동적으로 갱신가능한 XML 데이타에서 레이블 재작성하지 않는 원형 레이블링 방법

(A Circle Labeling Scheme without Re-labeling for Dynamically Updatable XML Data)

김 진 영 † 박 석 ††
(Jinyoung Kim) (Seog Park)

요 약 XML은 인터넷과 유비쿼터스 환경의 데이타에 대한 저장과 교환, 출판의 목적으로 널리 사용되고 있다. XML의 광범위한 사용에 따라 XML 데이타를 효율적으로 저장하고 활용하기 위한 방법으로 레이블링 방법이 연구되고 있다. 레이블링 방법에 대한 최근 연구들은 동적으로 업데이트 가능한 XML 문서에 대한 효과적인 레이블링 방법에 중점을 두고 있다. 그러나 레이블 재작성 비용, 레이블 저장을 위한 큰 저장공간 할당 등의 문제점이 있다. 이러한 문제점은 새로운 데이타가 지속적으로 삽입될 경우 더욱 심화된다. 본 논문에서는 XML 문서를 원으로 나타냄으로써 회전수, 부모원/자식원의 개념을 적용하여 전체 레이블 저장공간의 효율을 얻는 방법을 제시한다. 그리고 반지름 개념을 적용하여 동일 위치에 지속적인 새로운 데이타 삽입 시에도 레이블의 길이가 증가하지 않으면서 기존 레이블의 변경을 초래하지 않는 방법을 제시한다. 또한 실험을 통해 제안하는 원형 레이블링 방법의 우수성을 보인다. 본 논문은 XML 문서를 원으로 이해하는 새로운 시도를 한 점과 XML 문서의 크기 증가 시 레이블 저장공간의 효율을 얻을 수 있는 점과 동적 XML 환경에서 새로운 데이타의 업데이트 시에 기존 노드들에 대해 레이블 재작성을 피할 수 있는 점에 의미가 있다.

키워드 : 원형 레이블링 방법, XML, 동적 XML 환경, 레이블 재작성

Abstract XML has become the new standard for storing, exchanging, and publishing of data over both the internet and the ubiquitous data stream environment. As demand for efficiency in handling XML document grows, labeling scheme has become an important topic in data storage. Recently proposed labeling schemes reflect the dynamic XML environment, which itself provides motivation for the discovery of an efficient labeling scheme. However, previous proposed labeling schemes have several problems: 1) An insertion of a new node into the XML document triggers re-labeling of pre-existing nodes. 2) They need larger memory space to store total label. etc. In this paper, we introduce a new labeling scheme called a Circle Labeling Scheme. In CLS, XML documents are represented in a circular form, and efficient storage of labels is supported by the use of concepts Rotation Number and Parent Circle/Child Circle. The concept of Radius is applied to support inclusion of new nodes at arbitrary positions in the tree. This eliminates the need for re-labeling existing nodes and the need to increase label length, and mitigates conflict with existing labels. A detailed experimental study demonstrates efficiency of CLS.

Key words : Circle Labeling Scheme, XML, Dynamic XML Environment, re-labeling

· 본 연구는 한국과학재단 특정기초연구(R01-2006-000-10609-0) 지원으로 수행되었음

† 학생회원 : 서강대학교 컴퓨터공학과
bubble@sogang.ac.kr
†† 정 회 원 : 서강대학교 컴퓨터공학과 교수
spark@sogang.ac.kr

논문접수 : 2008년 9월 24일
심사완료 : 2008년 12월 20일

정보과학회논문지: 데이타베이스 제36권 제2호(2009.4)

1. 서 론

XML(eXtensible Markup Language)[1] 데이타 모델은 자기 기술적(self-describing)이며 반구조적 데이타(semi-structured data)를 저장하기 용이한 언어로써 인정받으면서 데이타 저장을 위한 언어로 광범위하게 사용되고 있다. 이에 따라 XML로 작성된 데이타를 효율적으로 저장하고 질의처리하기 위한 다양한 연구들이 제시되고 있다.

대표적인 질의처리 언어(query language)인 XPath[2]와 XQuery[3]는 XML 문서를 트리 구조로 나타내고 트리를 순회하여 구조적 조인[4]을 통해 질의에 대한 결과를 도출한다. 초기 질의 처리 시스템은 XML 트리를 하향식(Top-Down) 또는 상향식(Bottol-Up)[5]으로 순회함으로써 사용자 질의 처리를 수행한다. 이 전통적인 방법은 모든 질의에 대해 전체 XML 트리를 순회하게 되기 때문에 질의 처리 시간이 오래 걸린다. 이러한 문제점을 해결하기 위해 다양한 레이블링 방법들이 제시되고 있다. 레이블은 XML 문서 내의 각 엘리먼트에 할당된다. 레이블 간 비교를 통해 엘리먼트들의 조상-자손, 형제 관계를 파악할 수 있으므로 실제 XML 문서에 접근하지 않고도 구조적 질의들에 대한 질의처리를 수행할 수 있다.

XML 데이타 모델에 대한 초기 레이블링 방법들은 XML 문서가 업데이트(삽입, 삭제, 갱신 등)되는 환경을 고려하지 않았다. 그러나 최근 XML 문서가 동적으로 업데이트되는 환경이 증가하면서 데이타의 업데이트를 효율적으로 반영할 수 있는 레이블링 방법들이 제시되고 있다. 레이블링 방법은 XML 문서의 업데이트 고려 여부에 따라 다음과 같이 두 부류로 나눌 수 있다.

- **정적 레이블링 방법(Static Labeling Scheme)**

XML 문서의 업데이트가 고려되지 않는 환경에 적합하다. 엘리먼트에 할당된 레이블들 간의 비교를 통해 엘리먼트들의 순서와 조상-자손, 형제 관계를 알 수 있다. 그리고 전체 레이블에 대해 작은 저장공간을 할당할 수 있는 장점이 있다. 그러나, 새로운 데이타가 삽입되면 다수의 기존 노드들의 레이블을 재작성해주어야 하는 문제점이 발생하기 때문에 XML 문서의 업데이트가 빈번한 환경에 적합하지 않다. 예: 전위-후위 레이블링 방법[6], 범위 기반 레이블링 방법[7]등

- **동적 레이블링 방법(Dynamic Labeling Scheme)**

XML 문서에 대한 업데이트 연산이 수행될 때 레이블 재작성이 필요한 노드들의 수가 정적 레이블링 방법보다 적기 때문에 XML 문서의 업데이트가 고려되는 환경에 적합하다. 그러나 새로 삽입되는 노드에 대한 레이블 길이가 기존 레이블 길이보다 길어지는 문제점이 있다. 그리고 동일 위치에 새로운 데이타가 지속적으로 삽입될 때 데이타의 레이블 길이가 지속적으로 길어져서 전체 레이블 저장공간이 커진다[8-11]. 또한 레이블 생성 알고리즘이 복잡하기 때문에 기존 노드들의 레이블 재작성 시 계산시간이 오래 걸린다[12]. 예: 접두사 기반 레이블링 방법[8-10], 소수 기반 레이블링 방법[11], 섹터 기반 레이블링 방법[12] 등

환자 정보를 포함하는 XML로 작성된 병원 데이타는 환자 정보의 빈번한 업데이트가 발생한다. 이처럼 XML 데이타는 업데이트될 수 있고 레이블링 방법은 변동사항을 효율적으로 반영할 수 있어야 한다. 견고하고 레이블링 방법은 다음 요건들을 충족시켜야 한다.

- 레이블을 사용하여 XML 문서 내 엘리먼트의 순서를 표현할 수 있어야 한다.
- 엘리먼트에 할당된 레이블들을 비교하여 엘리먼트들의 조상-자손, 형제 노드 관계를 알 수 있어야 한다.
- 레이블 저장을 위해 작은 저장공간을 할당할 수 있어야 한다.
- XML 문서의 업데이트 발생시 변동사항을 효율적으로 반영해야 한다.
  - 기존 레이블에 대한 재작성 없이 해당 데이타의 레이블만 반영 가능해야 한다.
  - 새로운 데이타의 삽입 시 레이블 길이가 길어지지 않아야 한다.

본 논문에서 제시하는 '원형 레이블링 방법'은 '범위 기반 레이블링 방법'에 기반한다. '범위 기반 레이블링 방법'에서 사용하는 레이블은 (*start, end*)이며 간편한 계산(2.1.2 참고)으로 엘리먼트들의 순서와 조상-자손, 형제 노드 관계를 알 수 있다. '원형 레이블링 방법'도 동일한 계산을 사용하여 엘리먼트들의 순서와 관계를 파악할 수 있다.

본 논문에서는 XML 문서를 원으로 이해하는 새로운 시도를 함으로써 원의 속성을 적용해 다음과 같은 효율을 얻었다.

- **레이블 저장공간 효율**
  - '회전수' 개념을 적용하여 작은 크기의 변수를 반복 사용할 수 있기 때문에 레이블을 저장하기 위한 메모리 공간을 절약할 수 있다.
  - '부모원/자식원' 개념을 적용하여 레이블 값 누적을 방지하여 레이블 구조에 작은 크기의 변수를 할당할 수 있기 때문에 저장공간에 대해 효율적이다.
- **XML 문서의 업데이트가 빈번한 환경에 적합**
  - '반지름' 개념을 적용하여 새로운 데이타가 삽입되더라도 기존 데이타들의 레이블에 대한 재작성 없이 해당 데이타의 레이블만 갱신할 수 있다.
  - 또한, '반지름' 개념을 적용함으로써 새로운 데이타

의 빈번한 업데이트 시에도 레이블 길이가 한정되어 레이블을 저장하기 위한 메모리 공간을 절약할 수 있다.

제시하는 '원형 레이블링 방법'은 XML 문서를 원으로 이해하는 새로운 시도를 하였고 레이블을 저장을 위해 작은 메모리 공간을 할당할 수 있다. 그리고 새로운 데이타의 업데이트 시에 기존 데이타들의 레이블을 재작성하지 않아도 되고 레이블 길이가 한정되어 작은 저장공간을 할애할 수 있음에 의미가 있다.

본 논문의 구성은 다음과 같다. 2장에서 관련연구를 소개하고 3장에서 레이블 저장공간을 줄이기 위한 방법을 소개하고 동적 XML 환경에 효율적인 방법을 제시한다. 그리고 4장에서는 실험결과를 설명하고 5장에서는 결론을 기술한다.

2. 관련연구

2.1 정적 레이블링 방법

2.1.1 전위-후위 레이블링 방법

전위 후위 레이블링 방법[6]은 XML 트리를 전위 순서(pre-order)와 후위 순서(post-order) 탐색을 통해 각 노드에 두 값 (Pre, Post)을 부여하여 그것을 레이블로 사용한다. 그림 1(a)는 전위 후위 레이블링 방법의 예이다. 레이블 (2,4)를 갖는 노드는 전위 순서로 두 번째 탐색되는 노드이고 후위 순서로 네 번째로 탐색되는 노드이기 때문에 (2,4)의 레이블이 할당된다. 다음과 같은 레이블 값들 간의 비교를 통해 노드들의 조상-자손, 형제 관계를 판단할 수 있다.

A와 B 노드의 레이블이 $(pre_A, post_A)$, $(pre_B, post_B)$라고 할 때, $pre_A < pre_B \wedge post_A > post_B$를 만족하면 A는 B의 조상 노드이다.

이 레이블링 방법은 레이블 크기가 작기 때문에 전체 레이블 저장공간에 대해 효율적이지만 새로운 노드가 삽입되면 XML 문서 내의 전체 데이타의 레이블을 재작성해야 하는 문제점이 있다.

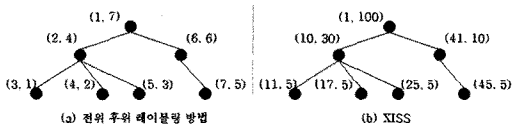


그림 1 정적 레이블링 방법의 예

2.1.2 범위 기반 레이블링 방법

XISS(XML Indexing and Storage System)[7]에서는 레이블로 (order, size)를 사용한다. order는 XML 문서 내에서 엘리먼트의 순서이고 size는 자손으로 가질 수 있는 엘리먼트들에 할당될 수 있는 범위이다. 다음과 같은 레이블 값들 간의 비교를 통해 노드들의 조상-자손, 형제 관계를 판단할 수 있다.

A와 B 노드의 레이블이 $(order_A, size_A)$, $(order_B, size_B)$라고 할 때, $order_A < order_B < order_A + size_A$를 만족하면 A는 B의 조상 노드이다.

그림 1(b)는 XISS에 대한 예이다. (10,30)의 레이블을 갖는 노드는 XML 문서 내에서 10번째 순서를 갖고 자손노드에 할당할 수 있는 30만큼의 범위를 소유하고 있다. 레이블이 (11,5)인 노드는 order가 11이고 "10< 11<10+30"이 성립하므로 레이블이 (10,30)인 노드의 자손노드이다. XISS는 전위 후위 레이블링 방법과 같이 order와 size 두 값으로 레이블이 구성되므로 레이블 크기가 작아 전체 레이블 저장공간에 대해 효율적이다.

전위 후위 레이블링 방법에 비해 XISS는 레이블 구조 중 size 값을 이용해 새로 삽입되는 데이타에 기존 데이타의 레이블을 재작성하지 않고 레이블을 할당할 수 있기 때문에 동적 XML 환경에 대한 고려를 했다고 할 수 있다. 그러나 해당 위치에 새로운 데이타를 삽입하고자 할 때, 미리 할당된 여분의 범위가 모두 소진되면 다수의 기존 노드들에 대한 레이블 재작성이 필요하게 되는 문제점이 발생한다.

2.2 동적 레이블링 방법

2.2.1 소수 기반 레이블링 방법

소수 기반 레이블링 방법[11]은 소수의 특성을 사용한다. XML 트리의 각 노드에 셀프 레이블로 소수(prime number)를 할당하고 각 노드의 실제 레이블은 부모 노드의 실제 레이블과 자식 노드의 셀프레이블의 곱이다. 다음과 같은 레이블 값들 간의 비교를 통해 노드들의 조상-자손, 형제 관계를 판단할 수 있다.

어떤 두 노드 A와 B가 있을 때, "$Label(B)$ mod $Label(A) = 0$"이면 A는 B의 조상 노드이다.

그림 2(a)에서 노드 u와, v의 레이블은 2과 10이다. 이때 "label(v) mod label(u) = 10 mod 2 = 0"이기 때문에 u는 v의 조상 노드이다.

순서화된 XML 트리에 대한 업데이트가 발생하면 중국인의 나머지 정리(CRT: Chinese Remainder Theorem)를 사용하여 전체 레이블에 대한 재작성 없이 새로 삽입되는 데이타만 레이블링 하면 된다. 이 때 XML 문서 내 데이타의 순서를 유지하기 위해 SC(Simultaneous Congruence) 값을 사용한다. SC 값을 x라 하고 어떤 노드의 셀프 레이블을 a라 했을 때 "x mod a"를 통해 얻은 값이 해당 노드의 순서이다. SC 값은 일반적으로 매우 큰 값이 할당되며 새로운 노드가 삽입될 때 SC 값에 대한 재계산 비용이 발생한다. 소수 기반 레이블링 방법은 소수를 사용하기 때문에 정수에 대한

가용성이 떨어지고 두 소수의 곱을 저장할 큰 변수가 필요하기 때문에 저장공간의 비효율적인 사용을 발생시킨다. XML 문서의 크기가 커질수록 더 큰 소수가 필요하고 더 큰 두 소수의 곱이 생성되기 때문에 큰 레이블 저장공간을 필요로 하며 두 소수의 곱에서 해당 소수들을 추출하는데 많은 시간이 소요되는 문제가 있다. 그렇기 때문에 소수 기반 레이블링 방법은 동적 XML 환경을 고려하지만 큰 XML 문서에 적용하기에는 적합하지 않다.

2.2.2 접두사 기반 레이블링 방법

LSDX(Labeling Scheme for Dynamic XML data)[8]는 숫자와 문자의 조합을 레이블에 사용한다. LSDX의 레이블은 XML 트리 내에서 해당 노드의 레벨(level) 정보와 부모 노드의 레이블과 해당 노드의 셀프 레이블의 접합으로 구성된다. 그림 2(b)에서 1a.b는 레벨이 1이면서 0a를 갖는 부모노드의 레이블인 a를 접두사로 사용하고 해당노드의 셀프레이블인 b를 붙여서 생성된다. 이와 같이 한 노드의 레이블 구조 내에 레벨 정보와 조상노드의 레이블을 접두사로 포함하고 있기 때문에 두 레이블의 비교를 통해 조상-자손 관계를 파악할 수 있고, 셀프 레이블을 통해 XML 문서 내에서의 순서와 형제 관계를 파악할 수 있다. 그림 2(b)에서 1a.b와 1a.z는 XML 트리 내에서 level 1에 위치하고 접두사가 a로 동일하기 때문에 같은 부모를 가지며 셀프레이블인 b와 z 중 b가 z보다 앞서기 때문에 1a.b가 1a.z보다 전임자 노드임을 알 수 있다. 또한 동적 XML 환경에 대한 고려를 위해 문자를 사용해 새로운 노드의 삽입 시 기준 노드의 레이블을 가져와 수정한 뒤 할당해 주는 방법을 사용한다. 그림 2(b)에서 새로운 노드 A가 삽입되면 1a.b 레이블을 갖는 노드 앞에 기준이 되는 노드의 셀프 레이블 b보다 빠른 순서를 갖는 문자 a를 붙여 레이블링 한다.

LSDX는 XML 트리의 깊이가 깊어질수록 레이블의 길이가 길어져서 레이블 저장공간에 대해 비효율적이며, 동적 XML 환경에 대한 고려는 있었으나 새로운 노드가 지속적으로 동일한 자리에 삽입될 경우 레이블의 길이가 길어지는 문제점이 있다. 그리고 그림 2(b)에서 B와 C 노드가 동일한 레이블을 갖게 되어 충돌이 발생함을 알 수 있다.

2.2.3 섹터 기반 레이블링 방법

섹터 기반 레이블링 방법[12]은 반지름(2^{Ar})과 호의 길이(A_s)를 이용하여 레이블 (A_r, A_s)를 구성한다. 부모노드 A의 레이블이 (A_r, A_s)이라 할 때, A의 자식노드들의 레이블은 다음과 같은 방법으로 결정된다.

1) A의 자식노드들의 개수보다 큰 2^k를 만족하는 최소의 k 값을 찾는다.
2) A의 자식노드들의 반지름은 $2^{Ar}+k$
3) A의 자식노드들의 호의 길이는 A_s*2^k+i (i : 자식노드들 중 해당 노드의 순서)
4) 자식노드들의 레이블은 (A_r+k, A_s*2^k+i) (i : 자식노드들 중 해당 노드의 순서)

그림 2(c)는 위 방법으로 A의 첫 번째 자식노드인 D를 레이블링 한 예이고 D의 레이블은 (A_r+k, A_s*2^k+1)이다.

섹터 기반 레이블링 방법은 레이블간 비교를 통해 두 노드의 조상-자손, 형제노드 관계를 판단할 수 있다.

A의 레이블을 (A_r, A_s)라 하고 D의 레이블을 (D_r, D_s)라 할 때,

- $D_r > A_r \wedge \lfloor D_s >> (D_r - A_r) \rfloor = A_s$를 만족하면 A는 D의 조상노드이다.
- $(\frac{A_r}{D_r} * D_s > A_s) \vee (\frac{A_r}{D_r} * D_s = A_s \wedge A_r < D_r)$ 을 만족하면 A는 D의 전임자 노드이다.

새로운 노드가 삽입되면 해당 노드의 형제노드들의 k 값을 찾고 형제노드가 2^k개 미만이면 그림 10(b)와 같이 새로 삽입되는 노드와 이후에 나올 노드들과 서브트리에 대한 레이블 재작성이 필요하다. 반면 형제노드가 2^k개로 삽입되는 노드에 할당할 레이블이 없을 경우 새로운 k 값을 찾아 그림 10(c)와 같이 모든 형제노드와 이를 루트로 하는 서브트리들의 레이블들에 대한 재작성이 필요하다.

섹터 기반 레이블링 방법은 작은 크기의 레이블을 사용함으로써 전체 레이블 저장공간에 대해 효율적이지만 레이블링 알고리즘이 복잡하다. 그리고 조상-자손, 형제노드 관계 파악을 위한 계산 또한 복잡하여 전체적인 계산시간이 오래 걸린다. 또한 새로 삽입되는 노드에 대한 레이블링을 위해 기존 노드들의 레이블들을 재작성해주어야 하는 문제점이 있다.

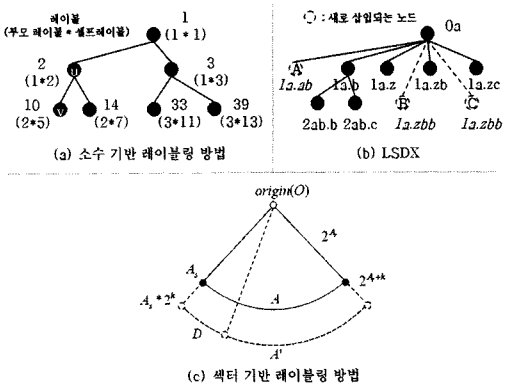


그림 2 동적 레이블링 방법의 예

3. 원형 레이블링 방법

XML 문서를 효율적으로 저장하고 활용하기 위해 레이블링 방법을 사용한다. 본 논문에서 제시하는 레이블링 방법은 XML 문서를 원으로 이해하는 새로운 시도를 함으로써 원의 성질을 레이블링 방법에 적용하여 다음과 같은 장점을 얻을 수 있었다.

1) 범위 기반 레이블링 방법에 기반하기 때문에 동일한 방법을 사용하여 조상-자손, 형제 관계를 쉽게 판단할 수 있다.
2) "회전수"와 "부모원/자식원"의 개념을 적용하여 전체 레이블 저장을 위한 메모리 공간의 효율을 얻을 수 있다.
3) "반지름" 개념을 적용하여 XML 문서의 업데이트 발생 시 기존 데이타들의 레이블을 재작성하지 않아도 되며, 레이블이 일정 길이 이상 증가하지 않는다.

본 논문에서 제시하는 새로운 레이블링 방법을 "원형 레이블링 방법"이라 한다.

3.1 원형 레이블링 방법의 기본 레이블 구조

원형 레이블링 방법은 범위 기반 레이블링 방법에 기반한다. 범위 기반 레이블링 방법의 레이블은 (*order*, *size*)이다. 본 논문에서는 (*order*, *size*) 대신에 (*start*, *end*)를 사용한다. 이 때 *start*는 *order*와 동일하고 *end*는 *order* + *size*이다.

정의 1. (*start*, *end*) : 범위 기반 레이블링 방법의 (*order*, *size*)의 변형 :

1) *start*는 *order*와 동일한 의미이다.
2) *end* = *order* + *size*

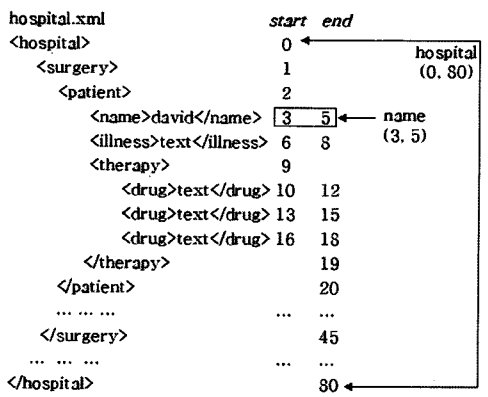
그림 3(a)에서처럼 XML 문서의 첫 엘리먼트의 시작 태그에 0을 부여하고 다음에 나오는 모든 태그들에 1씩 증가한 값을 부여하여 각 엘리먼트의 시작, 끝 태그에 부여된 값을 쌍으로 묶어 (*start*, *end*)를 생성한다. hospital 엘리먼트는 시작 태그에 0, 끝 태그에 80이 할당됐다. 이 두 값을 묶어 (0, 80)이 레이블로 할당된다. 그림 3(b)는 그림 3(a)에서 생성된 레이블을 XML 트리에 매칭한 예이다.

다음 두 조건을 사용해 엘리먼트들의 (*start*, *end*) 레이블들을 비교함으로써 조상-자손, 형제 노드 관계를 파악할 수 있다[7]:

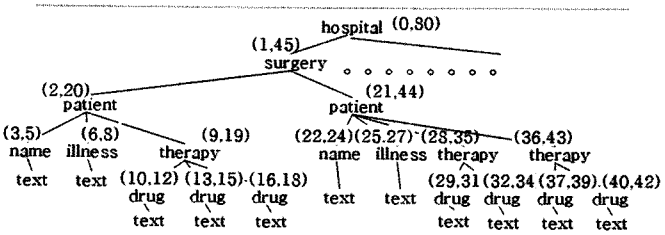
어떤 노드 u,v에 대해 u,v의 레이블이 $(start_u, end_u)$, $(start_v, end_v)$일 때,

1) "$start_u < start_v \wedge end_u > end_v$" 이면 u는 v의 조상노드이다.
2) "$start_u < end_u < start_v < end_v$"이면 u는 v의 전임자노드이다.

그림 3(b)는 [14]에서 인용한 XML 문서의 예이다. 그림 3(b)에서 hospital의 레이블이 자손 노드들의 레이



(a) XML 문서(hospital.xml)와 *start*, *end* 값 할당의 예



(b) XML 문서(hospital.xml)와 (*start*, *end*) 레이블링의 예

그림 3 XML 문서(hospital.xml)와 (start, end)를 사용한 레이블링의 예

블을 모두 포함하기 때문에 언제나 조상노드임을 알 수 있다. 그리고 첫 번째 patient 노드와 두 번째 patient 노드의 레이블을 비교하여 두 patient 노드들이 형제노드임을 알 수 있고 첫 번째 patient 노드가 두 번째 patient 노드의 전임자 노드임을 알 수 있다.

3.2 "회전수", "부모원/자식원" 개념을 적용한 원형 레이블링 방법

3.2.1 문제정의

(*start*, *end*)에 기반한 레이블링 방법은 범위 기반 레이블링 방법과 동일하게 하나의 레이블을 저장하기 위해 두 개의 변수가 필요하기 때문에 적은 저장공간을 필요로 한다. 그러나 접두사 기반 레이블링 방법은 레벨(level) 정보와 조상 노드들의 레이블(접두사), 셀프 레이블 등을 저장하기 위해 더 많은 변수를 필요로 한다. 또한 이 방법은 각 레이블이 조상 노드들의 레이블 정보를 저장하고 있어야 하기 때문에 XML 트리의 깊이에 따라 레이블 길이가 길어진다. 그러므로 (*start*, *end*)를 사용하는 레이블링 방법이 접두사 기반 레이블링 방법에 비해 레이블 저장을 위한 메모리 공간 사용 측면에서 효율적이다.

원형 레이블링 방법은 (*start*, *end*) 레이블을 사용함으로써 기본적으로 다음과 같은 장점을 갖는다.

1) 레이블 저장을 위한 적은 저장 공간 사용
2) 간편한 방법을 통해 조상-자손, 형제 관계 파악 가능
3) XML 문서의 데이타 순서 유지

기존 레이블링 방법들과 본 논문에서 제시하는 원형 레이블링 방법을 사용하는 시스템을 구현할 때 다음과 같은 사항을 가정한다.

가정 1. 레이블링 시스템의 구현 시 기계 워드(machine word) 단위를 사용한다. 중앙 처리 장치와 메모리 사이에 발생하는 입출력(I/O)의 기본 단위로 기계 워드(machine word)를 사용하므로 비트 단위 처리는 처리 속도가 느려지기 때문이다. 기계 워드를 사용함으로써 다음과 같은 장점을 얻을 수 있다.

1) 일반적인 프로그래밍 언어(C, JAVA 등)들은 기계 워드 단위로 연산하기 때문에 구현할 때 편의를 도모할 수 있다.
2) bit 단위 처리보다 기계 워드 단위 처리 속도가 더 빠르다.

XML 문서가 동적으로 갱신되는 환경에서 문서의 크기는 지속적으로 커지고 엘리먼트의 수도 증가한다. 이 때 (*start*, *end*)의 각 변수는 문서 내의 엘리먼트 수를 표현할 수 있는 크기로 할당되어야 한다. 만약 XML 문서 내 엘리먼트의 수가 30,000개라면 변수는 최소 60,000의 값을 표현할 수 있어야 한다. 예를 들어, (*start*, *end*) 레이블 내의 각 변수는 2byte의 부호가 없는 정수형(0~65,535)을 할당하여 레이블을 표현할 수 있다.

만약 기존 XML 문서에 새로운 데이타가 지속적으로 삽입되어 엘리먼트 수가 30,000개에서 35,000개로 증가하면 최소 70,000의 값을 표현할 수 있는 변수가 필요하다. 그러나 2byte로는 70,000을 표현할 수 없기 때문에 4byte(0~4,294,967,296)를 변수로 사용해야 한다. 실제로 표현해야 하는 값은 70,000이고 17bit로 표현할 수 있지만 4byte를 할당해야 하기 때문에 15bit 만큼의 저장공간의 낭비가 발생한다. 즉, 변수 내에 내부 단편화가 발생함을 알 수 있다.

그림 4는 XML 문서 크기 증가에 따라 레이블에 사용되는 기존 변수 크기를 증가시켰을 때, 불필요하게 큰 변수를 사용함으로써 내부 단편화가 발생하여 저장공간의 낭비가 발생함을 나타낸다.

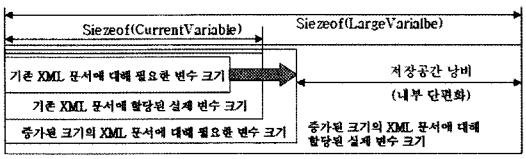


그림 4 XML 문서 크기 증가에 따른 변수 크기 변경과 변수의 내부 단편화 문제

용어정의 1. 각 값(angle value) : XML 문서의 각 엘리먼트의 시작 태그와 끝 태그에 부여된 값. XML 문서의 첫 태그에 0을 부여하고 이후 태그들에 1씩 증가된 값을 부여한다. 그리고 관리자가 지정한 최대 각 값의 범위로 한정한다.

용어정의 2. 최대 각 값(max angle value) : 각 값의 최소 값과 최대 값을 명시한 범위로서 관리자가 지정한 각 값.

정의 2. (*start_angle*, *end_angle*) : XML 문서를 원으로 표현함에 따른 (*start*, *end*)의 변형된 레이블 :

1) *start_angle*은 엘리먼트의 시작 태그에 할당할 수 있는 각 값이다.
2) *end_angle*은 엘리먼트의 끝 태그에 할당할 수 있는 각 값이다.
3) *start_angle*과 *end_angle*에 할당될 수 있는 각 값은 관리자가 지정한 최대 각 값의 범위에 한정된다.
4) 이 레이블 구조는 원형 레이블링 방법의 기본 레이블 구조이다.

(*start*, *end*) 레이블을 사용하는 방법은 기본적으로 적은 레이블 저장공간을 할당할 수 있는 장점이 있다. 그러나 XML 문서의 업데이트가 빈번하면 미리 할당된 변수를 더 큰 변수로 변경해주어야 하는 경우가 발생한다. 이 때 큰 변수를 사용하여 전체 문서에 대한 레이블

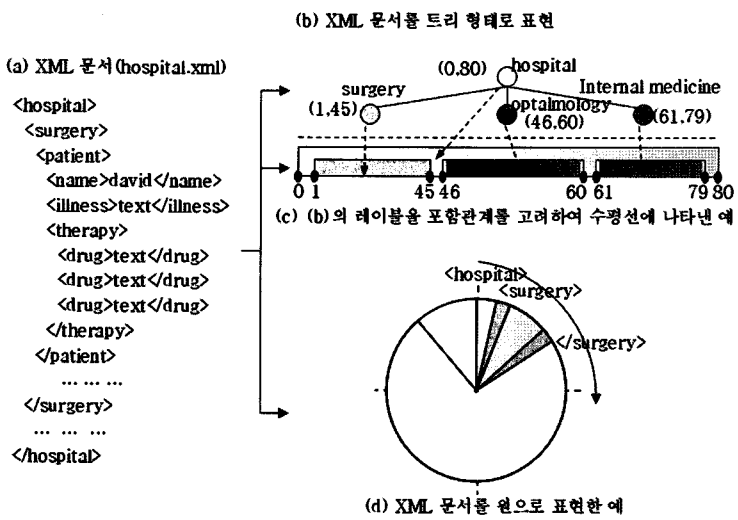


그림 5 XML 문서(hospital.xml)을 트리, 수평선, 원으로 표현한 예

링을 다시 해주어야 하는 비용이 발생한다. 또한 앞서 언급했듯이 큰 변수로 변경했을 경우 변수 내에 내부 단편화가 발생하여 레이블 저장을 위한 메모리 공간을 낭비하게 된다.

3.2.2 XML 문서의 원형 표현

그림 5(b)는 그림 5(a)를 XML 트리로 나타내고 각 노드에 (*start, end*) 레이블을 사용해 레이블링 한 예이다. 그림 5(c)는 그림 5(b)의 각 노드의 레이블 값을 수평선에 표시하고 두 점을 연결하여 영역으로 표현한 것이다. 그림 5(b)를 통해 (*start, end*) 레이블을 수평선에 나타내어 조상-자손, 형제 관계를 쉽게 알 수 있음을 알 수 있다.

본 논문에서는 XML 문서를 트리, 수평선으로 이해하는 것과는 달리 그림 5(d)와 같이 원으로 이해하는 새로운 시도를 했다. 시작 엘리먼트인 hospital의 시작 태그를 0도, 다음으로 나오는 태그들에 대해 1씩 증가하는 각 값(angle value)을 부여한다. 문서에 대한 각 값의 부여가 끝나면 엘리먼트의 시작과 끝 태그에 할당된 각 값을 쌍으로 묶어 (*start_angle, end_angle*) 레이블을 만든다. 이 때 각 값인 *start_angle*과 *end_angle*의 범위는 실제 원의 개념에서는 0~360이지만 원형 레이블링 방법에서는 관리자가 지정할 수 있다. 레이블에 사용되는 *start_angle*과 *end_angle*에 할당된 변수의 크기가 2byte라면 0~65,535만큼의 수를 표현할 수 있기 때문에 관리자는 최대 각 값을 360 대신에 65,535로 지정할 수 있다.

함수 1은 정의 2의 (*start_angle, end_angle*)을 사용해 XML 문서에 레이블링하는 함수를 간략히 기술한 것이다.

Function 3.1 : CLS basic-labeling Function
Input: $T=\{t_1,\ldots,t_m\}$: T is a set of tags in common document. t_i can be start tag or end tag.
Output: A set of labels such that the label l_j is ($start\_angle_j$, end_angle_j).
1 : set L; // L is a array. Each array element includes < *element_id, element_name, start_angle, end_angle, parent_id* >
2 : set D; // D is a array. Each array element includes < *element_id* >
3 : set *count* ← -1; // counter for label
4 : set j ← 0; // counter for *element_id*
5 : set k ← -1; // counter for depth in XML tree
6 : **foreach** t_i **do**
7 : **if** t_i is a start tag **then**
8 : set j ← j + 1;
9 : set L_j.*element_id* ← j;
10: set L_j.*element_name* ← t_i.*element_name*;
11: set *count* ← *count* + 1;
12: set L_j.*start_angle* ← *count*;
13: set k ← k + 1;
14: set D_k ← L_j.*element_id*;
15: **elseif** t_i is an end tag **then**
16: set *count* ← *count* + 1;
17: set L_{D_k}.*end_angle* ← *count*;
18: set k ← k - 1;
19: **endif**
20: **end**

함수 1 (start_angle, end_angle)를 사용한 레이블링 함수

엘리먼트들의 (*start_angle*, *end_angle*) 레이블들 간 비교를 통해 엘리먼트들의 조상-자손, 형제 관계를 쉽게 파악할 수 있다.

어떤 노드 u, v에 대해 u와 v의 레이블이 ($start\_angle_u$, end_angle_u), ($start\_angle_v$, end_angle_v)일 때,

1) "$start\_angle_u < start\_angle_v \wedge end\_angle_u > end\_angle_v$"이면 u는 v의 조상노드이다.
2) "$start\_anble_u < end\_anble_u < start\_angle_v < end\_angle_v$"이면 u는 v의 전임자노드이다.

3.2.3 원형 레이블링 방법에 회전수 개념 적용

3.2.1에서 설명하였듯이 XML 문서의 크기가 증가하게 되면 레이블에 대하여 더 큰 변수를 부여해야 하고 전체 레이블 저장 공간이 불필요하게 커지는 문제점이 발생한다. XML 문서를 원으로 표현했을 때 최대 각 값을 65,535라 했을 때 XML 문서의 첫 시작 엘리먼트부터 0도로 표현하고 원의 둘레를 따라 첫 번째 회전이라 가정한다. 만약 65,535의 수를 모두 사용했을 때 두 번째 회전, 다시 최대 값을 모두 사용했을 때 세 번째 회전이라 한다. 이 때, 사용할 수 있는 최대 각 값의 범위를 초과했을 경우 회전 수를 증가시켜준다.

용어정의 3. 회전수(Rotation Number): 레이블의 각 값이 관리자가 지정한 최대 각 값의 범위를 초과할 때 증가시켜주는 값. (*start_angle*, *end_angle*) 레이블에서 *start_angle*과 *end_angle*에는 관리자가 지정한 최대 각 값을 표현하기에 충분한 변수가 할당된다. 그러나 XML 문서의 엘리먼트 수가 최대 각 값보다 클 경우 범위가 초과되었음을 알 수 있는 정보가 필요하다. 이 정보를 표현하기 위해 처음 레이블링 시 회전수를 0으로 하고 최대 각 값이 초과될 때마다 회전수는 1씩 증가한 값을 부여한다.

정의 3. ($start\_RN.start\_angle_{RN}$, $end\_RN.end\_angle_{RN}$) : (*start_angle*, *end_angle*)에 회전수 개념을 적용한 레이블. 회전수 개념을 적용한 레이블은 관리자가 지정한 최대 각 값인 *Max_angle*과 *start_ angle*, *end_angle*을 사용해 계산한다:

1) *start_RN*은 *start_angle*의 회전수이고 *end_RN*은 *end_angle*의 회전수 이다.
2) *start_RN* = *start_angle* / *Max_angle* : *start_angle*을 *Max_angle*로 나눈 몫
3) $start\_angle_{RN}$ = *start_angle* % *Max_angle* : *start_angle*을 *Max_angle*로 나눈 나머지
4) *end_RN* = *end_angle* / *Max_angle* : *end_angle*을 *Max_angle*로 나눈 몫
5) end_angle_{RN} = *end_angle* % *Max_angle* : *end_angle*을 *Max_angle*로 나눈 나머지
6) *sizeof*(*Start_RN*) + *sizeof*($Start\_angle_{RN}$) < *sizeof*(*Start_angle*)
7) *sizeof*(*End_RN*) + *sizeof*(End_angle_{RN}) < *sizeof*(*End_angle*)

함수 2는 정의 3의 ($start\_RN.start\_angle_{RN}$, $end\_RN.end\_angle_{RN}$)을 사용해 XML 문서에 레이블링하는 함수를 간략히 기술한 것이다.

Function 2 : CLS Labeling with Rotation Number concept

Input 1: $T=\{t_1, \ldots, t_m\}$: T is a set of tags in common document. t_i can be start tag or end tag.
Input 2: *Max*: *Max* is a maximum angle value which is determined by administrator.
Output: A set of labels such that the label l_j is (s_RN, $start\_angle_j$, e_RN, end_angle_j).

```
1 : set L; // L is a array. Each array element
includes < element_id, element_name, s_RN,
start, e_RN, end, parent_id >
2 : set D; // D is a array. Each array element
includes < element_id >
3 : set count ← -1; // counter for label
4 : set j ← 0; // counter for element_id
5 : set k ← -1; // counter for depth in XML tree
6 : foreach t_i do
7 :     if t_i is a start tag then
8 :         set j ← j + 1;
9 :         set L_j.element_id ← j;
10:         set L_j.element_name ←
                              t_i.element_name;
11:         set count ← count + 1;
12:         set L_j.s_RN ← count / Max;
13:         set L_j.start_angle ← count % Max;
14:         set k ← k + 1;
15:         set D_k ← L_j.element_id;
16:   elseif t_i is an end tag then
17:         set count ← count + 1;
18:         set L_Dk.e_RN ← count / Max;
19:         set L_Dk.end_angle ← count % Max;
20:         set k ← k - 1;
21:   endif
22: end
```

함수 2 정의 3의 레이블을 사용한 레이블링 함수

다음 두 조건을 사용해 엘리먼트들의 ($start\_RN.start\_angle_{RN}$, $end\_RN.end\_angle_{RN}$) 레이블들을 비교함으로써 조상-자손, 형제 노드 관계를 파악할 수 있다.

어떤 노드 u, v에 대해 u와 v의 레이블이 ($start\_RN_u.start\_angle_{RNu}$, $end\_RN_u.end\_angle_{RNu}$),($start\_$

$RN_v.start\_angle_{RNv}$,$end\_RN_v.end\_angle_{RNv}$) 이고 관리자가 지정한 최대 각 값이 *Max_angle*이라 할 때,

1) "($start\_RN_u$. * *Max_angle* + $start\_angle_{RNu}$) < ($start\_RN_v$. * *Max_angle* + $start\_angle_{RNv}$) ∧ (end_RN_u. * *Max_angle* + end_angle_{RNu}) > (end_RN_v. * *Max_angle* + end_angle_{RNv})" 이면 u는 v의 조상노드이다.
2) "($start\_RN_u$. * *Max_angle* + $start\_angle_{RNu}$) < (end_RN_u. * *Max_angle* + end_angle_{RNu}) < ($start\_RN_v$. * *Max_angle* + $start\_angle_{RNv}$) < (end_RN_v. * *Max_angle* + end_angle_{RNv})" 이면 u는 v의 전임자노드이다.

예를 들어 그림 3(b)에서 hospital 엘리먼트는 기본 원형 레이블링 방법을 통해 (0, 80)의 레이블을 갖는다. 이 때 시스템의 기계 단어(machine word)가 6bit라 하고 사용할 수 있는 변수의 단위가 3, 6, 12bit라고 하자. 80을 표현하기 위해서는 12bit 크기의 변수가 필요한데 실제로는 7bit가 필요하다. 12bit를 사용하면 낭비이기 때문에 회전수 정보를 위해 3bit, 각 값을 저장하기 위한 변수로 6bit를 할당한다. (0, 80)은 (0.0, 1.16)로 나타낼 수 있다. 기존 방법대로 12bit를 사용하면 한 레이블의 크기는 24bit이지만 회전수 개념을 적용한 레이블의 크기는 18bit이기 때문에 전체 레이블에 대한 저장공간을 25%만큼 줄일 수 있다.

3.2.4 원형 레이블링 방법에 부모원/자식원 개념 적용

XML 문서에 자주 나타나는 데이타 묶음(subtree)들에 대해 별도로 관리함으로써 저장공간의 효율을 얻을 수 있다. 큰 XML 문서에 대한 레이블링 시 레이블 값이 누적되기 때문에 큰 크기의 변수를 사용해야 한다. 그림 3(b)에서 patient 노드들은 빈번하게 나타나기 때문에 별도 관리를 하게 되면 문서 내에서 레이블 값의 누적을 피할 수 있다.

용어정의 4. 빈출(頻出) 노드들 : 엘리먼트의 이름이 같으면서 동일한 부모 노드를 공유하는 노드들. 환자 정보를 부서 별로 관리하는 hospital.xml에서는 각 부서별로 다수의 환자들에 대한 데이타를 저장하고 있다. hospital.xml 문서를 XML 트리로 나타내면 각 환자 정보들의 루트 노드는 patient라는 엘리먼트 이름을 갖고 이들은 surgery와 같은 부서를 표현하는 엘리먼트들을 부모노드로써 공유한다. 이 때 patient들을 빈출 노드라 한다.

그림 6(a)는 큰 XML 문서 전체에 대하여 기본 레이블 방법을 적용한 예이다. 이 때 patient 노드들의 집합(C, F)들이 빈출 노드들의 집합이다. 그림 6(a)에서는 엘리먼트의 수가 십만 개이기 때문에 2byte의 변수로는 표현이 불가능하여 더 큰 수를 표현할 수 있는 변수를 할당해야 하기 때문에 큰 레이블 저장공간이 필요하다. 그림 6(b)에서처럼 빈출 노드들의 집합에 대해 따로 레이블링 한다면 XML 문서 내에서 레이블 값이 누적되지 않기 때문에 보다 적은 크기의 변수를 할당할 수 있고 레이블 저장공간에 대한 효율을 얻을 수 있다.

원형 레이블링 방법에 부모원과 자식원의 개념을 적용한다.

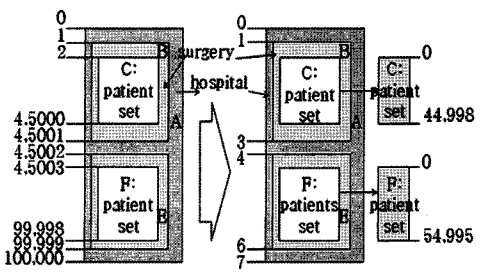
용어정의 5. 부모원(Parent Circle) : 빈출 노드들의 부모노드들과 이 노드들의 조상노드들을 포함하는 서브트리. 환자 정보를 부서 별로 관리하는 hospital. xml에서는 각 부서별로 다수의 환자들에 대한 데이타를 저장하고 있다. 각 부서 엘리먼트 별로 patient라는 이름을 갖는 엘리먼트를 루트로 하는 환자 정보들을 갖는다. 이 때 각 부서 별 환자 정보를 빈출 노드들이라고 한다. 그리고 부서 엘리먼트들과 부서 엘리먼트들의 조상노드들을 부모원이라 한다.

용어정의 6. 자식원(Child Circle) : 동일한 부모 노드를 공유하는 빈출 노드들. 환자 정보를 부서 별로 관리하는 hospital.xml 에서는 각 부서별로 다수의 환자들에 대한 데이타를 저장하고 있다. 각 부서 엘리먼트 별로 patient라는 이름을 갖는 엘리먼트를 루트로 하는 환자 정보들을 갖는다. 각 부서 별 환자 정보를 빈출 노드들이라고 한다. 이때 동일한 부서 엘리먼트를 부모로 하는 자손 엘리먼트들을 자식원으로 분류한다

정의 4. [*PC#*|*CC#*].(*L1*, *L2*) : 정의 4.3.의 레이블에 부모원/자식원 개념을 적용한 레이블.

1) *L1* : $start\_RN.start\_angle_{RN}$, *L2* : $end\_RN.end\_angle_{RN}$
2) *PC#*은 부모원의 식별자이다.
3) *CC#*은 자식원의 식별자이다.

용어정의 7. 부모원/자식원 관계 테이블: 자식원의 부모 노드인 부모원의 노드와의 관계 정보를 갖는 테이블. 테이블은 자식원의 식별자와 해당 자식원의 부모 노드(부모원에 존재)의 레이블 정보를 갖는다.



(a) 누적 값을 사용하는 방법의 예 (b) 자주 사용하는 patient 노드들에 대한 별도 레이블링의 예

그림 6 큰 XML 문서의 레이블 값 누적과 해결 방법의 예

자식원을 구성하기 전에 빈출 노드들과 빈출 노드들의 부모노드를 결정한다. 그림 3(b)에서 patient 엘리먼트들이 자주 나타나기 때문에 patient 엘리먼트들이 빈출 노드들이다. 그리고 각 부서 엘리먼트(surgery, ophthalmology 등)들이 빈출 노드들의 부모노드이기 때문에 부서 엘리먼트를 루트로 하는 서브트리를 자식원으로 결정한다. 그리고 각 부서 엘리먼트들과 부서 엘리먼트들의 조상 노드인 hospital을 부모원으로 결정한다.

그림 7(a)는 XML 문서에 대해 부모원의 범위와 자식원의 범위를 결정한 후 레이블링한 예이다. 그림 7(b)는 그림 7(a)의 레이블을 논리적으로 나타낸 그림이다. 그림 7(c)는 자식원의 식별자와 자식원의 부모 노드의 부모원 내의 레이블 간의 관계 테이블이다.

만약 부모원에서 자식원을 검색해야 한다면 부모원의 단말 노드에 대해 부모원/자식원 관계 테이블을 검색, 확인할 수 있다. 자식원에서 부모원으로의 검색은 자식원의 식별자를 통해 부모원/자식원 테이블을 검색하여 해당 자식원의 루트가 되는 노드의 부모원에서의 레이블을 찾음으로써 가능하다.

어떤 노드 u, v, x, y, z의 레이블이 다음과 같다고 하자.

u의 레이블 : $PC.(L1_u, L2_u)$, v의 레이블 : $PC.(L1_v, L2_v)$, w의 레이블 : $PC.(L1_w, L2_w)$, x의 레이블 : $CC1.(L1_x, L2_x)$, y의 레이블 : $CC1.(L1_y, L2_y)$, z의 레이블 : $CC2.(L1_z, L2_z)$. 이때 $L1$은 $start_RN.start_angle_{RN}$이고 $L2$는 $end_RN.end_angle_{RN}$이다.

이 문서의 부모원/자식원 관계 테이블은 다음과 같다고 하자:

자식원 식별자	부모원에서의 레이블
CC1	$PC.(L1_u, L2_u)$: 노드 u
CC2	$PC.(L1_v, L2_v)$: 노드 v

이때 다음 조건들을 사용해 엘리먼트들의 $[PC\#|CC\#].(start\_RN.start\_angle_{RN}, end\_RN.end\_angle_{RN})$ 레이블들을 비교함으로써 조상-자손, 형제 노드 관계를 파악할 수 있다.

1) 노드 x와 y의 식별자는 *CC1*으로 부모원/자식원 관계 테이블을 확인하여 x와 y가 u의 자손 노드들임을 알 수 있다.
2) x와 y의 식별자는 *CC1*으로 같다. 이때 x와 y의 $(L1_x, L2_x)$과 $(L1_y, L2_y)$를 3.2.3에서의 조상-자손, 형제 관계 판단 방법을 사용해 비교함으로써 동일 자식원 내에서 x와 y의 관계를 판단할 수 있다.
3) u, v와 w는 식별자가 *PC*이기 때문에 부모원에 속하는 노드들이다. 2)와 동일하게 식별자를 제외한 레이블들에 대해 3.2.3에서의 조상-자손, 형제 관계 판단 방법을 사용함으로써 조상-자손, 형제 관계를 판단할 수 있다.
4) y와 z는 식별자가 *CC1*과 *CC2*로 상이하다. 부모원/자식원 관계 테이블을 확인하여 y는 u의 자손이고 z는 v의 자손 노드임을 알 수 있다. 그리고 u와 v의 식별자를 제외한 레이블들에 대해 3.2.3에서의 조상-자손, 형제 관계 판단 방법을 사용하여 순서를 판단할 수 있다. 만약 u가 v의 전임자 노드이면 y는 z의 전임자 노드이다.

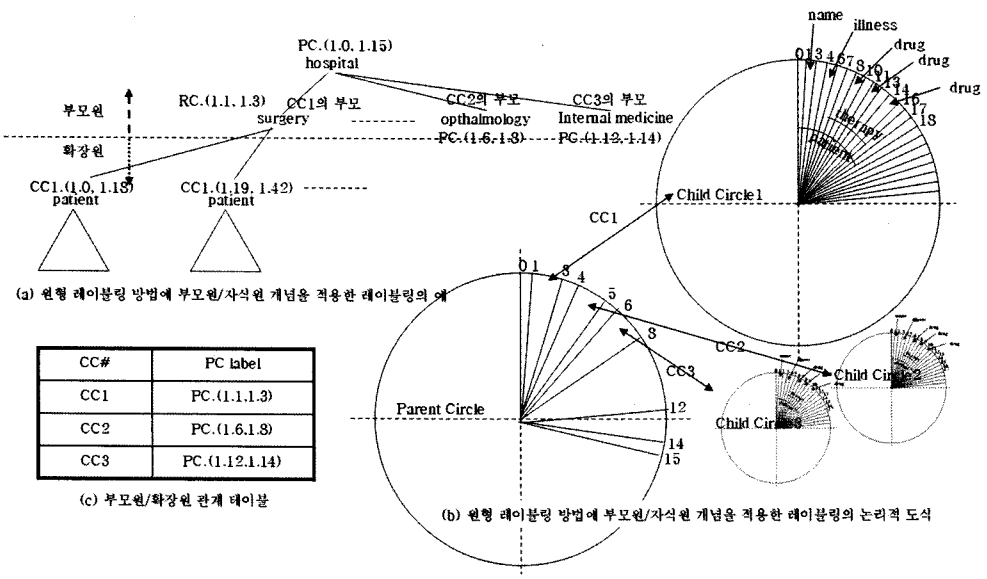


그림 7 원형 레이블링 방법에 부모원/자식원 개념을 적용한 예

XML 문서를 부모원 영역과 자식원 영역들로 분류했을 때, 각 자식원은 다시 부모원과 자식원으로 나뉠 수 있다. 이렇게 다중 레벨로 부모원/자식원 개념을 적용할 수 있다. 그러나 다중 레벨로 나눌 경우 임의의 어떤 노드들 간 조상-자손, 형제 관계 파악을 위해 해야 하는 레이블 간 비교 계산이 복잡해진다. 그러므로 본 논문에서는 단일 레벨만 지원하는 것을 가정한다.

원형 레이블링 방법에서 부모원/자식원의 개념을 적용하여 큰 XML 문서 내에서 레이블 값의 누적을 방지함으로써 적은 크기의 변수를 레이블 구조에서 사용할 수 있으므로 전체 레이블에 대한 적은 저장공간을 사용할 수 있다.

3.3 동적 XML 환경에서의 원형 레이블링 방법

3.3.1 반지름 개념을 적용한 원형 레이블링 방법

XML 문서에 대한 업데이트(update) 연산은 삽입(insert), 갱신(modify), 삭제(delete) 등이 있다. 갱신(modify)은 레이블에 상관 없이 데이타의 변경만 고려하면 되고, 삭제도 레이블에 상관없이 데이타와 레이블을 함께 삭제해 주면 되므로 전체 레이블에 영향을 주지 않는다.

그러나 새로운 데이타의 삽입 시에는 기존 데이타들의 레이블을 재작성해야 하는 문제점이 발생[6,7,12]하기 때문에 레이블 재작성 없이 새로 삽입되는 데이타에 대해서만 레이블링할 수 있어야 한다. 그리고 접두사 기반 레이블링 방법들[8-10]의 경우 같은 위치에 지속적으로 새로운 데이타를 삽입하는 경우 레이블의 길이가 일정하게 증가하기 때문에 레이블 저장공간의 효율을 떨어뜨린다. 또한 섹터 기반 레이블링 방법[12]은 레이블 재작성이 필요하고 레이블 계산과 조상-자손, 형제 관계 파악을 위한 계산 방법이 복잡하기 때문에 레이블링 시간이 오래 걸리는 문제점이 있다.

본 논문에서는 기존 연구들의 문제점들을 해결하기 위해 3.2에서 제시한 레이블링 방법에 반지름 개념을 적용하여 다음과 같은 레이블 구조를 제시한다.

용어정의 8. 반지름: 새로운 데이타가 삽입될 경우 증가/감소하는 값. 새로운 데이타가 기준이 되는 노드의 왼쪽에 삽입되면 반지름이 감소하고 반면에 오른쪽에 삽입되면 반지름이 증가한다.

정의 5. *Radius.L1.(L2, L3)* : 정의 4.4.의 레이블에 부모원/자식원 개념을 적용한 레이블:

1) *L1*은 [*RC#|CC#*], *L2*는 $start\_RN.start\_angle_{RN}$ 이고 *L3*는 $end\_RN.end\_angle_{RN}$ 이다.
2) *Radius*는 반지름으로서 소수를 표현할 수 있는 변수가 할당되며 초기 값은 1이다.

새로운 데이타가 기준이 되는 노드의 왼쪽에 삽입되면 반지름은 감소된 값이 할당되고 오른쪽에 삽입되면 반지름은 증가된 값이 할당된다.

시스템이 시작되면 기존 XML 문서에 대해 정의 5의 레이블을 사용해 레이블링한다. 이 때 반지름은 1로 초기화한다. 다음은 새로운 데이타가 삽입될 경우 레이블링 방법이다:

1) 삽입될 위치를 찾는다.
2) 기준이 되는 노드를 선정한다.
3) 기준이 되는 노드의 반지름과 레이블을 가져온다.
4) 기준이 되는 노드의 오른쪽에 삽입되면 반지름를 증가시키고 왼쪽에 삽입되면 반지름을 감소시킨다.
5) 삽입되는 노드/서브트리에 대한 셀프 레이블을 계산하여 붙여준다.
6) 삽입되는 데이타가 서브트리일 경우 루트를 제외한 자손노드들에 1로 초기화된 반지름과 셀프 레이블을 붙여준다.

알고리즘 1은 새로운 노드가 기준 노드의 왼쪽에 삽입될 때 레이블링 알고리즘이고 알고리즘 2는 기준 노드의 오른쪽에 삽입될 때 레이블링 알고리즘이다.

Algorithm 1 : Insert Left
Input 1 : *E* : *E* is an existing element.
Input 2 : *E'* : *E'* is an inserted element or subtree. depth(*E'*) = depth(*E*) ∧ parent(*E'*) = parent(*E*) ∧ *E'* < *E*. And, *E'* is a nearest element from *E*.
Input 3 : *N* : *N* is a new inserting element or subtree. *N* inserts left side of *E*.
Output : $R_{E'}.(start\_angle_E, end\_angle_E).(start\_angle_{E'}, end\_angle_{E'})$: *E'* label
1 : if *E'* is empty then // insert left side of E
2 : set $R_N \leftarrow \lceil R_E - 1 \rceil$;
3 : // We can label *N* by using [Function 1].
// If N is a subtree, root element of *N* is labeled $R_N.(start\_angle_E, end\_angle_E).(start\_angle_{Nroot}, end\_angle_{Nroot})$.
// Other elements, excepts root element of *N*, labels using [Function 1] only.
4 : **elseif** *E'* is not Empty **then** // insert between *E* and *E'*
5 : set $RN \leftarrow (R_E + R_{E'})/2$;
6 : // We can label *N* by using [Function 1].
// If N is a subtree, root element of *N* is labeled $R_N.(start\_angle_E, end\_angle_E).(start\_angle_{Nroot}, end\_angle_{Nroot})$.
// Other elements, excepts root element of *N*, labels using [Function 1] only.
7 : **end**

알고리즘 1 기준 노드의 왼쪽에 삽입될 때 레이블링 알고리즘

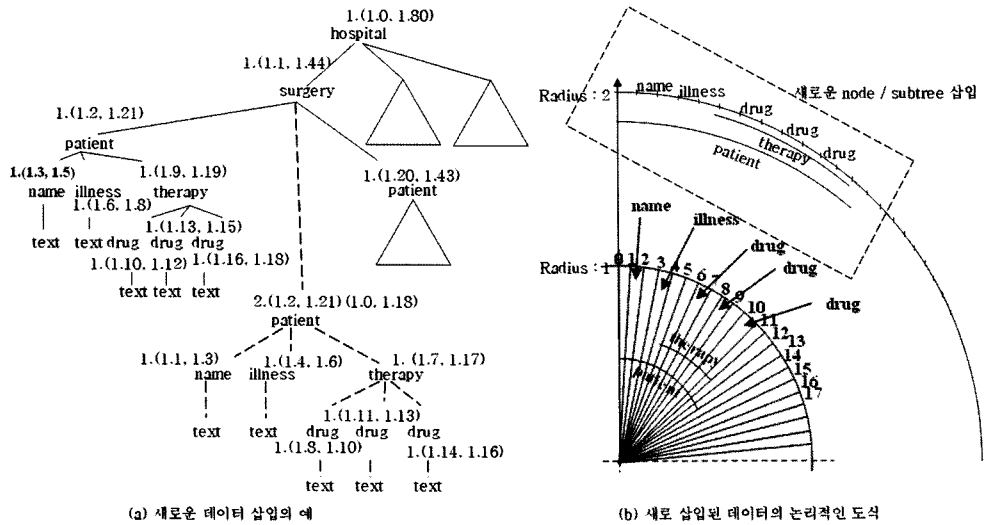
Algorithm 4.2 : Insert Right
Input 1 : E : E is an existing element.
Input 2 : E' : E' is an inserted element or subtree. depth(E') = depth(E) ∧ parent(E') = parent(E) ∧ $E' > E$. And, E' is a nearest element from E.
Input 3 : N : N is a new inserting element or subtree. N inserts right side of E.
Output : $R_{E'}$.($start\_angle_E$, end_angle_E).($start\_angle_{E'}$, $end\_angle_{E'}$) : E' label
1 : if E' is empty then **// insert right side of E**
2 : set $R_N \leftarrow \lfloor R_E + 1 \rfloor$;
3 : // We can label N by using [Function 1].
// If N is a subtree, root element of N is labeled R_N.($start\_angle_E$, end_angle_E).($start\_angle_{Nroot}$, end_angle_{Nroot}).
// Other elements, excepts root element of N, labels using [Function 1] only.
4 : **elseif** E' is not Empty **then // insert between E and E'**
5 : set $RN \leftarrow (R_E + R_{E'})/2$;
6 : // We can label N by using [Function 1].
// If N is a subtree, root element of N is labeled R_N.($start\_angle_E$, end_angle_E).($start\_angle_{Nroot}$, end_angle_{Nroot}).
// Other elements, excepts root element of N, labels using [Function 1] only.
7 : **end**

알고리즘 2 기준 노드의 오른쪽에 삽입될 때 레이블링 알고리즘

그림 8(a)는 첫 번째 patient 노드를 기준으로 오른쪽에 삽입되는 경우에 대한 예이다. 그림 8(b)는 반지름이 1인 patient 노드를 기준으로 오른쪽에 삽입된 경우 반지름을 2로 증가시킨 것에 대한 논리적인 도식이다.

반지름 개념을 적용한 원형 레이블링 방법을 사용할 때 조상-자손, 형제 관계를 파악하기 위해서는 판단해야 할 엘리먼트들의 전체 레이블을 산출해야 한다. 구현 시 B+트리를 사용하기 때문에 한 엘리먼트는 자신의 레이블과 부모 노드의 식별자 등에 대한 정보를 갖는다. 부모 노드의 식별자로 부모 노드의 레이블을 찾을 수 있고 전체 레이블을 구성할 수 있다. 그림 8(a)에서 첫 번째 patient의 name 엘리먼트는 hospital 까지 반지름이 변경되지 않고 1이기 때문에 1.(1.3,1.5)가 전체 레이블이다. 그러나 두 번째 patient의 name 엘리먼트는 hospital 까지 patient 엘리먼트에서 반지름 변경이 있었기 때문에 2.(1.2,1.21). 1.(1.1,1.3)이 전체 레이블이다. 두 번째 patient의 name 엘리먼트의 2.(1.2,1.21)과 첫 번째 patient의 name 엘리먼트의 1.(1.3,1.5)를 먼저 비교한다. (1.2,1.21)은 정리 3에 따라 (1.3,1.5)를 포함한다. 그러나 두 번째 patient의 name 엘리먼트의 반지름이 2이기 때문에 1.(1.3,1.5) 레이블을 갖는 노드의 후임자 노드이다.

그림 9의 a~e 노드는 기존에 있던 노드이고 f와 g는 d 노드를 기준으로 새로 삽입된 노드들이다. f는 d를 기준으로 삽입된 데이타의 루트이기 때문에 d의 레이블과 1 증가된 반지름을 갖는다. f를 루트로 하는 서브트리는 정의 3을 사용해 레이블링 되어 있다. 이때 다음 조건들을 사용해 엘리먼트들의 레이블들을 비교함으로써 조상



(a) 새로운 데이터 삽입의 예 (b) 새로 삽입된 데이터의 논리적인 도식

그림 8 반지름 개념을 적용한 원형 레이블링 방법의 예

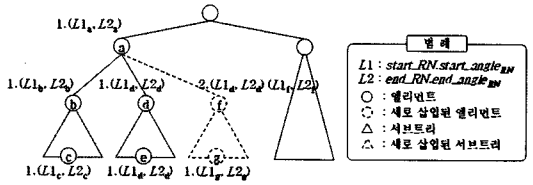


그림 9 XML 트리의 예

-자손, 형제 노드 관계를 파악할 수 있다.

1) a~e 노드들의 전체 레이블은 루트까지 반지름이 변하지 않기 때문에 할당된 레이블을 그대로 사용한다. f 노드의 전체 레이블은 2.($L1_d$, $L2_d$)($L1_f$,$L2_f$)이다. 또한 g 노드의 전체 레이블은 2.($L1_d$, $L2_d$)($L1_g$,$L2_g$)이다.
2) a~e 노드들의 조상-자손, 형제 관계는 3.2.3에서의 조상-자손, 형제 관계 판단 방법을 사용해 판단할 수 있다.
   예: 1.($L1_a$, $L2_a$)와 1.($L1_c$, $L2_c$)에서 $L1_a < L1_c$, $\wedge$ $L2_a > L2_c$ 이고 반지름이 1로 같기 때문에 a는 c의 조상노드이다.
3) a와 f 노드들의 전체 레이블은 1.($L1_a$, $L2_a$)와 2.($L1_d$, $L2_d$)($L1_f$,$L2_f$)이다. 이때 ($L1_d$, $L2_d$)는 ($L1_a$, $L2_a$)에 포함됨을 알 수 있고 f의 반지름이 2이기 때문에 a의 자손 노드 중 1.($L1_d$, $L2_d$)의 레이블을 갖는 노드의 오른쪽에 삽입된 노드임을 알 수 있다. 그렇기 때문에 f는 a의 자손 노드이다.
4) f와 g 노드들의 레이블은 2.($L1_d$, $L2_d$)($L1_f$,$L2_f$)과 2.($L1_d$, $L2_d$)($L1_g$,$L2_g$)이다. 이때 첫 번째 레이블이 ($L1_d$, $L2_d$)로 같고 반지름이 2로 같기 때문에 같은 서브트리에 위치함을 알 수 있다. 그리고 ($L1_f$,$L2_f$)는 ($L1_g$,$L2_g$)를 포함함을 알 수 있기 때문에 g는 f의 자손 노드이다.

소수를 표현할 수 있는 변수 *Radius*를 사용함으로써 XML 문서에 새로운 데이타의 삽입 시 전체 레이블에 대한 갱신 없이 새로 삽입되는 노드/서브트리에 대해서만 레이블링을 할 수 있다. 또한, 같은 위치에 지속적으로 새로운 데이타가 삽입되더라도 고정된 길이의 레이블을 유지할 수 있기 때문에 전체 레이블 저장공간의 효율을 얻을 수 있다.

3.3.2 삽입연산 발생 시 레이블 재작성이 필요한 노드들의 수

스트림 데이타를 사용하는 유비쿼터스 환경과 인터넷 환경에서 XML은 데이타의 저장과 교환, 출판에 있어 광범위하게 표준으로 사용되고, 데이타에 대한 업데이트(삽입, 갱신, 삭제 등) 연산이 빈번하게 발생한다. XML 문서에 대한 업데이트 연산은 빠른 속도로 기존 XML 문서에 반영되어야 하고 사용자에게 원하는 데이타를 제공할 수 있어야 한다. 업데이트 연산의 빠른 수행을 위해 새로운 노드가 삽입될 때 기존 데이타에 대한 레이블을 재작성하지 않아야 한다.

범위 기반 레이블링 방법은 2장 관련연구에서 언급했듯이 (*order*, *size*)의 레이블을 갖기 때문에 새로운 데이타가 삽입될 위치에 여분의 범위(range)가 존재하지 않으면 새로운 데이타의 조상노드들과 *order*가 큰 데이타들에 대한 레이블 재작성이 필요하다. 그림 10(a)는 범위 기반 레이블링 방법을 사용했을 때 새로운 데이타가 삽입될 경우 레이블 재작성이 필요한 구역을 나타낸다.

섹터 기반 레이블링 방법은 새로운 데이타 삽입 시 호의 길이로 사용할 수 있는 값이 남아있는 경우에는 그림 10(b)에서처럼 새로 삽입되는 노드 이후의 노드들에 대한 레이블 재작성을 해야 한다. 또한 호의 길이로 사용할 수 있는 값이 모두 소진됐을 때 그림 10(c)에서처럼 새로 삽입되는 노드의 모든 형제노드들과 이를 루트노드로 하는 서브트리의 모든 노드들에 대해 2장 관련연구에서 언급했던 알고리즘으로 레이블을 재작성해야 한다.

반면 LSDX는 새로운 노드 삽입 시 기준이 되는 노

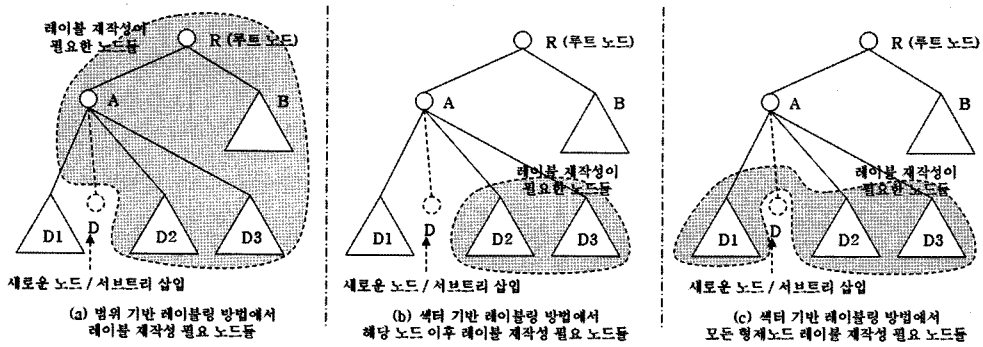


그림 10 각 레이블링 방법 별 레이블 재작성이 필요한 노드들의 범위

드의 레이블을 가져와 새로 삽입되는 노드의 순서를 결정하여 레이블에 반영하기 때문에 기존 노드들의 레이블에 대한 재작성이 불필요하다. 비슷한 방식으로 동적 XML 환경을 위해 반지름 개념을 적용한 원형 레이블링 방법은 기준이 되는 노드의 레이블을 가져와 반지름을 증가/감소 시킨 후 새로 삽입되는 노드와 이를 루트로 하는 서브트리에 대한 레이블을 계산하여 할당하는 방법을 취하기 때문에 반지름 개념을 적용한 원형 레이블링 방법도 기존 노드들의 레이블 재작성이 필요하지 않다.

동적 XML 환경에서 레이블 재작성을 하지 않아도 되기 때문에 업데이트 연산에 대한 레이블링 시간에 있어서 원형 레이블링 방법과 LSDX가 범위 기반 레이블링 방법과 섹터 기반 레이블링 방법 보다 효율적이다. 4장 실험에서 다양한 크기의 XML 문서의 업데이트 발생 시 레이블 재작성이 필요한 노드들의 수와 레이블링 시간에 있어 원형 레이블링 방법이 효율적임을 보인다.

3.3.3 원형 레이블링 방법의 증가하지 않는 레이블 길이

레이블 저장공간에 대한 효율성을 높이는 것은 레이블링 방법의 중요한 요건 중 하나이다. 레이블링 방법이 동적 XML 환경에 적용하기에 적합하더라도 XML 문서에 대한 업데이트 연산이 빈번히 발생했을 때 레이블 저장공간의 효율이 저하되는 문제점이 있다면 동적 XML 환경에 적합한 레이블링 방법이라 할 수 없다.

LSDX와 원형 레이블링 방법은 새로운 노드/서브트리의 삽입 시 기존 노드들의 레이블 재작성 비용이 발생하지 않는 점에서 효율적이다. 그러나 LSDX는 새로운 노드 삽입 시마다 레이블에 하나의 변수를 더 할당해서 삽입된 노드의 XML 문서 상의 순서를 나타낼 수 있도록 한다. 그렇기 때문에 삽입 연산이 발생할 때마다 해당 노드는 기준 노드의 레이블보다 길어진다. 또한 서브트리가 삽입되는 경우 서브트리의 루트노드의 레이블을 자손노드들이 접두사(Prefix)로 사용하기 때문에 동일 레벨의 형제노드들의 레이블보다 길어지고 동일 위치에 반복적으로 새로운 노드/서브트리가 삽입되면 레이블의 길이가 지속적으로 길어진다. LSDX는 삽입연산이 빈번히 발생하는 동적 XML 환경에서 전체 레이블 저장공간에 대해 비효율적이기 때문에 적합한 레이블링 방법이 아니다.

원형 레이블링 방법은 새로운 노드/서브트리가 삽입됐을 때 *Radius.(start_RN. start_angle*$_{RN}$, *end_RN. end_angle*$_{RN}$)을 기준으로 노드이거나 서브트리의 루트 레이블은 *new_Radius.(old_start_RN. old_start_angle*$_{RN}$, *old_end_RN.old_end_angle*$_{RN}$).(*new_start_RN.new_start_angle*$_{RN}$,*new_end_RN.new_end_angle*$_{RN}$)이고 서브트리의 루트를 제외한 자손노드들의 레이블들은 반지름을 1로 초기화하고 해당 노드의 셀프 레이블 (*start_RN. start_angle, end_RN.end_angle*$_{RN}$)을 할당한다. 이 때 새로 삽입되는 노드의 *new_Radius*는 기준 노드와 비교해서 XML 문서 내 순서를 유지할 수 있도록 할당된다.

동적 XML 환경을 위해 반지름 개념을 적용한 원형 레이블링 방법은 증가할 수 있는 최대 레이블 길이가 한정된다. 4.2에서 새로운 데이타가 업데이트되기 전에 CLS(RN+Radius)가 LSDX보다 항상 좋음을 알 수 있고 업데이트가 빈번하게 발생될수록 LSDX보다 원형 레이블링 방법이 전체 레이블 저장공간에 대해 효율적임을 알 수 있다.

4. 실험 및 평가

4.1 실험 환경

XMark[15]를 이용하여 표 1에 명시된 Scaling Factor에 해당하는 다양한 크기의 XML 문서를 생성하여 사용하였다. 생성된 각 문서는 표 1과 같은 노드 수를 갖는다. 실험 대상 레이블링 방법은 범위 기반 레이블링 방법(RLS : Range-based Labeling Scheme), 섹터 기반 레이블링 방법(SLS : Sector-based Labeling Scheme), 회전 수 개념을 적용한 원형 레이블링 방법(CLS(RN)), 동적 XML 환경을 위해 반지름 개념과 회전 수 개념을 적용한 원형 레이블링 방법(CLS(RN+Radius)), LSDX이다. 각 XML 문서는 이벤트 기반 SAX 파서에 기반하여 레이블을 작성했으며, 실험 환경은 인텔 core2 6300@1.86Ghz 프로세서와 1GB DDR2 메모리가 장착된 마이크로 소프트 윈도우 XP 운영체제에서 자바 가상머신 1.4.2를 이용하였다.

표 1 XML 문서 크기 별 노드 수와 XMark의 Scaling Factor

XML 문서(MB)	노드 수(K)	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	0.61
8.0	116.0	0.69
9.0	131.3	0.77
10.0	145.8	0.087

4.2 전체 레이블 저장공간

그림 11은 다양한 크기의 XML 문서에 대해 RLS, SLS, CLS(RN), CLS(RN+Radius), LSDX를 이용하여 레이블을 작성한 후 전체 레이블에 대한 저장공간을 측

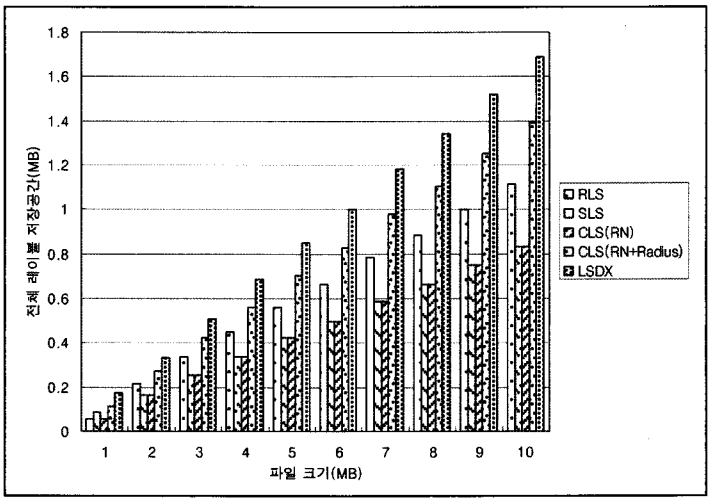


그림 11 다양한 크기의 문서, 레이블링 방법 별 전체 레이블 저장공간

정한 결과이다. LSDX가 항상 가장 큰 저장공간을 사용하고 있는데 레이블의 크기가 XML 트리의 깊이에 비례하여 증가하기 때문이다. CLS(RN)과 SLS는 거의 같은 결과를 보이고 RLS보다 25%의 저장공간 효율을 가지며 실험에서 항상 좋은 결과를 보이고 있다. CLS(RN)은 XML 문서를 원으로 표현하고 회전수 개념을 사용하여 작은 크기의 변수를 반복 사용할 수 있도록 하였기 때문에 RLS보다 효율적인 레이블 저장공간을 사용한다. 또한 CLS의 경우도 사용하는 두 변수 중 반지름을 표현하는 변수가 RLS보다 더 작은 크기의 변수를 할당 받기 때문에 작은 크기의 레이블 저장공간을 사용한다. CLS(RN+Radius)는 동적 XML 환경을 위해 CLS(RN)에 반지름(Radius) 개념을 적용한 것인데 소수를 표현할 수 있는 4byte 변수를 추가했기 때문에 CLS(RN) 또는 SLS의 60%만큼의 저장공간을 더 사용한다.

4.3 레이블 생성 시간

그림 12는 RLS, SLS, CLS(RN+Radius), LSDX를 이용하여 다양한 크기의 XML 문서에 대하여 레이블링 한 시간을 측정한 결과이다. LSDX가 항상 짧은 레이블링 시간을 소모하며 SLS가 항상 긴 레이블링 시간을 소모한다. 반면 RLS와 CLS(RN+Radius)는 비슷한 시간을 소모하지만 네 가지 실험군 중 평균 정도의 시간을 소모한다. SLS는 노드에 레이블링하기 위해서 해당 노드의 형제노드가 몇 개 있는지 파악을 해야 하고 레이블링 알고리즘이 다른 실험군들에 비해 복잡하기 때

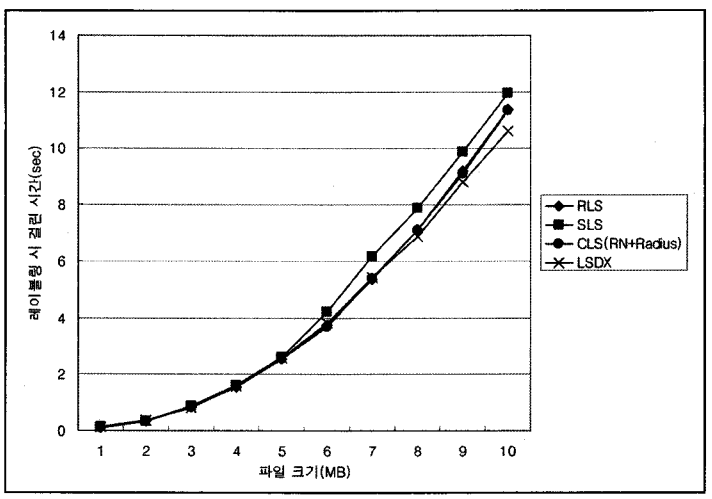


그림 12 다양한 크기의 문서, 레이블링 방법 별 레이블링 시간

문에 가장 많은 시간을 소모한다. RLS와 CLS(RN+Radius)는 LSDX보다 간단한 레이블 구조를 갖지만 size 또는 end 값을 조상노드에 저장하는 시간이 더 걸리기 때문에 LSDX보다 레이블링 시간이 더 소모된다.

4.4 동적 XML 환경에 대한 실험

XML 문서에 대한 레이블링 방법의 초기 연구는 XML 문서가 업데이트되지 않는 정적 XML 환경에 국한하였지만 현실적으로 XML 문서는 업데이트가 빈번히 일어날 수 있기 때문에 최근 연구들은 동적 XML 환경을 고려한 연구들이 주류를 이룬다. 하지만 새로운 노드가 삽입될 경우 기존 노드들에 대한 레이블 재작성 횟수를 최소화해야 하고 이에 따라 새로운 노드 삽입 시 적은 레이블링 시간을 소요해야 한다. 이 두 관건들에 대해 4.4.1과 4.4.2의 실험 결과를 통해 CLS의 우수성을 보인다. 또한 삽입되는 노드에 대해 작은 크기의 레이블을 할당하여 전체 레이블 저장공간의 효율을 얻는 것도 동적 레이블링 방법이 갖추어야 할 요건인데 이 내용은 3.3.3에서 LSDX보다 CLS가 우수함을 밝혔다.

4.4.1 리레이블링이 필요한 노드 수

표 2는 RLS, SLS, CLS(RN+Radius), LSDX를 이용하여 다양한 크기의 XML 문서에 대하여 1000개의 노드들을 삽입했을 때 레이블 재작성이 필요한 노드들의 수를 더한 결과이다. 이 실험에서 RLS는 새로 삽입되는 노드들에 대한 레이블링을 위한 여분의 범위(range)를 남겨두지 않음을 가정한다. RLS는 다른 실험군에 비해 레이블 재작성이 필요한 기존 노드들의 수가 월등하게 많음을 알 수 있다. 2장의 관련연구에서 언급했듯이 RLS는 (order, size)로 구성되어 있으며 새로운 노드가 삽입되면 그림 10(a)에서처럼 삽입되는 노드의 조상노드들과 order가 큰 모든 노드들에 대해 레이블 재작성을 해 주어야 하기 때문이다. SLS는 RLS에 비해 96~97% 정도 레이블 재작성을 해 주어야 하는 노드들의 수가 적다. SLS는 새로운 노드가 삽입될 수 있는 조건이 만족되면 그림 10(b)에서처럼 해당 노드의 후임자 노드/서브트리에 대해서 레이블 재작성을 해야 하고 만족되지 않으면 그림 10(c)에서처럼 해당 노드의 모든 형제노드/서브트리에 대해 레이블 재작성을 해야 한다. 반면 CLS(RN+Radius)와 LSDX의 경우 항상 레이블 재작성을 해주어야 하는 노드 수가 0개로 가장 좋은 결과를 보인다. CLS(RN+Radius)는 새로운 노드가 삽입될 때 기준이 되는 노드의 레이블을 가져와 반지름을 증가/감소 시키고 끝에 해당 노드 또는 해당 노드를 루트로 하는 서브트리에 대한 셀프 레이블들을 구해서 할당하기 때문에 기존 노드들에 대한 레이블 재작성이 필요하지 않다. CLS(RN+Radius)와 마찬가지로 LSDX도 새로운 노드가 삽입될 때 기준이 되는 노드의 레이블을 가져와서 삽입되는 노드의 레이블을 계산하여 할당해주기 때문에 기존 노드들에 대한 레이블 재작성이 필요하지 않다.

4.4.2 삽입 연산 수행 시 걸린 시간

그림 13은 RLS, SLS, CLS(RN+Radius), LSDX를 이용하여 다양한 크기의 XML 문서에 대하여 1000개의 노드들을 삽입했을 때 기존 노드들에 대한 레이블 재작성과 삽입되는 노드들에 대한 레이블링 시간을 더한 결과이다. 표 2의 결과와 동일하게 RLS가 가장 많은 수의 노드들에 대해 레이블 재작성이 필요하기 때문에 노드 삽입에 대한 레이블링 종료까지 걸리는 시간이 가장 길다. CLS는 RLS에 비해 레이블 재작성을 해주어야

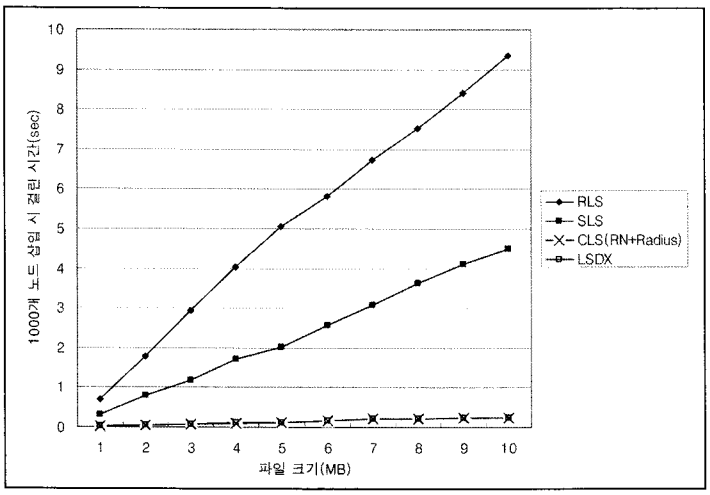


그림 13 1,000개의 노드를 삽입했을 경우 레이블 재작성이 필요한 시간

표 2 1,000개의 노드를 삽입했을 경우 레이블 재작성이 필요한 노드 수

File Size (MB)	RLS	SLS	CLS (RN+Radius)	LSDX
1	6,412,362	34,968	0	0
2	11,748,261	47,324	0	0
3	18,012,056	56,961	0	0
4	23,647,409	83,245	0	0
5	30,835,920	43,910	0	0
6	36,259,872	106,083	0	0
7	41,412,454	139,351	0	0
8	47,400,021	159,689	0	0
9	54,705,299	212,029	0	0
10	61,136,074	182,023	0	0

하는 노드 수가 96~97% 적음에도 걸린 시간은 45~50% 정도 적게 걸림을 알 수 있는데 CLS의 레이블링 방법의 복잡도가 RLS를 비롯한 다른 실험군에 비해 복잡하기 때문이다. CLS(RN+Radius)와 LSDX가 비슷한 시간을 소요하며 가장 좋은 결과를 보이는데 기존 노드들에 대한 레이블 재작성이 필요하지 않으면서 새로운 노드들에 대한 레이블링 방법도 SLS보다 간단하기 때문이다.

5. 결론 및 향후 연구과제

인터넷과 유비쿼터스 환경에서 XML 데이타는 데이타의 교환과 저장 등의 표준으로 광범위하게 사용되고 있다. XML 데이타를 효과적으로 이용하는 방법으로 레이블링 방법이 연구되고 있지만 기존 연구들에서는 XML 문서에 대해 큰 레이블 저장공간이 필요하다는 문제점을 갖는다. 또한, XML 문서의 업데이트가 고려되는 환경에 대한 연구들이 제시되고 있다. 그러나 새로운 데이타의 삽입 시 기존 데이타들의 레이블을 재작성하기 때문에 새로운 데이타에 대한 레이블링 시간이 오래 걸리고 동일 위치에 반복적으로 새로운 데이타가 삽입될 경우 지속적으로 레이블 길이가 길어져서 큰 레이블 저장공간이 필요하게 되는 문제점이 있다. 이 논문에서는 XML 문서를 원으로 이해하는 새로운 시도를 하였고 매우 큰 XML 문서에 대한 전체 레이블 저장공간의 절감에 대한 고려를 했다는 점, 동적 XML 환경에 효율적인 레이블링 방법을 제시했다는 점에 의미를 둘 수 있다. XML 문서를 원으로 이해함으로써 회전수와 부모원/ 자식원 개념을 사용하여 매우 큰 XML 문서에 대한 전체 레이블 저장공간을 줄였다. 또한 반지름(소수 표현 가능) 개념을 적용하여 새로운 데이타의 삽입 시 기존 데이타들의 레이블 재 작성이 발생하지 않기 때문에 XML 문서에 대한 업데이트 연산 시간을 절감할 수 있고, 레이블 길이가 한정되어 레이블 저장공간의 효율을 얻을 수 있다. 그리고 삽입되는 데이타들의 레이블 간 충돌이 발생하지 않는다.

원형 레이블링 방법은 모든 형태의 XML 문서에 유용하지만, 특히 동일한 형식의 데이타가 반복되는 병원(환자 데이타의 반복), 옥션(물품 데이타의 반복), 회사(사원 데이타의 반복) 등에 유용하다. 또한 업데이트가 빈번히 발생하는 환경에도 유용하게 활용될 수 있다.

그러나 원형 레이블링 방법은 반지름에 소수를 표현할 수 있는 변수를 할당하기 때문에 해당 변수가 표현할 수 있는 값의 범위를 벗어나는 경우가 발생할 수 있다. 결국 전체 XML 문서에 대한 레이블링을 다시 해주거나 이 문제점을 보완해줄 수 있는 새로운 변수가 필요하게 되기 때문에 한계가 있다.

레이블링 방법은 접근제어 메커니즘과 연관되어 하나의 시스템의 일부로 사용될 수 있다. 레이블링 방법과 접근제어 메커니즘에 대한 대부분의 연구들은 고유의 목적에 맞는 효율적인 방법 또는 메커니즘을 제시하는데 치우쳐있다. 그러므로 향후 연구과제는 접근제어 메커니즘에 레이블링 방법을 접목시켜 기존의 접근제어 메커니즘보다 효율적인 시스템을 모색하는 것이다.

참 고 문 헌

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

[2] J. Clark and S. DeRose, "XML Path Language (XPath) Version 1.0. W3C Recommendation," World Wide Web Consortium, 1999.

[3] S. Boag, D. Chamberlin, M. Fernandez, D. Florescu, J. Robie, J. Simeon, and M. Stefanescu, "XQuery 1.0: An XML Query Language," W3C Working Draft, Vol.15, 2002.

[4] S. Al-Khalifa, H. V. Jagadish, N. Koudas, J. M. Patel, D. Srivastava, and Y. Wu, "Structural Joins: A Primitive for Efficient XML Query Pattern Matching," Proc. of the 10th International Conference on Data Engineering (ICDE), pp. 141-154, 2002.

[5] J. McHugh and J. Widom, "Query Optimization for XML," Proc. of the 25th International Conference on Very Large Data Bases (VLDB), pp. 315-326, 1999.

[6] P. Dietz, "Maintaining order in a linked list," Proc. of the 14th annual ACM symposium on Theory of computing, pp. 122-127, 1982.

[7] Q. Li and B. Moon, "Indexing and Querying XML Data for Regular Path Expressions," The VLDB Journal, pp. 361-370, 2001.

[8] 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.
[9] P. O'Neil, E. O'Neil, S. Pal, I. Cseri, G. Schaller, and N. Westbury, "ORDPATHs: insert-friendly XML node labels," in Proc. of the 2004 ACM SIGMOD international conference on Management of data, Paris, France, 2004, pp. 903-908.
[10] A. Khaing and N. Thein, "A Persistent Labeling Scheme for Dynamic Ordered XML Trees," in Proc. of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence, 2006, pp. 498-501.
[11] 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), 2004, pp. 66-78.
[12] R. Thonangi, "A Concise Labeling Scheme for XML Data," in Proc. of ACM SIGMOD, COMAD, 2006.
[13] V. Gaede and O. Gunther, "Multidimensional access methods," ACM Computing Surveys (CSUR), Vol. 30, pp. 170-231, 1998.
[14] J.-G. Lee, K.-Y. Whang, W.-S. Han, and I.-Y. Song, "The dynamic predicate: integrating access control with query processing in XML databases," The VLDB Journal, Vol.16, pp. 371-387, 2007.
[15] 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: VLDB Endowment, 2002.

김 진 영

2007년 서강대학교 컴퓨터공학과 졸업(학사). 2009년 서강대학교 컴퓨터공학과 졸업(공학석사). 관심분야는 XML, 레이블링, 접근제어, 시멘틱웹, 웹디비 등

박 석

1978년 서울대학교 계산통계학과(이학사) 1980년 한국과학기술원 전산학과(공학석사). 1983년 한국과학기술원 전산학과(공학박사). 1983년 9월~현재 서강대학교 컴퓨터공학과 교수. 1989년~1991년/2002년~2003년 University of Virginia 방문교수. 1997년 2월~현재 한국정보보호학회 이사. 2005년 한국정보과학회 부회장. 2004년 1월~2005년 12월 한국정보과학회 편집위원장. 1999년~2007년 DASFAA Steering Committee 멤버. 관심분야는 데이타베이스 보안, 실시간 시스템, 트랜잭션 관리, 데이타웨어하우스, 웹 데이타베이스 임