The Journal of the Convergence on Culture Technology (JCCT)
Vol. 9, No. 3, pp.835-843, May 31, 2023. pISSN 2384-0358, eISSN 2384-0366

http://dx.doi.org/10.17703/JCCT.2023.9.3.835
JCCT 2023-5-99

삭제된 노드의 재사용을 이용한 Fast XML 인코딩 기법

Fast XML Encoding Scheme Using Reuse of Deleted Nodes

고혜경*

Hye-Kyeong Ko*

요 약 XML 데이터의 구조를 고려할 때 경로 및 트리 패턴 매칭 알고리즘은 XML 질의 처리에 중요한 역할을 하고 있다. 노드 간의 결정 또는 관계를 용이하게 하기 위해 XML 트리의 노드는 일반적으로 두 노드 간의 조상-후손 관계를 신속하게 설정할 수 있는 방식으로 레이블링된다. 그러나 이러한 기법은 순서에 따른 업데이트로 삽입이 발생할 경우 기존 노드에 레이블을 다시 지정하거나 특정 값을 다시 계산해야 하는 단점이 발생한다. 따라서 현재 레이블링 기법들에서는 레이블을 업데이트 하는 비용이 매우 높다. 본 논문에서는 재레이블링 또는 재계산 없이 순서에 민감한 XML 문서의 업데이트를 지원하는 Fast XML 인코딩 기법이라는 새로운 레이블링을 제안한다. 또한 XML 트리의 동일한 위치에서 삭제된 레이블을 재사용하여 레이블의 길이를 제어한다. 제안한 재사용 알고리즘은 삭제된 모든 레이블을 동일한 위치에 삽입할 때 레이블의 길이를 줄일 수 있다. 실험 결과에서 제안된 기법은 순서에 민감한 질의 및 업데이트를 효율적으로 처리할 수 있다.

주요어 : 재사용, 삭제된 노드, 빠른 인코딩, 레이블링, 업데이트

Abstract Given the structure of XML data, path and tree pattern matching algorithms play an important role in XML query processing. To facilitate decisions or relationships between nodes, nodes in an XML tree are typically labeled in a way that can quickly establish an ancestor-descendant on relationship between two nodes. However, these techniques have the disadvantage of re-labeling existing nodes or recalculating certain values if insertion occurs due to sequential updates. Therefore, in current labeling techniques, the cost of updating labels is very high. In this paper, we propose a new labeling technique called Fast XML encoding, which supports the update of order-sensitive XML documents without re-labeling or recalculation. It also controls the length of the label by reusing deleted labels at the same location in the XML tree. The proposed reuse algorithm can reduce the length of the label when all deleted labels are inserted in the same location. The proposed technique in the experimental results can efficiently handle order-sensitive queries and updates.

Key words : Reuse, Deleted Node, Fast Encoding, Labeling, Update

*정회원, 성결대학교 컴퓨터공학과 조교수 (단독저자)
접수일: 2023년 4월 16일, 수정완료일: 2023년 4월 30일
게재확정일: 2023년 5월 5일

Received: April 16, 2023 / Revised: April 30, 2023
Accepted: May 5, 2023
*Corresponding Author: ellefgt@sungkyul.ac.kr
Dept. of Computer Engineering, Sungkyul Universkty, Korea

Ⅰ. 서 론

월드 와이드 웹에서 XML 저장소의 수가 증가함에 따라 XML 데이터를 효율적으로 저장하고 질의 할 수 있는 시스템이 개발되었다[1][2]. XPath 및 XQuery와 같은 질의 언어는 XML 데이터를 처리하도록 설계되었고 XML 데이터의 구조를 고려할 때 경로 및 트리 패턴 매칭 알고리즘은 XML 질의 처리에 중요한 역할을 하고 있다[3][4].

노드 간의 결정 또는 관계를 용이하게 하기 위해 XML 트리의 노드에는 일반. 으로 두 노드 간의 조상-후손 관계를 신속하게 설정할 수 있는 방식으로 레이블이 지정된다. 따라서 XML 질의를 효율적으로 처리하려면 우수하고 간결한 레이블링 체계가 중요하다[5][6]. 트리 모델을 사용하면 요소, 속성, 텍스트 데이터 등의 데이터 개체가 트리의 노드로 모델링되고 관계가 트리의 노드를 연결하는 에지로 모델링된다. XPath 및 XQuery는 XML 문서의 구조를 선형 경로 또는 가지 패턴으로 표현하는 두 가지 주요 XML 질의 언어로 정렬된 XML 트리를 보여준다. 예를 들어 XPath 질의: */book[/title]//section*[3]/은 *section*의 형제 노드인 모든 *section* 노드를 찾고 이러한 *section* 형제 노드는 *section*[3] 보다 앞에 있어야 한다. 한편 *section*[3]는 **book** ("//")의 후손이어야 하고 또한 *book*은 하위 *title* ("/")을 가지고 있다는 제한을 충족해야 한다. 질의가 선형 경로인지 잔가지 패턴인지에 관계없이 핵심 작업은 조상-후손, 부모-자녀, 형제 및 순서 관계를 효율적으로 결정하게 된다. 이러한 순서 관계를 쉽게 결정하기 위해, 여러 가지 레이블링 기법들이 제안되었다[1][7][8]. 레이블만을 기준으로 조상-후손 및 부모-자녀 관계를 신속하게 결정할 수 있다. XML이 정적인 경우 기존 레이블링 체계는 다른 질의를 효율적으로 처리할 수 있지만 XML이 동적으로 변하는 경우 레이블링 체계의 레이블을 효율적으로 업데이트하는 방법이 중요한 연구 주제가 된다. XML 트리의 요소는 본질적으로 순서가 지정되어 있으며, 이를 XML 문서에서 문서 순서라고 한다. XML에서 두 문단의 상대적 순서는 순서가 질의에 영향을 미칠 수 있기 때문에 XPath 및 XQuery에서는 기본적으로 질의 출력을 문서 순서로 해야 한다[5],[6]. 따라서 XML이 업데이트될 때 문서 순서를 유지하는 것이 매우 중요하다.

레이블링 체계에 대한 초기 연구는 일반적으로 구간 기반 레이블링 기법 (Interval based labeling scheme)[9][10]이다. XML 트리의 깊이 우선 탐색은 각 노드에 하위 노드의 레이블에 있는 값 범위를 포함하는 값 쌍을 할당하기 위해 수행된다. 그러나 웹의 XML 문서는 자주 변경되기 때문에 이러한 정적 구간 기반 레이블링 기법은 노드 삽입 및 삭제가 자주 발생할 때 전체 XML 트리의 레이블링을 다시 지정해야 한다. 접두사 기반 레이블링 기법 (Prefix-based labeling scheme) [11] 및 이진 레이블링 기법 [12]은 삽입된 리프 노드에 대해서는 레이블링을 다시 하지 않지만 두 형제 노드 간의 삽입에 대한 레이블의 변경을 피할 수 없다. 따라서, XML 업데이트에서 문서 순서를 유지하기 위해 [13], [14] 일부 연구가 수행되었지만 이러한 접근 방식의 업데이트 비용은 매우 높다. 최근에 제안된 OrdPath [14는 접두사 기반 레이블링과 유사한 방법으로 기존 트리에 삽입하기 위해 짝수 및 음수 정수를 재예약 한다는 점을 제외한 접두사 기법이다. OrdPath의 단점은 부모 노드와 자식 노드 사이에 새로운 노드의 삽입에 대한 레이블을 다시 지정할 수 없다. 다음으로 QED [15] 레이블링 방법은 노드를 삽입할 경우 레이블의 크기가 2비트 증가한다. 반대로, 우리가 제안한 레이블링 방식은 노드를 삽입하기 위해 레이블 크기가 1비트만 증가하는 장점이 있다.

본 논문에서는 이진 문자열의 사전 순서를 활용하는 빠른 XML 인코딩 기법 (FXE) 이라는 XML 문서에 대한 향상된 레이블링 기법을 제안하였다. FXE에서는 순서가 지정된 XPath 기능으로 동적 업데이트 및 질의를 지원한다. 또한 삭제된 레이블을 XML 트리의 동일한 위치에서 재사용하여 레이블의 크기를 제어할 수 있다. 또한, XML에 노드를 삽입할 때 레이블을 업데이트하는 방법에 대한 연구는 증가하고 있지만 삭제된 레이블을 처리하는 방법은 연구의 진행이 미비하기 때문에 본 연구는 삭제된 레이블을 재사용하는 방법을 연구한다. 본 논문에서 연구한 방법은 다음과 같은 장점을 갖는다.

1. FXE 기법은 기존 노드의 레이블을 다시 지정할 필요가 없으며 XML 트리에 순서에 민감한 노드를 삽입할 때 값을 다시 계산할 필요가 없다.
2. FXE 기법은 주문형 XPath 기능을 지원하고 순서가 지정된 XML을 업데이트하는 비용은 기존 기

법보다 훨씬 저렴하다.

3. FXE 기법은 XML 트리의 동일한 위치에서 삭제된 레이블을 재사용할 수 있다. 이러한 방식으로 XML 트리의 동일한 위치에 노드를 삭제하고 삽입할 경우 레이블의 크기를 제어할 수 있다.

본 논문의 구성은 다음과 같다. 2장은 관련 연구를 검토하고 3장은 제안된 레이블링 기법의 속성을 설명하고 삭제된 모든 레이블을 재사용하기 위한 알고리즘을 제안한다. 4장은 실험 결과를 보여주고 마지막으로 5장은 결론을 나타낸다.

II. 관련연구

1. 접두사 레이블링 기법

접두사 레이블링 기법 (Prefix labeling scheme)에서 노드의 레이블은 상위 레이블이 자체 레이블을 연결하는 것으로 두 노드 u 및 v에 대해 u는 레이블(u)이 레이블(v)의 접두사인 경우 v의 조상이다. 레이블(v) 왼쪽에서 레이블(u)을 제거할 때 레이블(v)에 접두사가 없는 경우 노드 u가 노드 v의 상위 레벨이 된다. DeweyID [11]는 노드의 n번째 자식 노드에 정수 n을 레이블로 지정하고, 이를 부모 레이블 및 구분 기호(".")와 연결하여 이 자식 노드의 전체 레이블을 구성한다. 노드가 삽입되면 DeweyID 접두사 기법은 문서 순서를 유지하기 위해 삽입된 노드와 형제 노드의 하위 노드 뒤에 형제 노드의 레이블을 다시 지정해야 한다. OrdPath [15]는 초기 레이블링에서 홀수만 사용했고 XML 트리를 업데이트할 때 홀수 두 개 사이의 짝수를 사용하여 다른 홀수를 연결하였다. OrdPath는 전체 숫자의 절반을 낭비하게 되고 질의 시 성능이 Dewey의 질의 성능보다 나쁘다. 또한, OrdPath는 짝수 및 홀수 숫자를 기준으로 접두사 수준을 결정하는 데 더 많은 시간이 필요하다.

2. 소수 레이블링 기법

소수 레이블링 기법 (Prime number labeling scheme)은 루트 노드는 *1* (정수)로 레이블을 지정한다. 다음으로, 하향식 접근 방식을 기반으로 각 노드에는 고유한 소수가 부여되며 각 노드의 레이블은 상위 노드의 레이블과 자체 레이블의 곱으로 나타낸다. 문서 순서가 변경될 때 소수 레이블링 체계는 레이블링을 다시 하는 대신 SC 값만 다시 계산하면 되지만, 재계산에는 훨씬 더 많은 시간이 소요된다. 예를 들어, 첫 번째 노드 앞에 새 형제 노드를 삽입하면 다음에 사용할 수 있는 소수가 23이면 새 노드의 레이블은 23(1 * 23)이다. 이제 이 새로 삽입된 노드가 첫 번째 노드가 되고, 이 삽입된 노드 이후의 노드 순서는 모두 1과 함께 추가되어야 한다. 소수 레이블링 체계는 28364406 모드 23 = 1, 28364406 모드 2 = 2, 28364406 모드 3 = 3, ..., 28364406 모드 17 = 8, 28364406 모드 19 = 9와 같은 28364406 모드에 대한 새 SC 값을 계산한다.

3. 연구 동기

소수 레이블링 기법은 기존 노드에 레이블링을 다시 지정하지 않고 순서에 따라 업데이트를 지원하지만 노드의 새 순서를 기준으로 SC 값을 다시 계산해야 한다. 소수 레이블링 기법은 단일 SC 값이 아닌 여러 SC 값을 사용하여 SC 값이 매우 큰 숫자가 되는 것을 방지한다. 또한 소수 레이블링 기법은 소수를 얻기 위해 많은 정수를 건너뛰고, 자식 레이블링은 다음에 사용 가능한 소수와 부모 레이블링의 곱이며, 둘 다 소수 레이블링 체계 레이블링을 위한 저장 공간을 크게 만든다 [16]. 따라서 재계산에는 많은 시간이 소요된다. OrdPath [14]는 어느 정도 동적이지만 업데이트를 처리하려면 코드를 디코딩하고 덧셈 및 나눗셈 연산을 사용하여 두 홀수 사이의 짝수를 계산해야 하므로 업데이트 비용이 그렇게 저렴하지 않다. 또한 짝수 값이 많이 낭비되어 레이블 크기가 커지며 XML 질의 처리에서 짝수 및 홀수 값을 기준으로 접두사 수준을 결정하는 데 더 많은 시간이 필요하다.

III. Fast XML 인코딩 기법

본 논문의 주요 목표는 순서에 민감한 노드를 삽입할 때 업데이트 비용을 크게 줄이고 기존 노드의 레이블을 다시 지정하는 것을 완전히 피하는 것이다. 또한 XML 트리의 동일한 위치에서 노드를 삭제하고 삽입할 때 레이블 길이를 제어하기 위한 알고리즘을 제안한다. 이 장에서는 이진 문자열 기반 접두사 기법인 Fast XML Encoding 기법 (FXE)에 대해 자세히 설명한다. FXE은 재라벨링 또는 재계산 없이 라벨 삽입을 지원한

다. 먼저, 우리는 FXE을 기반으로 노드에 레이블을 지정하는 방법을 설명하기 위해 예제를 사용하고 그런 다음 FXE공식 레이블링 알고리즘을 설명한다. 또한 다양한 레이블링 기법의 크기 요구 사항을 분석한다.

1. 비트 표현

접두사 기법의 비트 표현은 레이블 비교를 위해 문자열을 정수로 변환하여 오버헤드를 줄이기 위해 고안되었다. 레이블링에 정수를 사용하기 위해 접두사 기법에서는 레이블링에 대한 점선 십진수 문자열(예: *"2.1.3"*)로 표시된다. 또한, 구분 기호 "."를 구현의 숫자와 함께 저장하여 서로 다른 구성 요소를 분리할 수 없다. 따라서 각 점(예: *"2"*, *"1"* 및 *"3"*)으로 구분된 구성 요소 값은 루트에서 노드로 표시되는 경로 아래의 연속적인 수준에서 "정수 값"의 맥락에서 노드 관계를 반영한다. 이런 이유로 레이블 비교를 위해 문자열을 정수로 변환할 때 오버헤드가 발생하게 된다. 본 논문은 이러한 오버헤드를 극복하기 위해 논문에서 제안된 레이블링 기법의 비트 표현을 고려하였다. 구분 기호 "."는 비트 시퀀스의 값으로 특징지었다. 비트 표현에서 문자열 값은 점선이 있는 10진수 문자열이 아니라 압축된 이진 표현이다. 인코딩 단위의 길이가 $m = log2\ (k + 1)$에 의해 결정되는 k 기반 표현을 사용하였다. 이 아이디어는 하나의 m-bit 코드를 예약하는 것으로 구분 기호 "."를 나타내며, m비트 코드 시퀀스는 기본 k를 가진 숫자로 해석된다. 이진 표현의 경우 다음 접근 방식을 사용하여 구분 기호를 처리한다. 구분 기호 "11"을 구분 기호로 사용하여 레이블의 다른 구성 요소를 구분합니다. 예를 들어, $k = 3$은 00: *"0"*, 01: *"1"*, 10: *"2"*, 11: "를 전달한다. 따라서 *"2.1.3"*은 $2 * 30.1 * 30.1 * 31 + 0 * 30$으로 표시되는 *10110100*에 의해 인코딩된다.

2. 순서가 있는 XML 트리

XML의 요소는 기본적으로 순서를 가지고 있는데 책 XML 문서를 예제로 사용하였다. XML 문서에 3개의 챕터 태그가 있고 이 챕터 태그의 발생 순서를 살펴보면 XML 노드의 시작 요소 바로 뒤에 발생하는 첫 장 태그가 책의 첫 장에 대한 자세한 내용을 제공한다고 추론할 수 있다. 첫 번째 장 태그 뒤에 나타나는 다음 장 태그는 책의 두 번째 장에 대한 세부 정보를 나타낸다. 그림 1은 순서가 있는 XML 트리의 예를 보여준다. XML의 순서는 사용자가 순서에 민감한 질의를 발행하는 데 관심이 있을 수 있으므로 중요하다. 예를 들어, 질의서 [*author*[2] = *"Lee"*]는 두 번째 *author*가 "Lee"인 책의 목록을 검색한다.

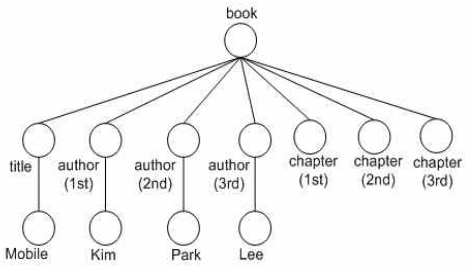

그림 1. 순서가 있는 XML 트리의 예
Figure 1. Example of an ordered XML tree.

이 작업은 XML 내에서 순서를 유지해야 하므로 많은 XML 레이블링 기법에 문제가 발생한다. 예를 들어 그림 1의 XML 트리에 새 작성자를 두 번째 작성자로 삽입해야 하는 경우 *"Park"*와 *"Lee"*를 각각 세 번째 및 네 번째 형제 위치로 밀어 넣어야 한다.

3. Fast XML 인코딩 기법

다음은 XML 문서에 대한 업데이트를 일반적으로 사용되는 트리 작업, 즉 요소, 텍스트 또는 속성을 삽입하고 요소, 텍스트 또는 속성을 삭제하는 방법으로 설명한다. 실제로 노드를 삭제해도 XML 트리의 노드 순서에는 영향을 주지 않는다. 그러나 INSERT 작업은 DELETE 작업보다 약간 더 복잡하다. 따라서 다음 세 가지 순서에 민감한 삽입 사례에 대해 설명한다.

정의 1. (사전순서 <) 두 이진 문자열 *N-left*와 *N-right*가 주어졌을 때, *N-left*는 정확히 동일하다면 사전 순서적으로 *N-right*와 같다고 한다. *N-left*는 사전적으로 *N-right*보다 작다고 한다.

1. *N-left*와 *N-right*의 사전 순서적 비교는 왼쪽에서 오른쪽으로 조금씩 있습니다. 만약 *N-left*의 전류 비트가 *0*이고 *N-right*의 전류 비트가 1이라면, *N-left* < *N-right*이고 비교가 중지된다.(*0* < *1*)
2. *len*(*N-left*) > *len*(*N-right*) 및 *N-right*는 *N-left*의 접두사이며, *N-left*의 접두사 문자열을 제외한 나머지 비트가 *0*이면 *N-left* < *N-right*가 되고 비교가

중지된다.

속성 1. 2개의 이진 문자열 *110*과 *11*이 주어졌을 때, 비교가 왼쪽에서 오른쪽으로 되어 있기 때문에 *110* < *11*은 사전 순서적으로, *110*의 두 번째 비트는 *1*이다.

FXE의 가장 중요한 특징은 숫자 순서가 아닌 사전 순서를 기반으로 레이블을 비교한다는 것이다. 예를 들어, 두 개의 이진 문자열 *10*과 *1*이 주어지면, *10* < *1*이 사전법적으로 비교가 가능하기 때문이다.

정의 2. (노드 *N*의 레이블) *N*의 첫 번째 자식은 레이블(*N*).*1*, *N*의 두 번째 자식은 레이블(*N*).*11*, *i*번째 자식은 레이블(*N*).*1i*("."는 구분 기호)로 레이블이 지정된다.

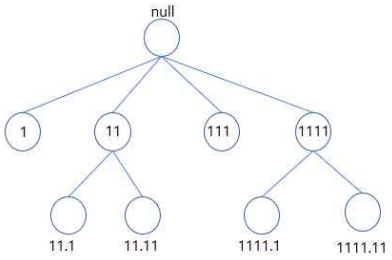

그림 2. FXE 기법의 레이블링 표현
Figure 2. Labeling representation of FXE scheme.

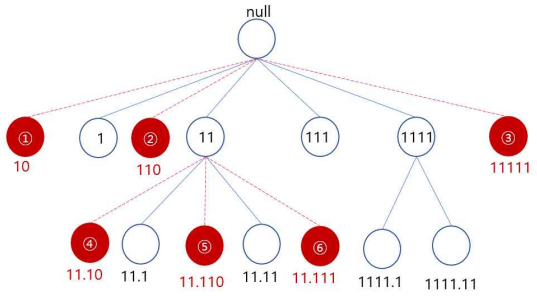

그림 3. FXE 기법의 업데이트 동작과정
Figure 3. Update Operation of FXE scheme.

그림 2와 3은 FXE의 레이블링 방법 및 업데이트 동작의 예를 나타낸다. FXE에서 우리는 먼저 XML 트리를 깊이 우선 탐색하고 각 노드에 레이블을 할당한다. 루트 노드에는 빈 문자열이 레이블이 지정된다. 다음으로, 루트의 4개 하위 노드에 레이블이 지정된다. 4개의 자식 노드의 *prefix-label*은 모두 빈 문자열이므로 *self-label*은 정확히 4개의 자식 노드에 대한 완전한 레이블이다. 첫 번째 자식 노드의 *self-label*은 *1*, 두 번째 자식 노드는 *11*, 세 번째 자식 노드는 *111*, 네 번째 노드는 *1111*이다. 그림 3에서, 노드의 레이블은 부모의 레이블과 자신의 레이블을 연결한다. 두 레이블 **11**과 *11.10*의 경우, *11*가 *11.10*의 접두사라면 *11*은 *11.10*의 조상이다.

정의 3. (문자열 순서 삽입) 기존의 두 인접 레이블인 *N-left* 및 *N-right*에 대해 다음의 삽입 문자열 순서로 현재 레이블을 수정하지 않고 항상 새 레이블 *N-new*를 할당할 수 있다. 여기서 *len*(*N*)은 비트 길이로 정의된다: (+는 문자열 연결자를 나타낸다):

1. 맨 왼쪽 형제 노드 앞에 노드를 삽입할 경우.
   N-new = *N-firstsibling* + *0*.
2. 두 노드 사이에 노드를 삽입할 경우.
   len(*N-left*)≤*len*(*N-right*)이면 *N-new* = *N-right* + *0*이다.
   len(*N-left*) > *len*(*N-right*)인 경우 *N-new* = *N-left* + *1*이다.
3. 맨 오른쪽 형제 노드 뒤에 노드를 삽입한다,
   N-new = *N-lastsibling* +*1*.
   레이블 된 노드들은 *1, 11, 111, 1111* 형태의 비트를 갖는 노드로 구성이 되고 업데이트된 노드들은 노드의 *self-label*이 마지막 오른쪽에 삽입된 노드를 제외하고 마지막 비트가 *0*인 값을 갖게 된다.

그림 3에서 노드 ①를 삽입할 때, ①의 *self-label*은 *10*(*10* ≤ *110*)이다. 노드 ②를 삽입할 때, ②의 왼쪽 *self-label*은 길이가 *1*인 '*1*'이고 ②의 오른쪽 *self-label*은 길이가 *2*인 '*11*'이므로, 우리는 오른쪽 *self-label*(*11*+*0*⇒ *110*) 후에 하나의 *0*을 직접 연결한다. 노드 ③을 삽입할 때 ③의 *self-label*은 *11111*(*1111*+*1* ⇒ *11111*)이다.

4. 삭제된 라벨의 재사용

FXE 기법의 경우 비트를 연결하여 레이블의 업데이트를 처리하는 방법으로 레이블 크기의 증가하게 된다. 증가하는 레이블이 크기를 줄이는 방법으로 XML에 노드가 삽입될 때 노드 재레이블을 피할 수 있도록 기존

에 삭제된 레이블을 재사용 할 수 있는 방법을 이용한다. 이 방법은 삭제된 모든 레이블을 재사용하여 레이블의 크기가 빠르게 증가하는 것을 제어할 수 있다. 제안된 알고리즘에 노드를 삽입하고 XML 트리의 동일한 위치에서 재사용되는 모든 삭제된 레이블의 경우 레이블 크기가 1비트씩 증가하여 기존에 2비트씩 증가하는 기법들에 비해 레이블의 크기를 줄일 수 있다.

정의 4. (삭제된 레이블의 재사용) 기존의 두 인접 레이블인 *N-left* 및 *N-right*에 대해 삭제된 레이블 *N-delete*는 인접 레이블의 길이보다 작은 길이의 *N-delete*인 경우 동일한 위치에서 삭제된 레이블을 수정하지 않고 항상 두 레이블 간에 재사용될 수 있다. 여기서 *Same_bit*는 *N-left*와 *N-right* 사이의 동일한 비트로 정의된다. (+는 연결자를 나타낸다)

1. *len(N-left)* ≤ *len(N-right)*이면
   N-new = Same_bit,
2. *len(N-left) = len(N-right)*이면 *N-new = N-left+ 1*

인접 레이블보다 길이가 큰 삭제된 레이블은 재사용되지만 길이가 작은 삭제된 레이블은 재사용되지 않는다. 전체 레이블 크기(즉, 공간)를 줄이고 질의 성능을 개선하면 삭제된 모든 레이블을 재사용해야 하므로 레이블 길이가 늘어나지 않는다. 삭제가 없는 경우 FXE 기법의 비용은 매우 저렴하며 인접 레이블의 한 비트만 연결하면 된다. 그러나 삭제가 있는 경우 삽입된 레이블이 가장 작은 길이라고 보장할 수 없다. 먼저, 우리는 삭제가 있는 경우 FXE 기법의 삽입된 이진 문자열의 길이를 가장 작게 보장할 수 없는 이유를 보여주기 위해 예제를 설명한다. 예를 들어, *11*과 *10* 사이의 레이블 1을 삭제하고(그림 3 참조) 같은 위치에 다른 라벨을 삽입하고자 할 때, 삽입된 코드는 *110*이 된다. 삭제된 레이블 *1*은 다음보다 길이가 작기 때문에 재사용되지 않는다. 이웃 레이블 (*10* 및 *11*) 및 재 삽입된 레이블 *110*은 삭제된 레이블 *1*보다 길이가 더 크므로 길이가 빠르게 증가한다. 반면에, 우리가 *10*과 *11* 사이의 레이블 *110*을 삭제하고, *1*과 같은 위치에 다른 레이블을 삽입해도 삽입된 레이블은 여전히 *110*이다. 삭제된 레이블 *110*은 인접 레이블(*10* 및 *11*)보다 길이가 길기 때문에 재사용 될 수 있다. 알고리즘 1은 주어진 두 레이블 사이에서 가능한 가장 작은 레이블을 사전 분석적으로 찾는 방법을 나타낸다. 입력은 왼쪽 레이블 *N-left* 및 오른쪽 레이블 *N-right*이고 출력은 재사용되는 레이블 *N-insert*이며 XML 트리의 동일한 위치에서 삭제된 레이블이다.

알고리즘 1. 삭제된 레이블의 재사용 알고리즘
Algorithm 1. Reuse algorithm of deleted labels

Algorithm 3 Assign *the smallest length* label
Input N_{left}, N_{right}
Output N_{insert}
begin
01: **if** ($len(N_{left}) < len(N_{right})$)
02: **if** ($leftSame_{bit}$ is equal to $right\ Same_{bit}$)
03: N_{insert} = getSubString($Same_{bit}$, s, $s-p$)
04: **else if** ($len(N_{left}) = len(N_{right})$)
05: **if** ($leftSame_{bit}$ is equal to $rightSame_{bit}$)
06: N_{insert} = getSubString($Same_{bit}$,,s, $s-p$)
07: **else** ($len(N_{left}) > len(N_{right})$)
08: $N_{insert} = N_{left} + 1$;
09: **end if**
10: **return** N_{insert}
end

알고리즘 1에서 *S*는 *N-left*와 *N-right* 사이의 동일한 비트를 나타낸다. *P*는 *N-left* 및 *N-right*의 첫 번째 비트의 위치를 기록하고 *getSubString*은 왼쪽에서 오른쪽으로 *N-left*와 *N-right*를 비트 단위로 비교하여 지정된 두 개의 동일한 비트 레이블을 추출한다. *getSubString(Nleft, S, S-P)*은 *getSubString(Nright, S, S-P)*과 동일하다. 우리는 알고리즘 1에 의해 *getSubString(Nleft, S, S-P)*을 왼쪽 *Samebit*로 표시하고 *getSubString(Nright, S, S-P)*을 오른쪽 *Samebit*로 표시한다. 예를 들면, *10*과 *11*사이의 레이블 *1*이 삭제되고 *10*과 *11*사이의 레이블을 새로 삽입해야 한다고 가정하자. 새로 삽입된 레이블은 *110*이다. 삭제된 레이블 1은 인접 레이블보다 길이가 작기 때문에 재사용되지 않는다. 반면에 *10*과 *11*을 왼쪽에서 오른쪽으로 조금씩 비교하여 *getSubString(Nleft, S, S-P)* = *getSubString(100, 1, 2)* = *1* 및 *getSubString(Nright, S, S-P)* = *getSubString(110, 1, 2)* = *1*. 정의 4에 따라 *N*을 삽입하면 *1*이다. 그 결과, 삭제된 레이블 *1*은 재사용된다. 즉, 길이가 작은 삭제된 레이블이 먼저 재사용 되어 알고리즘 1을 기반으로 레이블 크기를 제어할 수 있다.

5. 레이블 사이즈 분석

레이블의 사이즈를 분석하기 위해서 구간 기반 기법, DeweyID 접두사 기법, 소수 레이블링 기법과 FXE 기법의 크기를 분석하였다. 여기에서 *"D"*, *"F"*, *"N"*은 최대 깊이, 최대 팬 출력 및 번호를 나타내는 데 사용된다. XML 트리의 노드 중 하나로 이 문서의 크기는 비트로 나타낸다. 구간 기반 기법 [13]에서 각 노드에는 간격의 시작과 지점을 나타내는 두 개의 번호가 할당된다. 이러한 숫자가 사용할 수 있는 최대값은 *N*으로 여기서 N은 XML 트리의 노드 수이다. 즉, 간격 기반 레이블링 체계에 대한 레이블의 최대 크기는 $N * 2(1 + log(F))$ 비트입니다. DeweyID 접두사 기법은 self_label을 저장할 수 있 최대 크기는 $log(F)$이다. 최대 깊이가 D이고 접두사_label에 최대 $(D - 1)$ 구분 기호가 있으므로 전체 레이블(prefix_label L self_label)을 저장할 최대 크기는 $D * log(F)$이다. 따라서 DewID가 요구하는 최대 크기 XML 트리의 모든 노드를 저장하는 ID 접두사 체계는 $N * D * log(F)$이다. 소수 레이블링 기법 [1]에서 XML 트리의 모든 노드를 저장하는 데 필요한 최대 크기는 $N * D * log(N * log(N))$이다.

FXE에서 크기를 저장하기 위해 필요한 비트는 $log(log(2N))$이다. $2N$과 $2N + 1$ 사이에는 상수만 다르므로 $2N + 1 = 2n + 1 - 1$로 가정한다. 따라서 XML 트리의 모든 노드를 저장하는 실제 총 레이블 크기는 $log(log(2N)) + 2Nlog(N + 1) + 2log(N + 1) + 1$이다. FXE에서 형제 노드의 레이블은 *1, 11, 111*이며 XML 트리의 모든 노드를 $N * D * 4log(F)$로 저장한다. Dewey ID 및 소수 레이블링 기법과 비교하여 F는 확실히 $N * log(N)$보다 작으므로 DeweyID는 최악의 경우를 고려할 때 소수 레이블링 체계보다 작은 레이블 크기를 필요로 한다. 이것은 소수 레이블링 체계가 소수를 얻기 위해 많은 정수를 건너뛰고 두 숫자의 곱을 사용하기 때문이다.

Ⅳ. 성능 분석

본 논문에서는 제안한 FXE 기법의 성능을 4가지 레이블링 기법과 비교하였다. 4가지 라벨링 체계는 Java에서 구현되었다. 구간 기반 기법 (Interval) [16], DeweyID 접두사 기법(Dewey) [11], 소수 레이블링 기법(Prime) [1] 및 제안한 FXE 기법이 이용되었다.

1. 실험 환경

실험은 Windows 10을 실행하는 16GB RAM에서 수행되었다. XML 데이터 세트는 Oracle에 저장되었고 데이터베이스에는 레이블 지정 체계에 따라 각 요소 이름, 레이블 및 상위 레이블이 저장되었다. 네 가지 레이블링 체계의 성능을 평가하고 비교하기 위해 저장 및 업데이트 부분을 실험하였다. 데이터 셋은 파일 크기, 팬 아웃, 깊이, 총 노드 수 등 특성이 서로 다르기 때문에 선택하였고 간격(예: ">", "<") 및 프라임(예: *"mod"*, ">", "<", "=")에서 사용되는 연산은 데이터베이스 시스템에서 직접 지원된다. 데이터 셋의 특성은 표 1에 나타난다.

표 1. 데이터 셋의 특성
Table 1. Characteristics of Datasets

Dataset	Topic	# of nodes
D1	Sigmod record	36
D2	Club	342
D3	Actor	1219
D4	Department	2568
D5	NASA	4562
D6	Shakespeare plays	6521

2. 공간 요구사항

이 실험에서, 우리는 4가지 레이블링 기법에 대한 공간 요구 사항을 실험하였다. 고정 길이 레이블의 크기를 비교하였고 레이블의 길이는 데이터 셋의 레이블의 최대 길이에 의해 결정된다. 그림 4는 6개 데이터 셋에 대한 전체 레이블의 사이즈를 나타낸다.

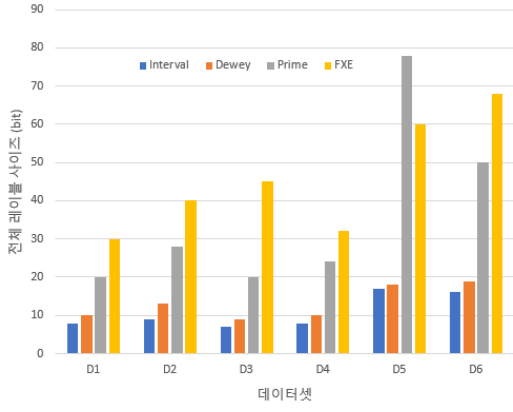

그림 4. 데이터셋의 공간
Figure 4. Space for dataset

예상대로, Interval의 최대 레이블의 크기는 Dewey,

Prime 및 IBSL의 것에 비해 작습니다. Prime은 소수를 얻기 위해 많은 숫자를 건너뛰고 상위라벨의 제품과 노드의 라벨로 사용하기 때문에 라벨 크기가 크다. FXE는 상위 노드의 레이블을 하위 노드의 레이블에 접두사로 추가했기 때문에 6개의 데이터 세트 각각에 대한 레이블 크기가 크다. 데이터 셋 *D3*는 배우를 위한 영화 목록을 포함하고 있으며 엄청난 팬 아웃을 가지고 있다. 결과적으로, 데이터 셋 *D5*는 팬 아웃이 적고 깊이가 높은 NASA 문서이다. 이 구조는 FXE에 이상적인 문제이다. Interval, Prime 및 FXE에 대한 6개 데이터 세트의 총 크기는 Dewey의 *0.76, 2.64, 3.2* 및 *3.9*배이다.

3. 삭제된 레이블의 재사용 성능

우리는 알고리즘 1을 제안한 레이블링 기법을 삭제된 레이블의 재사용이 없는 경우와 비교하였다. 데이터 셋 *D6*의 햄릿의 홀수 위치에서 노드가 삭제 및 삽입되는 경우를 테스트하였다. 삭제 및 삽입 후 새로운 햄릿 파일인 햄릿 2를 호출하였다. 둘째, 햄릿 2의 홀수 위치, 세 번째 위치, 햄릿 3의 홀수 위치, 햄릿 4의 네 번째 홀수 위치 등에서 노드가 삭제되고 삽입되는 것을 테스트하였다. 제안된 기법을 기반으로 *10,000*개의 레이블을 생성하였다. 그런 다음 레이블이 삭제된 다음 짝수 위치에 삽입되는 것을 테스트하였다. 삭제된 레이블의 재사용을 하지 않는 경우(proposed scheme)와 재사용(알고리즘 1기반)의 성능을 비교하였다. 그림 5는 두 기법의 레이블 사이즈를 나타낸다.

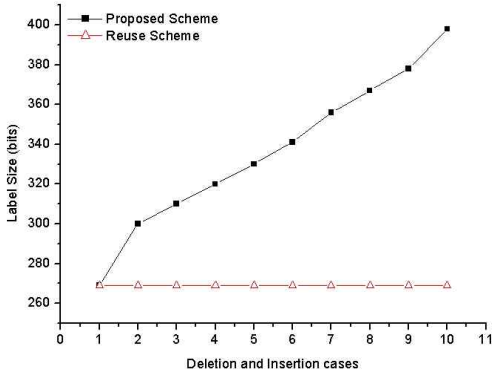

그림 5. 질의 응답 시간
Figure 5. Query response time

그림 5는 삭제된 모든 라벨이 동일한 위치에서 재사용되기 때문에 10건 모두에서 재사용의 레이블 크기가 증가하지 않음을 보여준다. 반면, 재사용을 하지 않는 경우 레이블 크기는 10개 사례에 대해 선형적으로 증가하였다. 그 결과 알고리즘 1은 삭제된 모든 레이블을 동일한 위치에서 재사용할 수 있음을 확인하여 레이블 크기 증가를 효율적으로 제어할 수 있음을 알 수 있다.

V. 결 론

정렬된 XML 트리에서 질의 및 업데이트를 효율적으로 지원하기 위해서 새로운 이진 문자열 기반 레이블링 체계가 개발되었다. FXE 기법은 이전에 할당된 레이블을 수정하지 않고 새 레이블을 여러 개 할당할 수 있으며, XML 트리에 순서에 민감한 노드를 삽입할 때 기존 노드의 레이블을 다시 지정하거나 값을 다시 계산할 필요가 없다.

또한 삭제된 모든 레이블을 재사용할 수 있도록 새로운 알고리즘을 제안하였다. 실험 결과 제안된 기법이 들어온 질의를 효율적으로 처리하고, 낮은 업데이트 비용을 유지하며, 많은 노드가 삽입 및 삭제될 때 레이블 크기를 줄일 수 있음을 나타냈다.

향후 연구로는 레이블 크기를 더 효율적으로 줄이고 접두사 기법의 구분자를 처리하는 방법에 대해 추가로 연구할 것이다. 또한, XPath의 모든 구조적 관계를 지원하기 위해 전체 XPath 축을 고려한 질의 처리 방법을 연구할 예정이다.

References

[1] X. Wu, M.-L. Lee, W. Hsu, “A prime number labeling scheme for dynamic ordered XML trees,” In proc. of International Conference on Data Engineering, pp. 66-78, 2004. DOI: 10.1109/ICDE.2004.1319985.

[2] J-H. Choi and M. Koo, “A study on the offering of the latest film information using XML Parser,” The Journal of the Convergence on Culture Technology (JCCT), vol. 3, no.1, pp. 19-23, 2017. DOI: 10.17703/JCCT.2017.3.1.19.

[3] H. Ko and M. Yang, “An effective XML schema conversion technique for improving XML document reusability using pattern list,” International Journal of Internet, Broadcasting and Communication (JIIBC), vol. 9, no.2,

pp.11-19, 2017. DOI: 10.7236/IJIBC.2017.9.2.11.
[4] Y. Jung, M. Kang, G. Cha, and S. Kwang, "Open Trade Technical Model using ebXML for FTA with China and Korea," International Journal of Advanced Culture Technology (IJACT) vol. 2, no. 1, pp. 25-29, 2014. DOI: 10.17703/IJACT.2014.2.1.025.
[5] A. Berglund, S.Boag, D.Chamberlin, M. F. Fernández, M. Kay, J. Robie, J. Simeon, XML Path Language (XPath) 2.0, W3C working draft, 2005.
[6] S. Boag, D. Chamberlin, M. F. Fernández, D. Florescu, J. Robie, J. Simeon, XQuery 1.0, W3C working draft, 2005.
[7] C. Zhang, J. F. Naughton, D. J. DeWitt, Q. Luo, G. M. Lohman, "On supporting containment queries relational database management systems," In proc. of International ACM SIGMOD Conference on Management of data, pp. 425-436, 2001. DOI: 10.1145/375663.375722.
[8] J. McHugh, S. Abiteboul, R. Goldman, D. Quass, J. Widom, "Lore: A database management system for semistructured data, ACM SIGMOD Record, v ol. 2, no. 3, pp. 54-66, 1997. DOI: 10.1145/262762.26 2770.
[9] Q. Li, B. Moon, "Indexing and querying XML data for regular path expressions," In proc. of International VLDB Conference, pp. 361-370, 2001.
[10]M. Yoshikawa, T. Amagasa, "XRel: A path-based approach to storage and retrieval of XML documents using relational databases," ACM Transaction Internet Technology, vol. 1, no. 1, pp. 110-141, 2001. DOI: 10.1145/383034.383038.
[11]Tatarinov, S. D. Viglas, K. Beyer, J. Shanmugasundaram, E. Shekita, C. Zhang, "Storing and querying ordered XML using a relational database system," In proc. of International ACM SIGMOD Conference Management of Data, pp. 204-215, 2002. DOI: 10. 1145/564691.564715.
[12]E. Cohen, H. Kaplan, T. Milo, Labeling dynamic XML trees, In proc. of ACM SIGMOD-SIGACT-SIGART symposium on Principles of Database Systems, pp. 271-281, 2002. DOI: 10.1145/543613.543648.
[13]C. Li, T. W. Ling, M. Hu, "Efficient processing of updates in dynamic XML data," In proc. of International Conference on Data Engineering, pp. 13-22, 2006. DOI: 10.1109/icde.2006.58.
[14]P. O'Neil, E. O'Neil, S. Pal, I. Cseri, G. Schaller, "Ordpaths: Insert-friendly XML node labels," In proc. of Iternational ACM SIGMOD Conf., pp. 903-908, 2004. DOI: 10.1145/1007568.1007686.
[15]C. Li, T. W. Ling, M. Hu, "Reuse or never reuse the deleted labels in XML query processing based on labeling schemes," In proc. of International Conference on Database Systems for Advanced Applications, pp. 659-673, 2006. DOI: 10.1007/11733836_46.
[16]S. Abiteboul, H. Kaplan, T. Milo, "Compact labeling schemes for ancestor queries," In proc. of 12th ACM-SIAM Symporsium on Discrete Algorithms, pp. 547-556, 2001. DOI: 10.1137/s00975 39703437211.

※ 이 논문은 2023년도 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임 (No.NRF-2021R1A2C1012827)