빈번히 갱신되는 XML문서에 대한 프라임 넘버 레이블링 기법

(An Improved Method of the Prime Number Labeling Scheme for Dynamic XML Documents)

유 지 열 † 유 상 원 † 김 형 주 ††
(Ji you Yoo) (Sang won Yoo) (Hyoung Joo Kim)

요 약 XML 레이블링 기법은 엘리먼트 간의 조상-자손 관계 및 형제들 간의 순서 등을 쉽게 결정할 수 있도록 하는 색인을 위한 인코딩(encoding)이라고 할 수 있다. 특히 근래에는 Web Services 및 AXML (Active XML)과 같은 기술에 동적 XML 문서가 등장하게 되었고 이로 인해 동적 XML 레이블링 기법이 필요하게 되었다. 대표적인 동적 레이블링 기법인 프라임 넘버 레이블링(prime number labeling)기법은 XML 문서의 엘리먼트 간의 부모-자식간의 관계를 소수의 특성을 이용하여 결정할 수 있도록 하는 기법이다. 이 기법은 새로운 엘리먼트가 삽입이 될 때 부여되는 레이블이 기존의 레이블 정보를 변화시키지 않는다는 장점이 있으나 형제간의 순서를 결정하는 순서 값(Order number)을 갱신하기 위해 추가의 연산 및 자료구조를 유지하는 비용을 갖는 단점을 가지고 있다. 본 논문에서는 이러한 비용을 줄이기 위해 요소의 순서정보를 나타내는 오더 값을 공유하는 기법과 삽입되는 위치에 따라 레이블의 값 또는 오더 값을 이용하여 형제간의 순서를 결정할 수는 방법을 제안하여 기존방법보다 적은 비용으로 처리할 수 있도록 하였다.

키워드 : XML, 레이블링 기법, 프라임 넘버

Abstract An XML labeling scheme is an efficient encoding method to determine the ancestor-descendant relationships of elements and the orders of siblings. Recently, many dynamic XML documents have appeared in the Web Services and the AXML(the Active XML), so we need to manage them with a dynamic XML labeling scheme. The prime number labeling scheme is a representative scheme which supports dynamic XML documents. It determines the ancestor-descendant relationships between two elements with the feature of prime numbers. When a new element is inserted into the XML document using this scheme, it has an advantage that an assigning the label of new element don't change the label values of existing nodes. But it has to have additional expensive operations and data structure for maintaining the orders of siblings.

In this paper, we suggest the order number sharing method and algorithms categorized by the insertion positions of new nodes. They greatly minimize the existing method's sibling order maintenance cost.

Key words : XML, XML labeling scheme, prime number labeling

1. 서 론

XML(eXtensible Markup Language)[1]은 1998년 2월에 W3C(World Wide Web Consortium)에서 인터넷 문서 교환의 표준으로 채택된 이래로 효율적인 XML 처리를 위한 연구가 지속적으로 진행되고 있다.

XML에 대한 질의는 데이타의 경로(path)를 표현하는 XPath[2]와 XQuery[3]등을 사용하며 실제 질의처리를 위해서 XML 트리를 탐색하여 원하는 데이타를 찾게 된다. 이러한 XML 트리의 탐색 범위를 줄이기 위해서 XML 트리의 각 노드에 레이블(label)을 부여하고 레이블 정보를 이용해서 엘리먼트 사이의 관계를 쉽게 표현

· 본 연구는 정보통신부 및 정보통신연구진흥원의 대학IT연구센터 육성지원 사업(IITA-2005-C1090-0502-0016)과 BK21의 지원을 받아 수행되었음
† 학생회원 : 서울대학교 컴퓨터공학과
sogod7@daum.net
swyoo@oopsla.snu.ac.kr
†† 종신회원 : 서울대학교 컴퓨터공학과 교수
hjk@oopsla.snu.ac.kr
논문접수 : 2005년 3월 22일
심사완료 : 2005년 10월 18일

하고 결정할 수 있는 레이블링 기법(labeling scheme)을 사용한다. 레이블링 기법은 엘리먼트 간의 조상-자손 관계 및 형제들 간의 순서 등을 쉽게 결정할 수 있도록 하는 색인을 위한 인코딩(*encoding*)이라고 할 수 있다. 대표적인 레이블링 기법으로는 [4-6]이 있다.

최근 들어 웹을 통한 데이타 통합에 대한 관심과 필요성이 증대되어 XML기술을 기반으로 하는 Web Services[7] 기술이 등장하게 되었고 대표적인 예로는 프랑스 정보기술연구소(INRIA)[8]에서 개발한 AXML (Active XML)[9]이 Web Services의 프레임워크(framework)로 등장하게 되었다. AXML은 빈번하게 업데이트가 되는 동적(dynamic)인 XML 문서의 대표적인 예라고 할 수 있다.

동적인 XML 문서의 질의 처리를 위해서는 문서 내의 엘리먼트 들의 삽입, 삭제 시에도 이를 쉽게 반영할 수 있는 레이블링 기법이 필요하다. 문서의 갱신에 대한 고려를 하지 않은 기존의 레이블링 기법들은 문서의 갱신이 일어날 때마다 변화된 레이블 정보를 반영하기 위해서 전체 XML 트리를 재탐색하여 전체 노드의 레이블을 다시 계산하는 비용이 들게 된다. 이 논문에서는 이와 같은 과정을 리레이블링(relabeling) 이라고 부르기로 한다. 그림 1은 XRel[4]의 방법으로 이미 레이블링 되어있는 문서에 새로운 "E-mail" 노드가 삽입이 되었을 경우에 전체 레이블이 새롭게 리레이블링 되는 과정을 보여준다. 문서의 크기가 커질수록 리레이블링 비용이 심각하게 증가되는 것을 쉽게 예측할 수 있다.

동적인(dynamic) XML 문서의 등장으로 변경된 정보를 잘 반영하고 다른 부분의 레이블에 큰 영향을 주지 않는 레이블링 기법이 등장하게 되었는데 가장 대표적인 것이 프라임 넘버(prime number) 즉 소수로 레이블링을 하는 프라임 넘버 레이블링 기법(prime number labeling scheme)[10]이다. 이 기법은 XML 트리에서 각 노드의 레이블을 소수로 부여하여 조상-후손관계를 나타낼 수 있는 특성을 가지며 문서의 갱신 시 다른 노드의 레이블에 영향을 주지 않도록 설계되어 있다. 그러나 문서가 갱신될 때 엘리먼트 간의 순서를 나타내는 순서정보 값(order number)이 갱신되는 비용이 XML 트리의 상당히 많은 부분을 재탐색하고 갱신된 순서정보를 재 기록하는 등 많은 갱신비용이 들게 된다. 본 논문에서는 이러한 비용을 줄이기 위해 트리의 재탐색 범위와 재기록 되는 정보의 양을 감소시킬 수 있는 방법을 제안한다.

본 논문의 구성은 다음과 같다. 2장에서 관련연구를 소개하고 3장에서 프라임 넘버 레이블링 기법과 문제점에 대해 자세히 소개한다. 4장에서는 이러한 문제점을 해결하기 위해 제안한 기법과 알고리즘에 대해 그리고 5장에서는 실험결과를 설명하고 6장에서는 결론을 기술한다.

2. 관련 연구

최근까지 연구된 레이블링 기법은 크게 대상 XML문서의 갱신 유무에 따라서 정적 혹은 동적인 레이블링 기법으로 나눌 수 있다. [4]는 대표적인 정적 레이블링 기법 중 하나로서 깊이 우선 탐색(depth-first traversal)으로 XML 트리를 운행하면서 각 노드에 두개의 정수를 할당하는 방식이다. 순서쌍의 처음 값은 start-point로 카운터(counter)를 1씩 증가시키면서 각 노드에 할당하고 나머지 값은 깊이 우선 탐색의 역방향으로 트리를 탐색하면서 카운터의 값을 할당하여 *end-point*를 각 노드에 부여하여 숫자 쌍을 만든다. 이 숫자쌍이 범위연산을 위해 사용되며 조상-후손 관계 결정은 후손노드의 범위가 조상노드의 범위에 포함된다면 그 관계가 성립된다. 이러한 방식은 XML문서의 일부가 변경이 되더라도 전체 XML 트리를 재탐색하여 모든 노드에 대해서 리레이블링을 해야 하는 큰 비용이 들게 된다.

[5]는 [4]의 범위 포함관계를 통해 조상-후손 관계를 결정하는 방식과 비슷한 방식이나 각 노드의 할당된 숫자 쌍을 이루는 값이 각각 order, order+size로 이루어져 있다. 조상-후손관계 결정은 두 노드 N1, N2가 존재하고 N1이 N2의 조상이라면 order(N1)<order(N2)

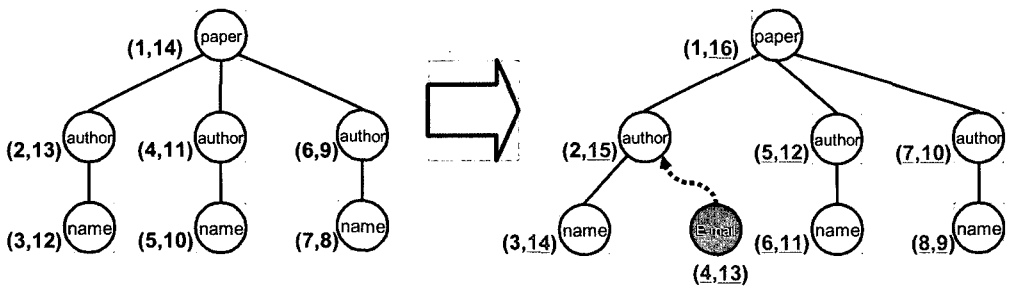

그림 1 XRel의 리레이블링 과정

<order(N1)+size(N1)인 관계가 성립된다. 이 방식도 마찬가지로 갱신이 일어날 경우에는 각 노드에 할당된 레이블을 다시 부여하는 비용이 많이 들게 된다.

이러한 문제점을 해결하기 위해서 동적 XML 문서에 적합한 레이블링 기법이 프라임 넘버를 이용한 레이블링 기법[10]이다. 이 기법은 XML문서의 갱신 시 전체 노드의 레이블을 새로 부여하는 비용을 줄임으로서 기존의 정적 레이블링 기법보다 동적 XML문서에 훨씬 적합한 방법이다. 이 기법의 자세한 설명은 3장 프라임 넘버 레이블링기법에서 기술하기로 한다.

3. 프라임 넘버 레이블링 기법

3.1에서는 프라임 넘버 레이블링 기법의 조상-후손 관계 결정법을 자세히 설명하고 3.2에서는 XML문서의 갱신이 일어날 경우를 살펴본다. 형제간의 순서 결정법과 이 기법의 문제점에 대해 3.3에서 설명하도록 한다.

3.1 조상-후손 관계

프라임 넘버 레이블링 기법[10]은 프라임 넘버의 특성인 1과 자기 자신으로만 나누었을 때 나머지가 0인 성질을 이용하여 XML 트리의 각 노드에 레이블을 부여한다.

제일 먼저 XML 트리의 루트(root) 노드에 1로 레이블을 주고 그 이하 서브 트리를 구성하는 각 노드에 프라임 넘버의 순서대로 레이블을 부여한다. 여기서 레이블 값은 양의 정수 값을 의미한 다. 이 레이블을 셀프-레이블(self-label)이라고 부르며 XML 문서를 파싱하는 순서대로, 즉 트리에서 전위순회(preorder traversal)로 부여한다. 또한 자식노드의 레이블은 부모노드의 레이블과 자신에게 부여된 셀프-레이블의 곱으로 만들어진다. 그러므로 프라임 넘버의 특성상 자식노드의 레이블은 부모노드의 레이블로 나뉘어졌을 때 나머지는 0이 되고 그 두 노드는 조상-자손 관계가 성립이 된다. 그림 2에서는 레이블 값 및 레이블 2와 6인 노드사이의 조상-자손 관계를 결정하는 과정을 보여주고 있다.

3.2 갱 신

3.1과 같은 방식으로 레이블이 각 노드에 부여된다면 새로운 노드가 삽입되거나 삭제가 되었을 때 다른 노드의 레이블 값에 영향을 주지 않고 새로운 레이블을 부여할 수 있다. 그 이유는 새로운 노드가 삽입될 때 부여되는 레이블 값은 기존의 노드들이 갖는 레이블 값의 연속된 값이 아니라 그동안 부여되지 않은 새로운 값으로 레이블 생성하기 때문이다.

예를 들어 그림 3에서 4번째 위치에 새로운 노드가 삽입된다면 새로 삽입되는 노드의 셀프-레이블은 지금까지 부여된 프라임 넘버의 다음번 프라임 넘버 17을 부여받게 되고 그 값과 부모노드의 레이블 값 2를 곱한 값 34를 자신의 레이블로 할당받게 된다. 또한 노드가 삭제될 때도 삭제되는 노드만 제거되면 되고 삽입과 같이 나머지 노드의 레이블 값에는 영향을 주지 않는다. 그림 3은 새로운 노드의 삽입과정을 보여주고 있다.

3.3 형제간의 순서 결정 및 문제점

질의 중에는 조상-후손 관계 뿐 아니라 형제간의 순

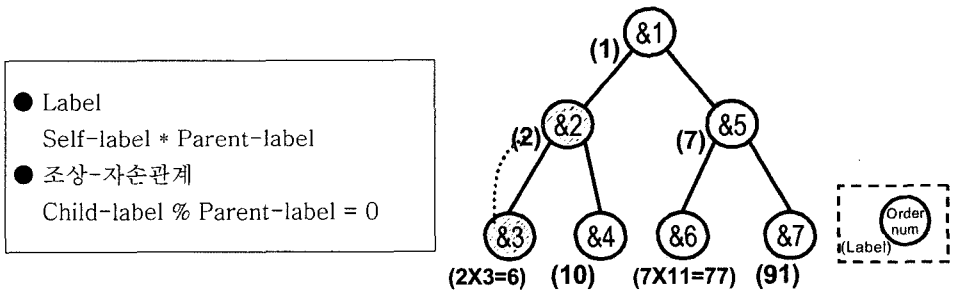

그림 2 조상-자손 관계 결정

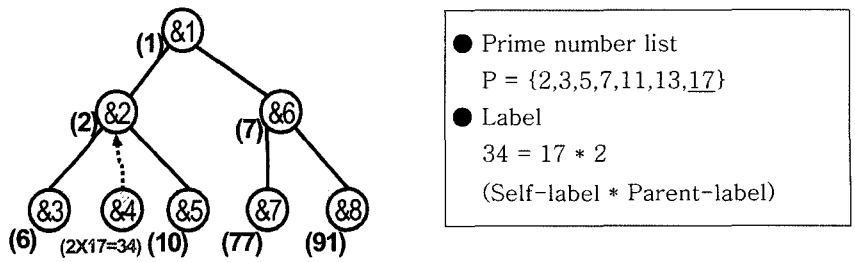

그림 3 새로운 노드 삽입

서를 결정해야만 하는 경우도 있다. 예를 들어 그림 1의 XML 문서에 XPath 형식으로 "PAPER/AUTHOR[2]" 와 같은 질의가 주어진다면 우리는 AUTHOR 형제 노드간의 순서를 알아야만 한다. 그런데 조상-후손 관계 결정법과는 달리 형제간의 순서 결정은 레이블의 값만을 가지고 결정할 수 없는 경우가 생길 수 있다. 3.2에서 언급했던 노드의 삽입 시 새로운 노드에 대한 레이블 부여방법에 따라 새로운 노드의 레이블은 항상 기존의 노드보다 큰 셀프-레이블을 부여받게 됨으로 새로운 노드가 형제들 사이에 삽입이 된다면 형제노드 중에 새로 삽입되는 노드의 레이블이 가장 크게 된다. 그러므로 노드에 부여된 레이블 값만을 가지고는 순서를 결정할 수 없다. 그래서 형제노드간의 순서를 결정하기 위해 레이블 정보 이외의 순서 정보를 나타내는 값을 유지해야 하는데 [10]에서는 이를 위해 SC 테이블이라는 자료구조를 유지한다.

SC 테이블의 스키마를 살펴본다면 주요 필드로 max, sc value 가 있고 max 필드의 값은 보통 5개의 그룹으로 이루어진 프라임 넘버 중 가장 큰 값을 나타내고 sc value 필드 값은 그 그룹의 모든 프라임 넘버를 가지고 중국의 나머지 정리(Chinese remainder theorem)[10]를 적용시킨 SC값을 기록해 놓는다. SC 테이블을 이용하여 형제간의 순서 결정을 하는 방법은 해당 노드에 부여된 셀프-레이블 값으로 자신이 속한 그룹의 SC 값을 나누어 그 나머지를 가지고 전체 문서에서 자신의 순서를 나타내는 글로벌 오더(Global order)값으로 형제간의 순서를 결정하게 된다. 형제간의 순서 결정을 위한 정보를 유지하는 비용은 XML 문서의 갱신 시 증가하게 되는데 예를 들어 XML 트리에 새로운 노드가 삽입될 때 다른 노드의 레이블 값에 영향을 주지 않지만 순서정보를 나타내는 글로벌 오더 값은 XML 트리를 재탐색해서 각 노드에 다시 할당해야 하는 비용이 든다. 또한 SC 테이블 내의 각 그룹에 해당하는 SC 값을 다시 계산하는 비용이 들 뿐만 아니라 해당 그룹에 속하는 노드들과 SC 테이블간의 연결정보 또한 다시 계산해야 하는 비용이 든다. 만약 메모리 크기보다 문서의 크기가 더 크게 될 경우에는 트리의 재탐색 연산으로 인해 디스크 I/O 비용이 증가하게 된다. 또한 문서의 크기가 커짐에 따라서 탐색하는 범위가 늘어나기 때문에 비용이 증가되는 것을 알 수 있다. 그림 4는 노드의 추가 시 갱신되는 SC 테이블의 값들을 보여준다. 결론적으로 정적인 문서들을 대상으로 한 레이블링 기법의 문제점을 해결하기 위해 제안된 프라임 넘버 레이블링 기법도 형제노드간의 순서정보 유지에 많은 오버헤드가 존재함을 알 수 있다.

4. 제안 방법

본 논문에서는 XML 트리에서 노드의 갱신에 따른 SC 테이블의 유지비용을 줄이기 위해서 기존의 SC 테이블을 쓰지 않고 레이블과 글로벌 오더 정보만을 가지고 자식간의 순서를 결정할 수 있는 방법을 제안한다. 부모-자식간의 관계를 결정하는 방법과 레이블 부여방법은 기존 방법과 동일하다. 레이블이 부여된 XML 문서에 새로운 노드가 삽입되었다고 가정한다면 새로운 노드의 레이블은 기존 방법대로 부여되나 글로벌 오더는 기존의 방법과 다르게 부여된다.

4.1 글로벌 오더 값 공유

글로벌 오더 부여방법은 새 노드가 삽입될 때 삽입되는 위치의 바로 이전 노드의 글로벌 오더를 공유하는 방법을 쓴다. 이 방법을 쓰는 이유는 새로 삽입된 노드로 인해서 기존의 다른 노드들의 글로벌 오더 값에 영향을 주지 않기 위함이다. 이후에 나오는 글로벌 오더 값은 간단히 오더 값이라고 부르기로 한다. 예를 들어 그림 4에서 4번째 위치에 노드가 삽입되었을 때 기존 방법처럼 오더 값 4를 갖지 않고 이전 노드의 오더 값 3을 공유하게 된다. 이 기법을 이용하여 형제간의 순서를 결정하기 위해서는 다음과 같은 알고리즘을 제안한다.

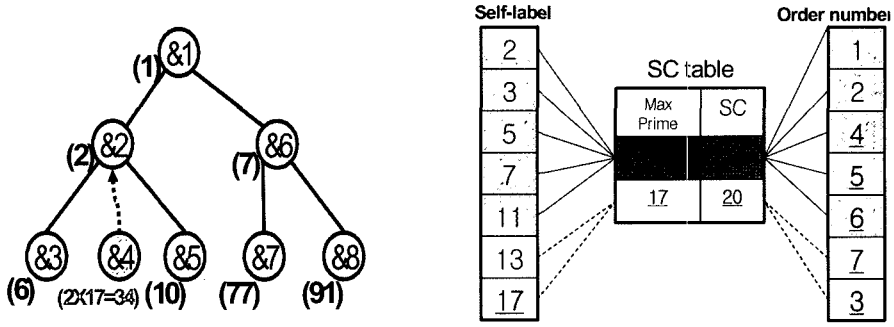

그림 4 노드 삽입 시 SC 테이블 갱신

```
Function CompareOrder(N1, N2)
  N1, N2: 서로 다른 두 노드
  order_num: 오더 값
  label_value: 레이블 값
begin
  if N1.order_num = N2.order_num  then
     Determine order by comparing each label_value
else
     Determine order by comparing each order_num
end
```

알고리즘 1 두 노드간의 순서 비교

형제간의 순서를 결정할 때 먼저 오더 값을 비교하고 오더 값이 서로 같으면 레이블 값을 비교해서 순서를 결정하게 된다. 그림 5에서 레이블 6과 레이블 34의 노드간의 순서를 결정하기 위해서 알고리즘1을 적용시키면 두 노드의 오더 값이 서로 같기 때문에 레이블 값으로 순서를 결정하면 레이블 6인 노드의 순서가 레이블 34 노드의 순서 보다 앞선다는 것을 알 수 있다. 반대로 레이블 77 노드와 91 노드는 오더 값이 서로 다르기 때문에 할당된 오더 값으로 순서를 결정할 수 있다.

새로운 노드가 삽입되는 위치에 따라 제시한 알고리즘을 적용하기 위해서 크게 3가지로 나누어 처리하는 과정이 필요하다. 표 1에서는 이러한 3가지 경우를 정의한다. 또한 표 2에서는 제시하는 알고리즘에 필요한 기본연산을 설명하고 있다. 기본 연산은 주로 트리를 탐색하는 연산과 기존 방식[10]과 공통으로 사용하는 새로운 레이블을 얻는 연산 등이 포함되어 있다.

표 1에서 Case 1은 새로운 노드의 삽입 위치가 부모 노드를 기준으로 마지막 자식노드 위치이며 Case 2와 함께 알고리즘 1을 바로 적용할 수 있는 경우이다. 특히 Case 2는 한개 이상의 노드를 깊이 우선 탐색 순으로 연속해서 삽입할 경우라고 볼 수 있으며 좀더 직관적으로 각기 상이한 위치에 연속적으로 삽입하는 경우라고 볼 수 있다. 일반적인 XML 파서가 XML 문서를 파싱하는 순서와 같기 때문에 Case 2의 여러 노드의 연속 삽입 순서는 자연스럽다고 할 수 있다.

표 1 삽입 위치에 따른 분류

Case 1 : 마지막 자식노드 위치
Case 2 : 오더 값이 다른 노드 사이
Case 3 : 오더 값을 공유한 노드 사이

표 2 기본 연산

Operations	Contents
Prev_Sibling(N)	노드 N의 이전 노드를 반환
Next_Sibling(N)	노드 N의 다음 노드를 반환
Parent_node(N)	노드 N의 부모 노드를 반환
getNewLabel()	새로운 레이블 값 생성

또한 노드의 삽입, 삭제 시 오더 값 공유기법을 사용했기 때문에 다른 노드의 오더 값을 갱신할 필요가 없다. 그러므로 오더 값 갱신 비용은 0이라고 할 수 있다.

Case 3인 경우는 노드의 삽입 위치가 오더 값을 공유하는 두 노드 사이에 있는 경우이다. 이 경우는 Case 2와 반대로 깊이 우선 탐색 반대 순서로 한개 이상의 노드들이 연속적으로 삽입되는 경우이거나 이전에 삽입된 위치에 다시 삽입이 되는 경우 및 동일한 위치에 연속적으로 삽입되는 경우라고 볼 수 있다. 일반적으로 AXML[9]과 같은 동적 XML을 처리하는 시스템에서는 잘 일어나지 않는 경우라고 볼 수 있다. 그림 6은 노드가 동일한 위치에 연속적으로 삽입되는 모습을 보여주고 있다.

Case 3에서는 레이블 값으로 순서를 결정할 수 있도록 링크노드(link node)를 새로운 노드의 자리에 삽입하는 과정이 필요하다. 여기서 링크노드란 연결되는 노드의 글로벌 오더 정보와 레이블 값을 갖는 노드를 말하며 링크노드 자신의 레이블은 기존의 레이블 부여 방법과 같이 새로 부여 받는다. 자세한 링크노드 생성부분은 알고리즘 2에서 제안한다.

링크노드를 삽입하지 않고 기존의 일반노드의 레이블을 직접 갱신하는 경우에는 갱신되는 노드를 루트로 하는 서브트리의 모든 노드의 레이블도 갱신해주어야 하

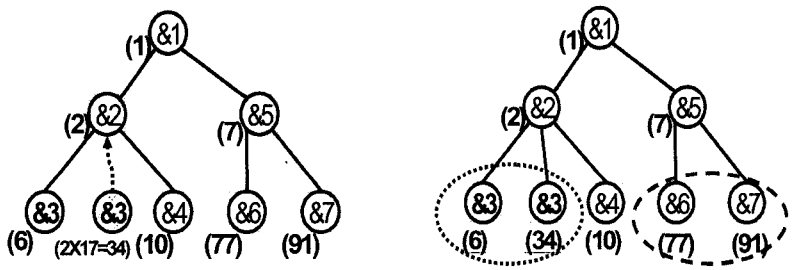

그림 5 노드 삽입 시 오더 값 공유 및 순서결정

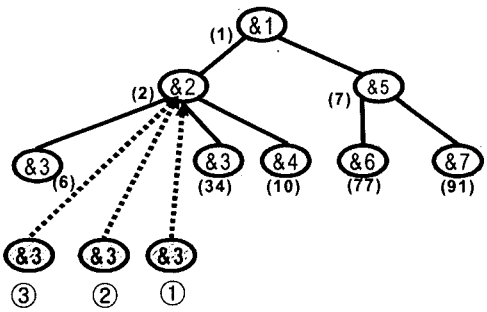

그림 6 동일한 위치의 노드 연속 삽입

```
Functionmake Link_node(TN)
TN: Target Node, LN:  Link Node
   order_num:  오더 값
   label_value: 레이블 값
   tagname: 태그 명
begin
LN.order_num := TN.order_num
   LN.label_value := getNewLabel()
   LN.tagname := TN.tagname
   LN.mark := TN.label_value
  return LN
 end
```

알고리즘 2 링크노드 생성

는 비용이 들기 때문에 링크노드를 삽입하는 것보다는 훨씬 비효율적일 수 있다. 링크노드는 형제간의 순서 결정에만 사용되고 조상-자손간의 관계 결정 과정에서는 제외된다.

이와 같이 링크노드를 삽입함으로서 새로 삽입되는 노드의 레이블 값보다 큰 레이블을 바로 다음번 노드가 갖게 되어 자식간의 순서를 레이블 값을 사용하여 결정할 수 있다.

4.2 노드의 삽입 및 삭제

4.1에서 삽입되는 위치에 따라 크게 세 가지 경우로 나눈 것을 기준으로 실제 삽입 알고리즘을 설명하기로 한다.

```
Function Insertion(N)
  N : 노드
  order_num: 오더 값
  label_value: 레이블 값
 begin
0   N.label_value = getNewLabel();
1   if ( Prev_Sibling(N) is not exist ) then
2     N.order_num = Parent_node(N).order_num
3   else if (Next_Sibling(N) is not exist) then
4       N.order_num = Prev_Sibling(N).order_num
5   else
6       N.order_num = Prev_Sibling(N).order_num
7       while (Next_Sibling(N) is not exist)
8       {
9         if (Next_Sibling(N).order_num != N.order_num) then
10            break
11        else
12            if (Next_Sibling(N) is a link node) then
13                Next_Sibling(N).label = getNewLabel()
14                N = Next_Sibling(N)
15            else
16                makeLink_node()
17             Insert the link node into the next sibling position
18                N = Next_Sibling(N)
19      }
 end
```

알고리즘 3 새로운 노드 삽입

Case 3은 부가적으로 삽입되는 노드의 수를 줄이기 위해서 두 가지 경우로 나누어서 생각할 수 있는데 새

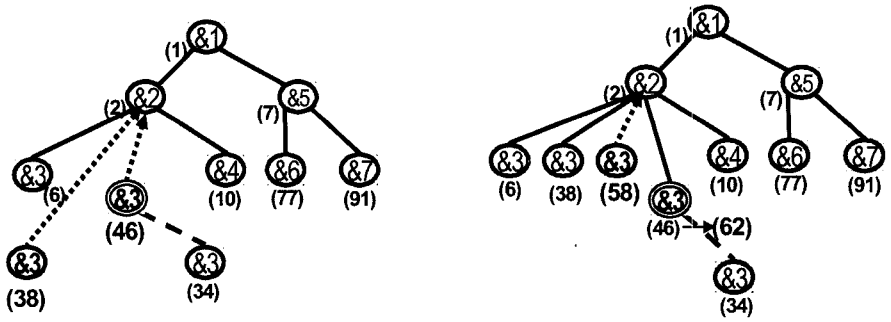

그림 7 링크노드 삽입 및 링크노드 레이블 갱신

로운 노드 N이 삽입된다면 Next_Sibling(N)이 링크노드인지 아닌지에 따라서 다르게 처리하게 된다. Next_Sibling(N)이 링크노드가 아닌 경우에는 새로운 링크노드를 삽입하고 링크노드인 경우에는 그 링크노드의 레이블 값을 새로 갱신한다. 이러한 과정을 새로운 노드 삽입 이후의 오더 값이 같은 노드에 대해서 수행한다. 그림 7은 Case 3의 링크노드 삽입 및 링크노드의 레이블 갱신 과정을 보여준다. 또한 알고리즘 3에서는 자세한 삽입과정을 설명한다.

Case 3에서 노드삽입 시 Next_Sibling(N)이 링크노드가 아닌 경우에는 오더 값 갱신비용은 오더 값이 같은 형제노드의 개수만큼 Next_Sibling(N) 연산 및 makeLink_node() 연산을 하는 비용이 든다. 또한 Next_Sibling(N)이 링크노드일 경우에는 마찬가지로 오더 값이 같은 형제노드의 개수만큼 탐색하는 비용과 링크노드의 레이블을 갱신하는 비용이 든다.

노드를 삭제할 경우에는 삭제되는 노드에 대한 링크노드 유무를 체크한 후 링크노드가 있는 경우는 링크노드와 같이 삭제하고 그렇지 않을 경우에는 대상 노드만 삭제하면 된다.

5. 실 험

5.1 실험 환경

실험환경은 CPU가 Intel Pentium 1 GHz (Celeron), 메모리 256MB이고 운영체제는 Windows XP professional인 PC에서 수행 하였다. 구현 언어는 JAVA (JDK1.4.2)를 사용하였다. XML Parser는 SAX parser로 Xerces [11] 버전 2.6.0을 사용하였으며 또한 DBMS는 MySQL 4.0을 사용하였다. 실험 데이타는 Shakespeare 2.0[13]으로 셰익스피어의 희곡들의 모음을 XML로 기술한 데이타로 웹상에서 공개되어 있는 데이타이다. 이 데이타는 초기에 ASCII 파일로 공개한 이후에 SGML로 포맷이 바뀌었고 최종 1998년에 XML로 기술되었다. Shakespeare 2.0은 40개의 파일로 구성되어 있고 크기는 139Kb에서 282Kb이다. 실험 데이타는 스택(stack) 연산을 이용한 SAX parser를 이용하여 XML 문서를 데이타베이스에 레이블이 부여된 저장 상태로 저장하였다.

5.2 실험 결과

실험은 4장에서 삽입 위치에 따른 3가지 Case에 대해서 오더 값 갱신비용을 측정한다. 기존방식에서 제안하는 방법과 본 논문에서 제안하는 방법에서 동일하게 사용되는 연산은 특정 위치에 노드를 삽입하거나 삭제하는 연산 비용으로 비교대상에서 제외하기로 한다. 그러나 4장에서 언급한 링크노드에 관한 삽입, 삭제 연산 비용은 포함된다. 기존방법과의 성능비교는 오더 값이 갱신되었을 때 탐색되는 노드의 개수와 갱신되는 노드의 개수를 4장에서 분류한 3가지 경우로 만들어 비교하였다.

각 Case는 실제 AXML[9] 등과 같은 동적인 문서에서 일어 날 수 있는 환경을 임의로 만든 것이다. AXML에서 갱신되는 형태는 문서의 여러 부분에서 갱신이 일어날 수 있으며 시간에 따라 변화하지만 특정 시간을 기준으로 볼 때 한 번에 한 위치에서만 삽입 혹은 삭제 연산이 일어난다고 볼 수 있다. 그래서 한 개의 노드가 삽입될 때마다 오더 값의 갱신이 일어난다고 가정을 했으며 실제 동적인 문서에 대한 질의 처리 시 비슷한 환경[12]이라고 볼 수 있다. 특히 Case3인 경우는 동일한 위치에 노드를 반복적으로 삽입하는 경우와 같다. 또한 노드의 삽입 순서는 XML문서의 파싱되는 순서와 반대로 이루어진다.

실험은 크게 2가지로 나눈다. 첫 번째는 삽입위치가 문서의 임의 장소에 삽입이 일어나는 경우이며 삽입형태는 단일 노드가 삽입되는 경우와 서브트리의 형태로 삽입되는 경우를 나누어서 실험을 하였다. 삽입형태를 분류한 이유는 단일노드일 경우보다 서브트리의 형태로 삽입하는 형태가 오더 값 갱신비용이 감소함을 보이기 위함이다. 실험 데이타는 Hamlet.xml를 사용했으며 특징은 노드의 수가 9000개 정도이며 같은 이름의 형제노드의 개수는 최대 11개까지 구성되어 있다. 트리 형태의 삽입 데이타는 같은 문서내의 노드 16개로 구성된 문서의 일부를 단일 노드 삽입과 같이 임의의 위치 50곳을 선정하여 삽입하였다. 단일 노드 삽입형태는 이 삽입 데이타를 단일노드 형태로 본 구조를 생각하지 않고 형제노드로만 삽입했다. 그림 8은 삽입되는 데이타의 예이다.

```
<SPEECH>
<SPEAKER>HAMLET</SPEAKER>
<LINE>Up from my cabin,</LINE>
<LINE>My sea-gown scarf'd about me, in the dark</LINE>
<LINE>Groped I to find out them: had my desire.</LINE>
<LINE>Finger'd their packet, and in fine withdrew</LINE>
<LINE>To mine own room again; making so bold,</LINE>
<LINE>My fears forgetting manners, to unseal</LINE>
<LINE>Their grand commission; where I found, Horatio,--</LINE>
<LINE>O royal knavery!--an exact command,</LINE>
<LINE>Larded with many several sorts of reasons</LINE>
<LINE>Importing Denmark's health and England's too,</LINE>
<LINE>With, ho! such bugs and goblins in my life,</LINE>
<LINE>That, on the supervise, no leisure bated,</LINE>
<LINE>No, not to stay the grinding of the axe,</LINE>
<LINE>My head should be struck off.</LINE>
</SPEECH>
```

그림 8 삽입 데이타의 예

그림 9는 실험결과를 보여준다. 각 Case 2, 3의 경우를 만들어서 삽입을 했다. 세로축은 제안방법과 기존방법의 재기록된 노드 수를 비율로 나타낸 것이며 특히 Case 3의 경우 삽입되는 형태가 제안방법의 서브트리 형태 삽입이 기존 방법과 단일 노드 삽입보다 적은 비용이 드는 이유는 링크노드의 삽입은 서브 트리 당 한 개 씩만 삽입이 되기 때문에 링크노드에 대한 운영비용이 다른 경우보다 훨씬 줄어들게 된다.

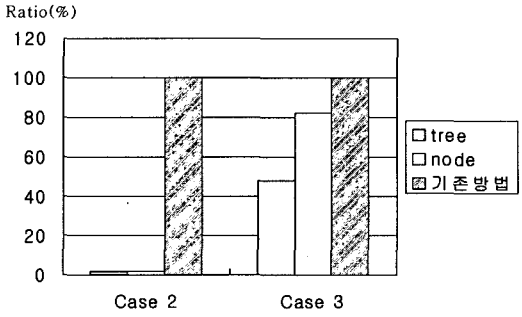

그림 9 삽입노드의 형태에 따른 비교

두 번째 실험에서는 노드의 삽입에 따른 오더 값 갱신비용 변화를 보이기 위해 삽입개수는 1에서 문서의 크기만큼 연속적으로 문서의 중간 위치(3012번째 노드)에 노드 삽입을 수행하였다. 이 실험은 특히 Case 3과 기존방법에 대한 비용을 알아보기 한 위치에 단일 노드를 삽입하는 극단적인 방법을 취했다. 실험 데이타는 이전 실험과 같다. 문서의 크기만큼 삽입노드의 수를 증가시키는 이유는 변화폭을 크게 하기 위함이다. 그림 10은 삽입 노드의 개수를 변화시키면서 오더 값을 갱신하기 위한 탐색범위를 참조된 노드의 수로 나타낸 것으로 기존의 방법보다 제안 방법이 훨씬 탐색 범위가 줄어드는 것을 알 수 있다. 그 이유는 기존방법은 삽입되는 횟수만큼 기존 문서를 탐색하나 제안방법 중 Case 3인 경우

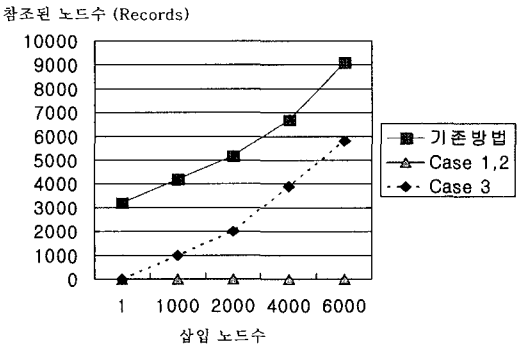

그림 10 삽입노드 개수 증가에 따른 탐색범위 변화

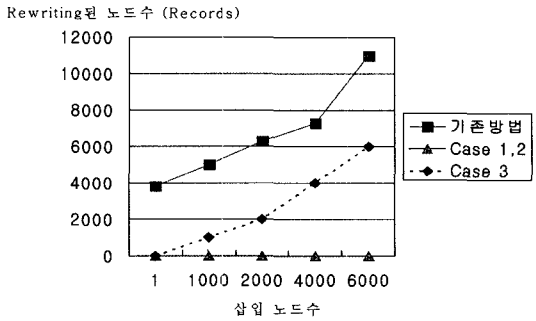

그림 11 삽입노드 개수 증가에 따른 Rewriting 노드 수

는 단지 삽입되는 노드의 개수에 비례하여 트리를 탐색을 하기 때문이다. Case 1,2인 경우는 거의 비용이 들지 않는 것을 보여준다.

그림 11은 오더 값을 갱신하기 위해서 재기록(rewriting)되는 노드수의 변화를 나타내주고 있는데 기존방법에서 SC 테이블의 각 필드 값 및 노드 정보를 저장하고 있는 노드 테이블의 갱신된 레코드 수를 계수했고 제안방법에서는 링크노드의 삽입개수 및 레이블이 갱신된 링크노드를 합하여 나타내었다. 기존기법은 마찬가지로 삽입되는 횟수와 기존 문서의 크기에 따라 갱신비용이 증가하고 제안기법은 Case 3의 경우 삽입되는 노드 수에 따라 링크노드의 수가 증가하기 때문에 이전 실험과 같이 삽입되는 노드의 개수가 커질수록 재기록 되는 노드의 수도 증가하는 것을 볼 수 있다.

6. 결론 및 향후 연구

대표적인 동적 XML 레이블링 기법은 프라임 넘버 레이블링 기법이며 이 기법은 노드간의 조상-후손 관계 결정을 위해 프라임 넘버의 특성을 이용한다. 이러한 특성을 이용하여 새로운 노드의 삽입 시 주변의 다른 노드의 레이블 값에 영향을 주지 않는 장점이 있으나 형제간의 순서정보를 위해서 오더 값을 갱신하는 비용은 문서의 크기에 비례하여 탐색 범위와 갱신되는 데이타의 수가 증가되는 단점을 지닌다. 제시한 방법은 이러한 오더 값의 갱신비용을 현저히 줄이기 위해서 오더 값 공유기법을 사용하였고 노드가 삽입되는 위치에 따라서 3가지로 분류하여 처리하였다. 제안기법은 기존의 방법보다 오더 값을 갱신하는 비용을 월등히 줄였다는 것을 실험을 통해서 보여주었다. 향후연구로는 제안기법을 가지고 기존의 질의처리 기법의 성능을 보다 향상시키는 방법이나 독자적인 질의처리 기법에 대한 연구 및 좀 더 다양한 동적 XML에 대한 질의처리에 더욱 적합한 방법에 대한 연구를 지속할 계획이다.

참 고 문 헌

[1] Tim Bray, Jean Paoli, C.M. Sperberg-McQueen, and Eve Maler, Extensible Markup Language (XML)1.0 (second edition), http://www.w3.org/TR/2000 /REC-xml-20001006, 2000.

[2] W3C Working Draft. XML Path Language(XPath) 2.0. http://www.w3.org/TR/2002/WD-xpath20-20021115, November 2002.

[3] D.Chamberlin et.al, XQuery 1.0: An XML Query Language, W3C Working Draft, 2001.

[4] Masatoshi Yoshikawa, Toshiyuki Amagasa, et al., XRel: A Path-Based Approach to Storage and Retrieval of XML Documents Using Relational Databases, ACM Transaction on Internet Technology, 2001.

[5] Quanzhong Li, Bongki Moon, Indexing and Querying XML Data for Regular Path Expressions, VLDB, 2001.

[6] Igor Tatarinov, Stratis D.Viglas, Chun Zhang, et al., Storing and Querying Ordered XML Using a Relational Database System,. Proceedings of SIGMOD 2002.

[7] The World Wide Web Consortium, http://www.w3.org/2002/ws/.

[8] http://www.inria.fr/

[9] http://activexml.net/

[10] Xiaodong Wu, Mong Li, Lee Wynne Hsu, A Prime Number Labeling Scheme for Dynamic Ordered XML Trees, ICDE, 2004.

[11] http://xml.apache.org/xerces2-j/

[12] Serge Abiteboul, Angela Bonifati, Gregory Cobena, et al., Dynamic XML Documents with Distribution and Replication, SIGMOD 2003

[13] http://www.oasis-open.org/cover/bosakShakespeare200.htm

유 지 열

서울대 컴퓨터공학부 석사

유 상 원

서울대 컴퓨터공학부 박사과정

김 형 주

1982년 서울대학교 전산학과(학사). 1985년 미국 텍사스 대학교 대학원 전산학(석사). 1988년 미국 텍사스 대학교 대학원 전산학(박사). 1988년 5월~1988년 9월 미국 텍사스 대학교 POST-DOC 1988년 9월~1990년 12월 미국 조지아 공과대학 조교수. 1991년~현재 서울대학교 컴퓨터공학부 교수