대용량 XML 문서에서 효율적인 갱신을 위한 비트-패턴 기반의 XML 레이블링 기법

(XML Labeling Scheme based on Bit-Pattern for Efficient Updates of Large Volume of XML Documents)

서 동 민 † 박 용 훈 ††
(DongMin Seo) (YongHun Park)
임 종 태 †† 김 명 호 †††
(JongTae Lim) (MyoungHo Kim)
유 재 수 ††††
(JaeSoo Yoo)

요 약 기존에 제안된 XML 레이블링 기법은 XML 문서 변경 시, 문서 내 노드들의 구조 관계를 정확하게 유지하기 위해 재레이블링을 수행하거나 한 노드의 레이블이 많은 정보를 표현할 수 있는 레이블링 기법을 사용한다. 하지만, 대용량 XML 문서 내에서의 재레이블링은 많은 비용이 요구되고, 많은 정보를 표현할 수 있는 레이블링 기법은 많은 저장 공간이 요구돼 질의 처리 성능이 저하되는 문제를 야기한다. 그래서 본 논문에서는 재레이블링을 피하고 질의 처리 성능을 향상시키기 위해 최적화된 저장 공간을 사용하는 비트-패턴 기반의 레이블링 기법을 제안한다. 제안하는 비트-패턴 기반의 레이블링 기법은 노드들의 구조 관계를 하나의 비트열에 표현함으로써 기존에 제안된 레이블링 기법들에 비해 우수한 성능을 가진다.

키워드 : XML 레이블링, 동적 XML, 비트-패턴

Abstract When an XML document is updated in order to represent correctly the structural relationships of nodes in a document, the existing XML labeling schemes relabel nodes or use a labeling scheme that the label of a node has much information. However, the relabeling on large XML documents needs many labeling costs and the labeling scheme that the label of a node has much information requires many storage costs. Therefore, the existing labeling schemes degrade significantly query processing performance on dynamic XML documents. This paper proposes the bit-pattern labeling scheme that solves the problems of the existing schemes. The proposed labeling scheme outperforms the existing labeling schemes because the structural relationships of nodes are represented with a bit string.

Key words : XML Labeling, Dynamic XML, Bit-Pattern

1. 서 론

XML 레이블링 기법은 빠른 질의 처리를 위해 XML 문서 내 노드들 간의 구조 관계를 빠르게 판단할 수 있는 레이블을 노드에 부여하는 방법으로, 기존 XML 레이블링 기법들은 *Dietz's* 번호부여 기법과 *Durable* 번호부여 기법과 같은 컨테인먼트(containment) 레이블링[1], *DeweyID* 접두사 기법과 바이너리(Binary) 문자열 접두사 기법과 같은 접두사(prefix) 레이블링[2] 그리고 *Prime* 번호부여 기법[3]과 같은 소수 레이블링으로 분류할 수 있다.

최근에는 내용이 변경되는 동적인 XML 문서에 대한 레이블링 기법에 대한 중요성이 커지고 있다. 기존 레이블링 기법은 문서 변경 시, 문서 내 노드들의 구조 관계를 정확하게 유지하기 위해 재레이블링을 수행하거나 한 노드의 레이블이 많은 정보를 표현할 수 있는 레이블링 기법을 사용한다. 하지만, 대용량 XML 문서 내에서의 재레이블링은 많은 비용이 요구되고, 많은 정보를 표현할 수 있는 레이블링 기법은 많은 저장 공간이 요구돼 질의 처리 성능이 저하되는 문제를 가진다.

Li *et al.* [4]은 기존 레이블링 기법에 적용해, XML 문서 변경에 대해 재레이블링을 피하고 레이블 저장 공간을 줄일 수 있는 *CDBS*(*Compact Dynamic Binary*

· 이 연구는 2009년도 한국과학기술원 BK21 정보기술사업단과 2009년 교육과학기술부로부터 지원받아 수행된 연구임(지역거점연구단육성사업/충북BIT연구중심대학육성사업단)
· 이 논문은 2009 한국컴퓨터종합학술대회에서 '대용량 XML 문서에서 효율적인 갱신을 위한 비트-패턴 기반의 XML 레이블링 기법'의 제목으로 발표된 논문을 확장한 것임

† 정 회 원 : KAIST 전산학과 박사후연구원
dmseo@chungbuk.ac.kr
†† 학생회원 : 충북대학교 정보통신공학과
yhPark@netdb.cbnu.ac.kr
jtlim@netdb.cbnu.ac.kr
††† 종신회원 : KAIST 전산학과 교수
mhkim@dbserver.kaist.ac.kr
†††† 종신회원 : 충북대학교 정보통신공학과 교수
yjs@chungbuk.ac.kr
(Corresponding author임)

논문접수 : 2009년 8월 14일
심사완료 : 2009년 11월 3일

String) 인코딩 기법을 제안했다. 하지만, 제안된 *CDBS*는 인코딩을 위해 할당된 비트열을 최대한 활용하지 못해 레이블 저장 공간이 효율적으로 사용되지 못하는 문제를 가진다. 또한, *CDBS*는 단지 기존 레이블링 기법에 레이블 값을 인코딩하기 위해 적용되는 방법으로, 노드들 간의 정확한 구조 관계를 표현하기 위해 여전히 많은 바이너리 문자열을 요구한다.

본 논문에서는 재레이블링을 피하고 질의 처리 성능을 향상시키기 위해 노드들의 구조 관계를 하나의 비트열로 표현하는 비트-패턴 기반의 레이블링 기법을 제안한다.

본 논문의 구성은 다음과 같다. 2장에서는 기존 XML 레이블링 기법들의 문제에 대해 서술한다. 3장에서는 본 논문에서 제안하는 비트-패턴 기반의 레이블링 기법에 대해 기술하고 4장에서는 비트-패턴 기반의 레이블에 대한 질의 처리 알고리즘에 대해 기술한다. 5장에서는 기존에 제안된 *CDBS*와의 성능 평가를 수행하고 마지막 6장에서는 결론 및 향후 연구 방향에 대해 기술한다.

2. 관련 연구

2.1 컨테인먼트 레이블링 기법

컨테인먼트 레이블링 기법에서 XML 문서 트리의 각 노드는 트리에서의 자기 위치를 기반으로 <*start*, *end*, *level*>에 대한 번호가 부여된다. 또한, 두 노드 u와 v가 있고 두 노드의 관계가 $u.start<v.start$ AND $v.end<u.end$에 있다고 가정하면, u는 v의 부모($v.level-u.level=1$) 또는 조상($v.level-u.level>1$) 노드임을 나타낸다. *Dietz's* 번호부여 기법과 *Durable* 번호부여 기법은 대표적인 컨테인먼트 레이블링 기법이다. 이 기법들은 부여된 번호를 통해 두 노드들의 관계를 빠르게 판단할 수 있다. 하지만, 레이블링을 위한 많은 저장 공간이 요구된다. 또한, *Dietz's* 번호부여 기법은 새로운 노드가 삽입될 때마다 그리고 *Durable* 번호부여 기법은 size를 초과하는 삽입 발생 시 재레이블링을 수행해야하는 문제를 가진다.

2.2 접두사 레이블링 기법

접두사 레이블링 기법에서 각 노드는 자신의 부모 노드에 부여된 레이블 값과 자신에게 부여된 값의 연결을 통해 레이블 값이 부여된다. 그리고 두 노드 u와 v가 있고 $u.label$이 $v.label$의 접두사이면, u는 v의 조상 노드임을 나타낸다. 또한, $u.label$을 $v.label$에서 제거했을 때 v에 부여된 값이 남는다면 u는 v의 부모 노드임을 나타낸다. *DeweyID* 접두사 기법과 바이너리 문자열 접두사 기법은 대표적인 접두사 레이블링 기법이다. 이 기법들은 컨테인먼트 레이블링 기법의 재레이블링 문제를 해결하기 위해 제안되었다. 하지만, *DeweyID* 접두사 기법과 바이너리 문자열 접두사 기법 모두 재레이블링 문제를 완벽하게 해결하지 못한다.

2.3 소수 레이블링 기법

Prime 번호부여 기법은 대표적인 소수 레이블링 기법으로, 트리 너비 우선 탐색을 통해 각 노드에 소수 값을 순차적으로 부여한다. 그리고 각 노드는 자신의 부모 노드에 부여된 레이블 값과 자신에게 부여된 소수 값의 곱을 통해 레이블 값이 부여된다. 만약, 두 노드 u와 v가 있고 두 노드의 관계가 $v.label \bmod u.label = 0$이면 u는 v의 조상 노드이고 $v.label/v.prime = u.label$이면 u는 v의 부모 노드임을 나타낸다. *Prime* 번호부여 기법은 컨테인먼트 레이블링 기법의 재레이블링 문제를 해결하기 위해 제안되었다. 하지만, 재레이블링 문제를 완벽하게 해결하지 못하고 대용량 XML 문서를 레이블링하는데 필요한 소수 값의 결핍현상이 발생할 수 있다.

2.4 *CDBS* 인코딩 기법

*CDBS*는 기존 제안된 레이블링 기법들에 적용될 수 있는 인코딩 기법으로 두 노드의 구조 관계는 접두사 레이블링 기법과 같은 방법으로 판단할 수 있다. 그리고 *AssignMiddleBianryString* 알고리즘을 통해 두 노드 사이에 노드 삽입 시, 재레이블링 없이 삽입된 노드에 레이블 값을 부여할 수 있는 방법을 제공한다. 또한, 레이블 값을 위해 바이너리 문자열을 사용함으로써 기존 제안된 레이블링 기법들에 비해 적은 저장 공간을 사용한다. 하지만, *CDBS*는 어떤 상황에서도 삽입된 노드에 대해 재레이블링 없이 레이블 값을 부여하기 위해, 모든 노드에 부여된 레이블 값은 끝이 "1"로 끝나는 바이너리 문자열만을 사용한다. 즉, 바이너리 문자열의 모든 비트를 레이블 값으로 사용할 수 없는 문제를 가진다. 또한, *CDBS*는 기존 레이블링 기법들에서 사용된 레이블 값을 바이너리 문자열로 인코딩하기 위해 사용되는 방법으로 두 노드들의 구조 관계를 표현하기 위해서는 많은 바이너리 문자열이 요구된다.

3. 제안하는 비트-패턴 기반의 레이블링 기법

3.1 BAT에서의 비트-패턴 할당 기법

그림 1은 제안하는 비트-패턴 기반의 레이블링 기법을 통해 XML 문서를 레이블링한 예를 보여준다. 노드 안에 부여된 번호는 레이블 순서를 나타낸다. BAT(Binary-string Assignment Table)는 XML 문서 트리 내 각 레벨별 노드들의 구조 정보를 레이블링하기 위해 사용되는 비트들의 정보를 나타내는 캐쉬(cache)이다. 또한, BAT는 질의 처리 시, 두 노드들의 구조 정보를 파악하기 위해서도 사용된다. BAT는 그림에서와 같이 레벨 정보를 표현하기 위해 사용된 비트열 정보를 나타내는 *L*(*Level info*)과 각 레벨별 노드들의 구조 정보를 표현하기 위해 사용된 비트열 정보를 나타내는 *C*(*Child*

order), *I*(*Integer order*) 그리고 *F*(*Float order*)로 구성된다. *C*는 XML 문서 내 노드들을 구분하기 위한 비트열 정보, *I*와 *F*는 XML 문서 레이블링 후 삽입되는 노드들에 대해 재레이블링 없이 레이블 값을 부여하기 위해 사용하는 비트열 정보이다. BAT의 값은 SAX 파서를 통해 XML 문서 내 각 노드가 분석될 때마다 아래의 비트-패턴에 맞춰 설정된다.

(BAT-Pattern 1) *L의 값이 분석된 노드의 트리 레벨보다 작은 경우, BAT에서 현재 L과 모든 레벨의 C, I 그리고 F에서 "1"로 설정되지 않은 가장 오른쪽 비트 값을 "1"로 설정한다.*

예를 들어, 그림 1(d)에서 *L*의 값은 "00000001"로 이 값("00000000", "00000001")을 통해 트리 2 레벨을 구분할 수 있다. 하지만, 그림 1(e)와 같이 레벨 3의 노드 6이 분석되면, 트리 3 레벨을 구분하기 위해 *L*의 값을 증가시켜줘야 한다. 이때, 그림 1(d)에서 현재 *L*과 모든 레벨의 *C*, *I* 그리고 *F*에서 "1"로 설정되지 않은 가장 오른쪽 비트 값은 두 번째 비트로 이 값을 "1"로 설정하면 그림 1(e)와 같이 BAT의 *L*은 "01000001"로 설정된다. 새로 설정된 값("00000000", "00000001", "01000000", "01000001")을 통해 트리 4 레벨을 구분할 수 있다.

(BAT-Pattern 2) 각 레벨(1 레벨 제외)의 *C* 값과 *I* 값은 동일 부모를 가지는 각 레벨 노드들 간의 순서를 구분하기 위한 값(*C*는 "0" 비트부터 시작, *I*는 "1" 비트부터 시작)으로 동일 부모에 대해 가장 많은 자식 노드를 가지는 노드 수에 의해 요구되는 비트 수가 결정된다. 만약, 분석된 노드의 순서를 구분하기 위해 *C* 값이 부족한 경우, BAT에서 현재 *L*, 모든 레벨의 *C* 그리고 같은 레벨의 *I*와 *F*에서 "1"로 설정되지 않은 가장 오른쪽 비트 값을 같은 레벨 *C*의 "1"로 설정한다. 그리고 *I* 값이 부족한 경우, 현재 *L*, 모든 레벨의 *C* 그리고 같은 레벨의 *I*에서 "1"로 설정되지 않은 가장 오른쪽 비트 값을 같은 레벨 *I*의 "1"로 설정한다.

예를 들어, 그림 1(a)에서와 같이 레벨 2의 노드 2가 분석되면, BAT-Pattern 1에 위해 *L*의 값이 "00000001"로 설정된다. 그리고 루트 노드를 부모 노드로 가지는 노드 2의 순서를 구분하기 위해 *C*의 값은 "00000010"로 *I*의 값은 "00000100"로 설정된다. 그리고 그림 1(b)에서와 같이 노드 3이 분석되면, *C*의 값("00000000"과 "00000010")은 노드 2와 노드 3을 구분할 수 있지만 *I*의 값은 노드 2와 노드 3을 구분할 수 없기 때문에 *I*의 값은 "00001100"으로 설정된다. *I*의 값("00000100", "00001000"과 "00001100")은 세 개의 노드를 구분할 수 있는 값으로 사용된다. 또한, 그림 1(c)에서와 같이 노드 4가 분석되면, *C*의 값으로 세 개의 노드를 구분할 수 없기 때문에 *C*의 값은 "00010010"로 설정된다.

(BAT-Pattern 3) 각 레벨에서 새로 삽입된 노드의 레이블링을 위해 요구되는 *F* 값이 부족한 경우, BAT에서 현재 *L*, 모든 레벨의 *C* 그리고 같은 레벨의 *I*와 *F*에서 "1"로 설정되지 않은 가장 오른쪽 비트 값을 같은 레벨 *F*의 "1"로 설정한다. *F*의 부족은 *L*, *C* 그리고 *I*의 부족과는 차이가 있다. 예를 들어, *F*, *L*, *C* 그리고 *I*에 "111"이 설정되면 *L*과 *C*는 8개의 값을 구분하기 위해, *I*는 7개의 값을 구분하기 위해 사용된다. 하지만, *F*는 "000", "001", "011", "101" 그리고 "111"로 끝 비트가 "1"로 끝나는 형태로 사용되기 때문에 5개의 값을 구분하기 위해 사용된다. (자세한 설명은 3.3에서 기술)

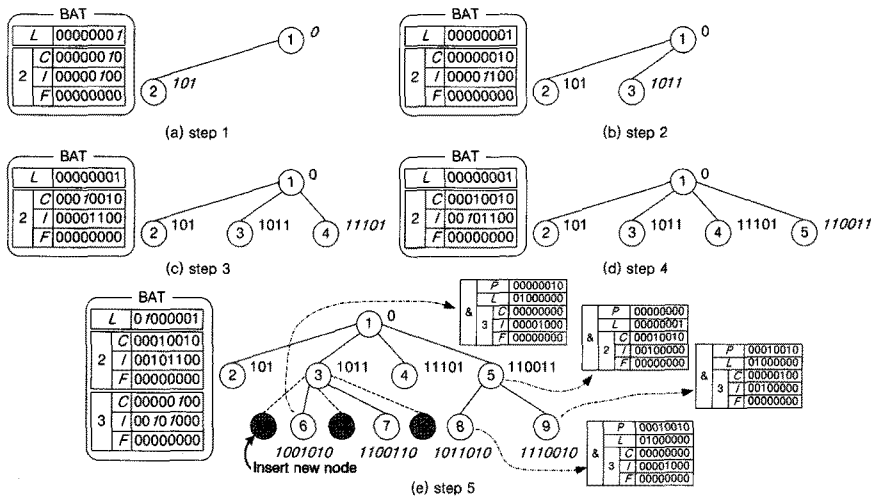

그림 1 제안하는 비트-패턴 기반의 레이블링 기법

3.2 비트-패턴 기반의 레이블링 알고리즘

그림 2는 비트-패턴 기반의 레이블 부여 알고리즘을 보여준다. 그림 1(e)는 *AssignLabelBinaryString* 알고리즘을 통한 레이블링 과정을 보여준다. 예를 들어, XML 트리에서 레벨 3을 가지는 노드 8에 레이블 값을 부여하기 위해, 먼저 부모 노드 5번에 부여된 레이블 값 "110011"과 BAT에서 레벨 2의 C 값 "00010010"에 대한 AND 연산을 통해 "00010010"을 획득한다. 이 값은 부모 노드가 4번째 자식 노드임을 나타낸다(Algorithm 1. Line 07). 그리고 레벨 3을 구분하기 위해 BAT의 L에서 할당된 값 "01000000"을 획득한다(Algorithm 2. Line 09). 또한, BAT에서 노드 5의 첫 번째 자식 노드 8을 구분하기 위해 할당된 C 값 "00000000"과 I 값 "00001000"을 획득한다(Algorithm 1. Line 10~11). 마지막으로, 획득한 모든 값들에 대한 AND 연산 결과 값 "01011010"을 노드 8의 레이블 값으로 부여한다.

Algorithm 1: *AssignLabelBinaryString(N, P, BAT)*

```
Input: N is a parsered node by SAX
       P is the parent node of N
Output: The label binary string for N
01 if N.level == 1 then // N.level is level in XML tree
02     return "0" // The root node is assigned with "0"
03 else if N.level > 1 then
04 // P.label is the label binary string of P
05 // BAT(N.level).C is the child order with N.level in BAT
06     PC_OBS = P.label & BAT(N.level).C
07 // OBS(BAT.L, N.level) is the assigned order binary
08 // string for level info of N.level in BAT
09     NL_OBS = OBS(BAT.L, N.level)
10 // N.PCN is the child number of N in P
10     NCL_OBS = OBS(BAT(N.level).G, N.PCN) &
11                OBS(BAT(N.level).h, N.PCN)
12     return PC_OBS & NL_OBS & NS_OBS
13 end
```

그림 2 비트-패턴 기반의 레이블 부여 알고리즘

3.3 동적 갱신 레이블링 알고리즘

그림 3은 XML 문서 레이블링 후 삽입되는 노드들에 대한 레이블 부여 알고리즘을 보여준다. 그림 1(e)는 *DynamicUpdateLabeling* 알고리즘을 통한 레이블링 과정을 보여준다. 예를 들어, XML 트리에서 레벨 3을 가지는 노드 10이 삽입되면, 노드 6에 부여된 레이블 값보다 I 값과 F 값에 대해서만 사전적으로(lexicographically) 앞서는 바이너리 문자열이 부여되면 된다. 노드 10은 오른쪽 형제 노드 6만을 가지고 있기 때문에, 노드 6을 기준으로 레이블 값을 획득한다. 노드 6의 I 값은 "00001000"으로 첫 번째 자식 노드이고 F 값은 "0"이다(Algorithm 3. Line 04). 노드 6의 I 값과 F 값은 실질적으로 "01.0" 의미하므로 사전적으로 앞서는 바이너리 문자열 "00.1"을 만들 수 있다. 새로 삽입되는 노드들의 레이블 값으로 소수 부분의 끝 비트가 "1"로 끝나는 것은 어떤 상황에서도 두 노드들 사이에 삽입되는 노드들에 대한 레이블 값을 획득할 수 있기 때문이다. "00.1"의 소수 부분을 구분하기 위해 BAT의 레벨 3에 대한 F 값을 BAT-Patter 3에 의해 "10000000"으로 할당하고 노드 10에 대한 F 값으로 "10000000"을 부여한다(Algorithm 2. Line 05). 그리고 노드 10에 대한 C 값은 삽입 위치에 상관없이 순차적으로 증가하기 때문에 세 번째 자식노드로 "100000000"가 부여된다(Algorithm 2. Line 01). 마지막으로, Algorithm 2. Line 14을 수행하면 노드 10에 대한 레이블 값 "111000010"을 획득할 수 있다.

4. 비트-패턴 기반의 레이블에 대한 질의 처리

(Structure-Pattern 1) *노드 u가 있을 때, BAT.L & u.label의 결과 값은 XML 트리에서 u의 레벨을 나타낸다.* 예를 들어, 그림 1(e)에서 노드 3의 레이블 값 "1011"과 BAT.L 값 "01000001"의 AND 연산한 결과 값은 "00000001"이다. 이 값은 트리의 레벨을 구분하기 위해 $BAT._L$에서 사용되는 "00000000", "00000001", "01000000" 그리고 "01000001"의 두 번째 값으로 노드 3이 레벨 2임을 나타낸다.

(Structure-Pattern 2) *$u.level \le v.level$ 관계를 가지는 노드 u와 v가 있고, 레벨 2부터 $u.level$까지의 모든 $BAT._C$를 OR 연산한 결과 값을 ADF(Ancestor-Descendent Filter)라고 가정하자. 이때, ADF & $u.level$ 결과 값과 ADF & $v.level$ 결과 값이 같으면 u는 v의 조상 노드임을 나타낸다. 물론, $v.label - u.label = 1$이면, u는 v의 부모 노드임을 나타낸다.* 예를 들어, 그림 1(e)에서 노드 3의 레벨이 노드 7의 레벨보다 작다. 그러므로 ADF의 값은 "00010010"이다. 이때, ADF와 노드 3, 노드 7 레이블 값 각각을 AND 연산한 결과 값은 "00000010"이고 레벨 차는 1이기 때문에 노드 6은 노드 7의 부모 노드임을 나타낸다.

5. 성능 평가

본 논문에서 제안하는 피트-패턴 기반의 레이블링 기법인 *BPDBS(Bit Pattern Dynamic Binary String)*의 우수성을 증명하기 위해, 기존에 제안된 *CDBS-C (Dietz's with CDBS)*, *CDBS-P(DeweyID with CDBS)*와 저장 공간 측면에서 성능 평가를 수행했다. 트리 높이 L과 한 노드의 평균 자식 노드 수 N을 가지는 XML 문서에서 한 노드 레이블을 위해 최대 필요한 비트 수는 아래 수식과 같다. 식 (1)은 *CDBS-C*, 식 (2)는 *CDBS-P* 그리고 식 (3)은 *BPDBS*에서의 한 노드의 최대 비트 수이다.

Algorithm 2: *DynamicUpdateLabeling(N, P, SL, SR, BAT)*

Input: *N* is the new inserted node, *SL* is the left sibling node, and *SR* is the right sibling node
Output: The label binary string for *N*

```
01 PC_OBS = P.label & BAT[N.level].C ; NL_OBS = OBS(BAT.L, N.level) ; NC_OBS = OBS(BAT[N.level].G, N.PCN)
02 if SL == null then
03    // ON(A, B) returns the order number of the result with A&B in A
04    if (ON(BAT[N.level].I, SR.label) == 1) && ((BAT[N.level].F & SR.label) == "0") then
05       NI_OBS = "0" ; NF_OBS = OBS(BAT[N.level].F, 1)
06    else
07       NI_OBS = BAT[N.level].I & SR.label
08       NF_OBS = OBS(BAT[N.level].F, ON(BAT[N.level].F,
09                                    (BAT[N.level].F & SR.label) with last bit "1" changed to "01"))
10 else
11    NI_OBS = BAT[N.level].I & SL.label
12    NF_OBS = OBS(BAT[N.level].F, ON(BAT[N.level].F, (BAT[N.level].F & SL.label) ⊕ "1"))
13 end
14 return PC_OBS & NL_OBS & NC_OBS & NI_OBS & NF_OBS
```

그림 3 동적 갱신 레이블링 알고리즘

$$\lceil \log(2N^L+1) \rceil \times 2 + \lceil \log L \rceil \quad (1)$$

$$\lceil \log(N+1) \rceil \times L \quad (2)$$

$$\lceil \log N \rceil \times L + \lceil \log L \rceil + \lceil \log N \rceil \quad (3)$$

그림 4는 N이 5일 때, L에 따른 한 노드의 최대 비트 수를 보여준다. 그림 5는 L이 4일 때, N에 따른 한 노드의 최대 비트 수를 보여준다. *CDBS-P*는 전체 노드 수, *CDBS-C*와 *BPDBS*는 각 레벨에서의 노드 수에 의해 최대 비트 수가 결정된다. 그리고 *CDBS-C*는 부모 노드의 레이블 값이 자식 노드에 반영되지만, *BPDBS*는 반영되지 않기 때문에 제안하는 기법이 저장 공간 측면에서 가장 좋은 성능을 가진다.

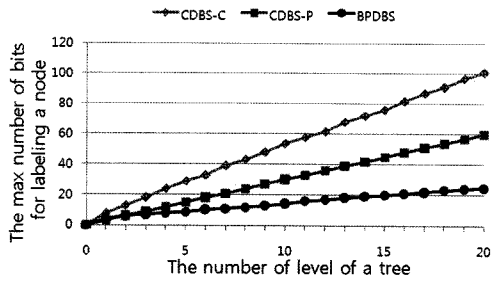

그림 4 트리 높이에 따른 노드 레이블 비트 수

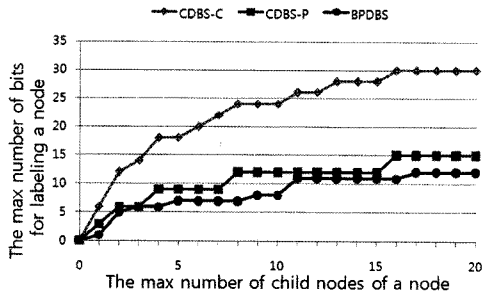

그림 5 자식 노드 수에 따른 노드 레이블 비트 수

6. 결론 및 향후 연구

본 논문에서는 XML 문서 변경에 대한 재레이블링을 피하고 질의 처리 성능을 향상시키기 위해 최적화된 저장 공간을 사용하는 비트-패턴 기반의 레이블링 기법을 제안했다. 향후, 다양한 XML 문서와 질의를 가지고 기존 레이블링 기법들에 비해 저장 공간 및 질의 처리 측면에서 우수한 성능을 나타냄을 보일 것이다.

참 고 문 헌

[1] L. Quanzhong, M. Bongki, "Indexing and Querying XML Data for Regular Path Expressions," *Proc. of International Conference on Very Large Data Bases*, pp.361-370, 2001.

[2] L. Changging, T.W. Ling, "An Improved Prefix Labeling Scheme: A Binary String Approach for Dynamic Ordered XML," *Proc. of International Conference on Database Systems for Advanced Applications*, pp.125-137, 2005.

[3] W. Xiaodong, L. Mongli, H. Wynne, "A Prime Number Labeling Scheme for Dynamic Ordered XML Trees," *Proc. of International Conference on Data Engineering*, pp.66-78, 2004.

[4] L. Changging, T.W. Ling, H. Min, "Efficient Updates in Dynamic XML Data: from inary string to quaternary string," *The Very Large Data Bases Journal*, vol.17, no.3, pp.573-601, 2008.