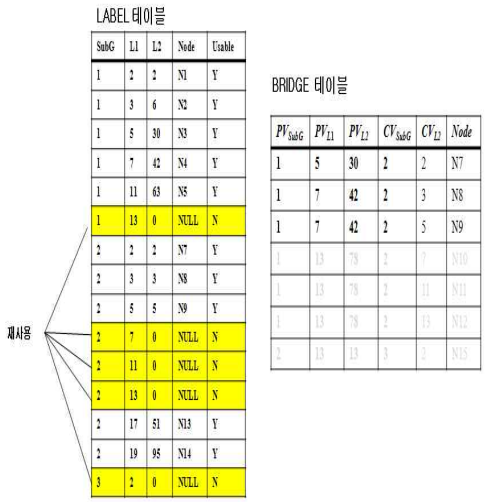
```
UPDATE LABEL
SET L2=0, Node=NULL , Usable = 'N'
WHERE Node IN ( SELECT Node FROM T6);
T6(T1: Select Node, SubG, L2
    From Label
    Where SubG = SUBG(n) and L2 % L2(n) = 0;
Union
T2: Select Node, SubG, L2
    From Label, (select CVSubG, CVL2
                 from bridge, T1
                 where PVSubG = T1.SubG and PVL2 = T1.L2) T3
    Where Label.SubG = T3.CVSubG and Label.L2 % T3.CVL2 = 0;
Union
T4: Select Node, SubG, L2
    From Label, (select CVSubG, CVL2
                 from bridge, T2
                 where PVSubG = T2.SubG and PVL2 = T2.L2) T5
    where label.subG = T5.CVSubG and Label.L2 % T5.CVL2 = 0;)

DELETE BRIDGE
WHERE BRIDGE.Node IN ( SELECT Node FROM T6);
```

그림 10. 삭제 SQL 질의 템플릿
Fig. 10. DELETE SQL template

예를 들어 [그림 5]의 노드 N6뿐만 아니라 이하의 서브트리가 삭제되면 [그림 10]의 템플릿에 의해 노드 N6, N10, N11, N12, N15에 대한 값이 [그림 11]처럼 테이블 LABEL에 반영된다. 특히 N10, N11, N12, N15는 bridge 노드이기 때문에 테이블 BRIDGE에서도 삭제된다.

www.kci.go.kr



SubG	L1	L2	Node	Usable
1	2	2	N1	Y
1	3	6	N2	Y
1	5	30	N3	Y
1	7	42	N4	Y
1	11	63	N5	Y
1	13	0	NULL	N
2	2	2	N7	Y
2	3	3	N8	Y
2	5	5	N9	Y
2	7	0	NULL	N
2	11	0	NULL	N
2	13	0	NULL	N
2	17	51	N13	Y
2	19	95	N14	Y
3	2	0	NULL	N

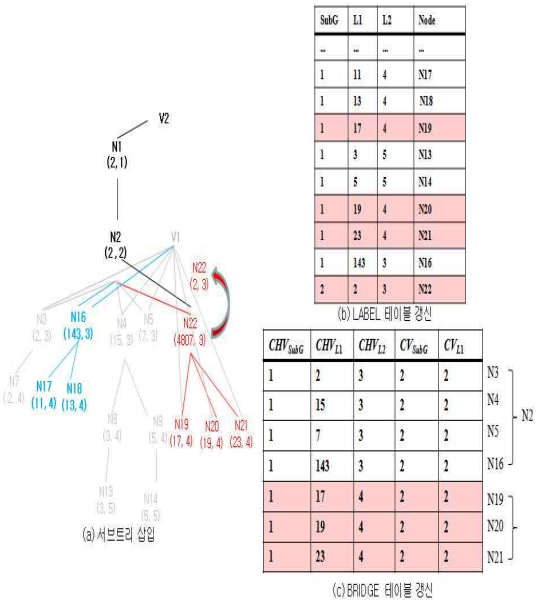
PV_{SubG}	PV_{L1}	PV_{L2}	CV_{SubG}	CV_{L2}	Node
1	5	30	2	2	N7
1	7	42	2	3	N8
1	7	42	2	5	N9
1	13	78	2	7	N10
1	13	78	2	11	N11
1	13	78	2	13	N12
2	13	13	3	2	N15

그림 11. 노드 N6 이하의 서브트리 삭제 후 LABEL과 BRIDGE 테이블
Fig. 11. LABEL and BRIDGE tables after deleting sub tree with a root node N6

삽입 시 삽입되는 노드들의 레이블 할당의 용이성 및 삽입에 의해 기존의 다른 노드들의 레이블 재작성 영역의 최소화의 특성을 갖추어야 한다. 논문에서 제안하는 방법은 삽입되는 위치의 노드(삽입되는 서브트리의 부모노드)가 bridge 노드인지 그렇지 않은 일반 노드인지 확인하는 작업부터 시작된다. Bridge 노드인 경우에도 삽입되는 서브트리에 오버플로우가 발생되는 경우가 있고, 그렇지 않은 경우가 있다.

Bridge 노드에 삽입되는 서브트리에 오버플로우가 발생되지 않은 경우 그 노드의 L2 값을 적용하여 삽입되는 서브트리의 단말노드까지의 경로 단계를 파악하고 레이블을 할당한다. 단말노드의 L1 값의 할당은 배타적인 소수를 할당하는데, 일단 테이블 LABEL에서 재사용이 가능한 것을 이용한다. 이 경우는 기존의 나머지 영역에 대한 레이블 재작성이 필요 없다.

Bridge 노드에 삽입되는 서브트리에 오버플로우가 발생되는 경우는 발생되지 않는 경우까지는 동일한 방법을 적용한다. [그림 12(a)]는 bridge 노드 N2에 노드 N19, N20, N21, N22로 이루어진 서브트리가 삽입되는 경우를 보여주고 있다. 이 경우 노드 N2의 L2 값 2를 이용하여 노드 N19, N20, N21의 L2 값을 구하고, 각 노드의 L1 값은 [그림 12(b)]처럼 테이블 LABEL에서 재사용 가능한 소수를 작은 값부터 할당한다(N19인 경우). 더 이상 재사용할 소수가 없으면, 동일한 SubG 내에 다음으로 사용할 수 있는 배타적 소수를 할당한다. 노드 N20의 L1 값은 19, 노드 N21의 L1 값은 23이 할당된다. 이를 통해 노드 N22의 L1 값을 구하는데 오버플로우가 발생된다. 이 경우는 부모노드의 L1 값을 그대로 할당한다. 따라서 노드 N22의 L1 값은 2가 할당된다. 추가적으로 bridge 노드 N22가 생겼기 때문에 자식 정보를 [그림 12(c)]처럼 BRIDGE 테이블에 추가한다. 이 경우도 기존의 나머지 영역에 대한 레이블 재작성이 필요 없다.



SubG	L1	L2	Node
...	...	...	...
1	11	4	N17
1	13	4	N18
1	17	4	N19
1	3	5	N13
1	5	5	N14
1	19	4	N20
1	23	4	N21
1	143	3	N16
2	2	3	N22

CHV_{SubG}	CHV_{L1}	CHV_{L2}	CV_{SubG}	CV_{L1}	
1	2	3	2	2	N3
1	15	3	2	2	N4
1	7	3	2	2	N5
1	143	3	2	2	N16
1	17	4	2	2	N19
1	19	4	2	2	N20
1	23	4	2	2	N21

그림 12. bridge 노드에 삽입되는 서브트리에 오버플로우가 발생되는 경우
Fig. 12. Overflow resolution after inserting a sub tree on a bridge node

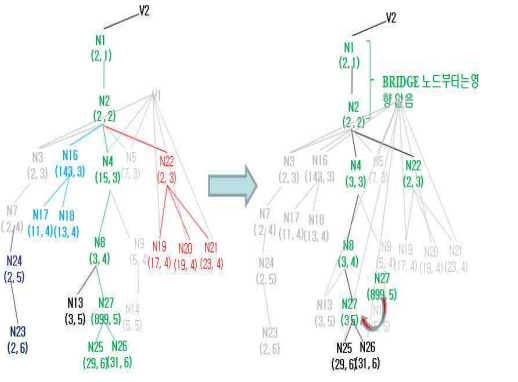
일반노드에 삽입되는 경우도 두 가지 경우로 나뉜다. 단말노드에 자식이 하나만 있는 서브트리를 삽입하는 경우와 형제가 있는 노드에 서브트리를 삽입하는 경우다.

단말노드에 자식을 하나만 갖고 있는 서브트리를 삽입하는 경우는 단말노드의 L2 값을 이용하여 삽입되는 서브트리의 단말노드까지의 경로 단계를 파악하고 L2 값을 할당하고, L1의 값은 그대로 적용한다. 이 경우도 레이블 재작성이 필요 없다.

일반노드면서 형제가 있는 노드에 서브트리를 삽입하는 경우에만 유일하게 기존 노드의 레이블 재작성이 요구된다. 그렇지만 그 영역은 몇 개의 조상노드뿐이다. [그림 13(a)]은 일반노드 N8에 노드 N13의 형제노드로 N27, N25, N26으로 이루어진 서브트리를 삽입하는 경우를 보여주고 있다. 삽입되는 서브트리의 단말노드 N25와 N26에 배타적인 소수를

www.kci.go.kr

할당하여 (29, 6), (31, 6)의 레이블이 할당된다. 노드 N27의 L1의 값은 오버플로우가 발생된다. 따라서 상위의 가상 서브트리인 V2에서 배타적인 소수를 할당받는데 현 상황에서는 3이 할당된다. 따라서 노드 N27은 bridge 노드가 되어 테이블 BRIDGE에 그 정보가 추가된다. 이와 같은 가상 서브트리의 변화를 조상노드에 반영해야 하기 때문에 노드 N8의 (SubG, L1, L2)는 (2, 3, 4)가 된다. 추가로 노드 N8이 다른 자식노드가 있는 경우 bridge 노드가 되기 때문에 그 정보를 테이블 BRIDGE에 추가한다. 여기서 L1의 값은 자식의 L1 값을 상속하게 된다. 노드 N4 역시 (SubG, L1, L2)는 (2, 3, 3)이 되고 다른 자식노드를 갖고 있기 때문에 bridge 노드가 되어 그 정보를 테이블 BRIDGE에 추가한다. 이와 같은 조상노드의 변경은 기존에 bridge 노드였던 조상노드까지 수행한다. [그림 13(b)]는 이에 대한 테이블 LABEL, [그림 13(c)]는 테이블 BRIDGE의 변화 내용을 보여주고 있다.



SubG	L1	L2	Node
...	...	...	...
2	3	3	N4
...	...	...	...
2	3	4	N8
...	...	...	...
1	29	6	N25
1	31	6	N26
2	3	5	N27

(b) LABEL 테이블 갱신

CHV_{SubG}	CHV_{L1}	CHV_{L2}	CV_{SubG}	CV_{L1}	
...	...	...	...	...	
1	5	4	2	3	N9 } N4
1	3	5	2	3	N13 } N8
1	29	6	2	3	N25 } N27
1	31	6	2	3	N26 } N27

(c) BRIDGE 테이블 갱신

그림 13. 일반노드에 서브트리를 삽입하는 경우
Fig. 13. Inserting a sub tree on a internal node

IV. 실험 및 평가

1. 실험 환경

XMark를 이용하여 [표 1]에 명시된 Scaling Factor에 해당하는 다양한 크기의 XML 문서를 생성하여 사용하였다[11]. 생성된 각 문서는 [표 1]과 같은 노드 수를 갖는다. 실험 대상 레이블링 방법은 범위 기반 레이블링 방법인 XISS, LSDX, 그리고 논문에서 제안한 방법이다. 각 XML 문서는 SAX 파서를 통해 레이블을 작성했으며, 인텔 core2 6300@1.86Ghz 프로세서와 1GB DDR2 메모리가 장착된 마이크로 소프트 윈도우 XP 운영체제에서 자바 가상머신 1.4.2를 이용하였다.

표 1. XML 크기, 노드 수와 XMark의 Scaling Factor
Table 1. XML size, number of nodes and scaling factor of XMark

XML 문서(MB)	노드 수 (천개)	Scaling Factor
1.0	14.9	0.009
2.0	28.4	0.017
3.0	44.1	0.026
4.1	58.9	0.035
5.0	73.7	0.044
6.0	87.0	0.052
7.0	102.7	0.061
8.0	116.0	0.069
9.0	131.3	0.077
10.0	145.8	0.087

2. 레이블 저장 공간

[그림 14]는 XISS, LSDX, 그리고 제안 방법을 이용하여 레이블을 작성한 후 전체 레이블에 대한 저장 공간을 측정한 결과이다. 제안 기법은 XML 트리 데이터의 하나의 노드를 표현하는 레이블의 크기를 고정하여 표현할 수 있는 장점이 있다. 그러나 다른 레이블링 방법들은 노드의 수가 증가할수록 레이블 지정 크기도 상대적으로 증가하는 문제를 갖고 있다. LSDX가 항상 가장 큰 저장 공간을 사용하며 제안 기법은 고정 크기를 사용하기 때문에 노드가 많은 XML 트리 데이터인 경우 XISS보다 효율적인 레이블 저장 공간을 사용한다.

www.kci.go.kr

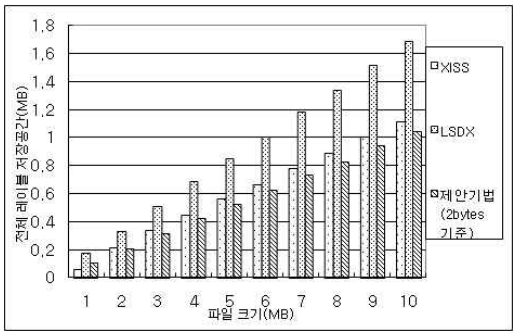


그림 14. 레이블 저장 공간
Fig. 14. Label storage space

3. 동적 XML 환경에 대한 실험

동적 XML 데이터 환경에서의 레이블링 기법은 삽입되는 새로운 노드들에 대한 짧은 레이블링 시간과 기존 노드들에 대한 레이블 재작성 횟수를 최소화해야 한다.

3.1 레이블 재작성 노드 수

직관적으로 XISS는 (order, size)로 구성되어 있으며 새로운 노드가 삽입되면 삽입되는 노드의 조상노드들과 order가 큰 모든 노드들에 대해 레이블 재작성을 해 주어야 한다. LSDX는 새로운 노드가 삽입될 때 기준이 되는 노드의 레이블을 가져와서 삽입되는 노드의 레이블을 계산하여 할당해주기 때문에 기존 노드들에 대한 레이블 재작성이 필요하지 않다. 제안기법은 일반노드면서 형제가 있는 노드에 서브트리를 삽입하는 경우에만 유일하게 기존 노드의 레이블 재작성이 요구된다. 그렇지만 그 영역은 몇 개의 조상노드뿐이다.

3.2 삽입 연산 시간

[그림 15]는 XISS, LSDX 그리고 제안 기법를 이용하여 다양한 크기의 XML 문서에 대하여 1000개의 노드들을 삽입했을 때 기존 노드들에 대한 레이블 재작성과 삽입되는 노드들에 대한 레이블링 시간을 더한 결과이다. XISS가 가장 많은 수의 노드들에 대해 레이블 재작성이 필요하기 때문에 노드 대한 레이블링 종료까지 걸리는 시간이 가장 길다. LSDX가 다소 제안 기법보다 좋은 결과를 보이는 것은 제안 기법이 BRIDGE 테이블을 작성해야 하는 시간이 더 필요하기 때문이다.

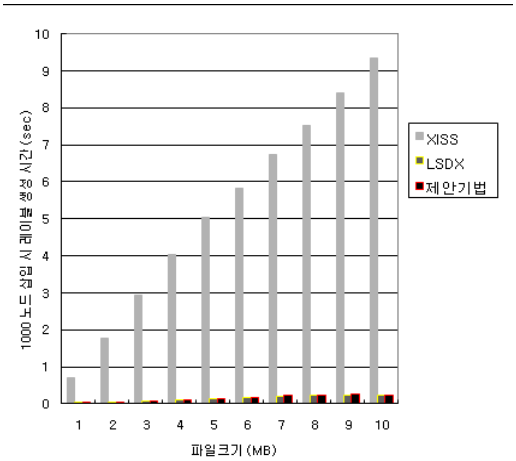


그림 15. 1000개 노드 삽입시 레이블 재작성 시간
Fig. 15. Relabeling time when 1000 nodes are inserted

V. 결 론

본 논문에서는 동적인 XML 환경에서 트리 분해 기반의 소수 레이블링 기법을 제안하였다. 제안한 트리 분해 기반의 소수 레이블링 기법은 노드를 표현하는 레이블의 크기를 고정시킬 수 있어 XML 트리 데이터의 크기에 상관없이 적용 가능하다는 특징을 갖고 있다. 또한 소수의 특징을 그대로 이용하기 때문에 조상, 자손, 형제 관계를 파악하는데 다른 방법보다도 그 처리 방법이 간단하고 빠르다. 또한 삽입되는 노드들의 레이블링 용이성 및 기존 노드의 레이블 재작성을 초래하지도 않는다.

본 논문은 동적 XML 환경에서의 레이블링 기법들 중에서 깊이와 너비 증가에 따른 레이블 확장성 문제 때문에 비교대상에서 좋지 않은 결과를 보여주었던 소수 레이블링 기법을 새롭게 고찰할 수 있었다.

향후에는 레이블 할당 시 bridge 노드의 증가 문제를 해결하기 위해 상향식 방법을 선택하였는데, 하향식 방법과 함께 실험을 통해 분석할 필요가 있다. 추가적으로 트리 환경을 벗어나 RDF와 같은 그래프 환경에서의 소수 레이블링 기법에 대한 연구로 확장하고자 한다.

참고문헌

[1] T. Bray, J. Paoli, C. Sperberg-McQueen, E. Maler, and F. Yergeau, "Extensible Markup Language (XML) 1.0," W3C Recommendation, vol. 6, 2000.

www.kci.go.kr

[2] Sangyoon Oh, “X2RD: Storing and Querying XML Data Using XPath To Relational Database,” Journal of the Korea Society of Computer and Information v.14, no.3, pp.57-64, 2009.
[3] Q. Li and B. Moon, "Indexing and Querying XML Data for Regular Path Expressions," The VLDB Journal, pp. 361-370, 2001.
[4] E. Cohen, H. Kaplan, and T. Milo, “Labeling Dynamic XML Trees”, In Proc. of PODS, 2002, 271-281.
[5] M. Duong and Y. Zhang, "LSDX: a new labeling scheme for dynamically updating XML data," Proc. of the 16th Australasian database conference, vol. 39, pp.185-193, 2005.
[6] A. Khaing and N. Thein, "A Persistent Labeling Scheme for Dynamic Ordered XML Trees," in Proc. of the International Conference on Web Intelligence, pp. 498-501, 2006.
[7] X. Wu, M. Lee, and W. Hsu, "A prime number labeling scheme for dynamic ordered XML trees," in Proc. of the 20th International Conference on Data Engineering (ICDE), pp.66-78, 2004.
[8] G. Wu, K. Zhang, C. Liu, J. Li, "Adapting Prime Number Labeling Scheme for Directed Acyclic Graphs," DASFAA 2006, pp. 787-796, April 2006.
[9] KangWoo Lee and JoonDong Lee, “A Prime Numbering Scheme with Sibling-Order Value for Efficient Labeling in Dynamic XML Documents,” Journal of the Korea Society of Computer and Information v.12, no.5, pp.65-72, 2007.
[10] R. Thonangi, "A Concise Labeling Scheme for XML Data," in Proc. of ACM SIGMOD, COMAD, 20, 2006.
[11] A. Schmidt, F. Waas, M. Kersten, M. J. Carey, I. Manolescu, and R. Busse, "XMark: a benchmark for XML data management," in Proceedings of the 28th international conference on Very Large Data Bases Hong Kong, China, 2002.

저 자 소 개

변 창 우
2001: 서강대학교 공학석사.
2007: 서강대학교 공학박사.
2007 - 현재: 인하공업전문대학 컴퓨터시스템과 교수
관심분야: XML 저장기법, XML 접근제어, 접근제어 모델, 트랜잭션 관리, 유비쿼터스 보안, 모바일 보안, 모바일 데이터처리
Email: cwbyun@inhatc.ac.kr

www.kci.go.kr