Journal of Korea Multimedia Society Vol. 8, No. 5, May 2005(pp. 606-617)

XML 문서 갱신을 위한 확장 가능한 노드 넘버링 구조

박충희[†], 구흥서[††], 이상준[†††]

요 약

XML 질의 처리를 위한 XML 문서 트리내 노드들간의 구조적 관계를 효율적으로 찾는 많은 연구들이 수행되었으며 이러한 연구들의 대부분은 노드들의 위치에 기반한 영역 넘버링을 사용하고 있다. 그러나 위치 기반의 노드번호 부여 방식은 동일장소에 반복적으로 노드을 삽입할 때 기존 노드번호들의 값 재조정을 필요로 한다. 본 논문에서는 가변길이 문자열을 이용한 버켓단위의 노드번호를 부여함으로써 재조정 노드수를 줄이는 ENN(Extensible Node Numbering) 방법을 제안한다. 또한 기존의 노드번호 부여 방식인 EP(extended preorder)와의 성능 비교를 실시하였다.

Extensible Node Numbering Scheme for Updating XML Documents

Park Chung-Hee[†], Koo Heung-Seo[††], Lee Sang-Joon[†††]

ABSTRACT

There have been many research efforts which find efficiently all occurrences of the structural relationships between the nodes in the XML document tree for processing a XML query. Most of them use the region numbering which is based on positions of the nodes. But The position-based node numbering schemes require update of many node numbers in the XML document if the XML nodes are inserted frequently in the same position. In this paper, we propose ENN(extensible node numbering) scheme using the bucket number which is represented with the variable-length string for decreasing the number of the node numbers which is updated. We also present an performance analysis by comparing ENN method with EP(extended preorder) method.

Key words: XML, Node Numbering Scheme(노드 번호부여 방식), Structural Relationship(구조적 관계)

1. 서 론

XML 질의 처리는 노드들간의 기본적인 부모/자식 관계와 조상/자손 관계의 집합으로 분해되고 난 후 각각의 이진 구조적 관계를 적용하여 구한 결과를 기반으로 최종 결과를 생성한다. 그러므로 기본적인 구조적 관계의 모든 어커런스들을 구하는 작업은 XML 데이터베이스의 관계 구현이나 네이티브 시스템(native system)을 이용한 방식 모두에서 XML 질의 처리의 핵심 연산이 된다[2].

네이티브 XML 데이터베이스 엔진뿐만 아니라 관계형 데이터베이스 관리 시스템을 기반으로 한 구조적 관계의 어커런스들을 구하는 방법들에 대해 많은 연구가 이루어져 왔다[4,6-10]. 이러한 연구들은 공통적으로 구조적 조인을 처리하기 위해 XML 태그들의 노드번호를 사용하고 있다. 노드번호는 XML 문서의 구조적 특성을 나타내기 위해 XML 엘리먼트들이나 문자열노드의 위치를 기반으로 한 값들의 쌍으로 표현된다[2,5,6,10].

그러나 이 연구들에서 사용한 문서내의 노드들에

※ 교신저자(Corresponding Author) : 박충희, 주소 : 제주도 제주시 영평동 2235(690-714), 전화 : 064)754-0310, FAX : 064)754-0313, E-mail :chpark_cs@jeju.ac.kr
접수일 : 2004년 7월 7일, 완료일 : 2005년 1월 4일
† 제주대학교 컴퓨터공학과 박사과정
†† 청주대학교 정보기술공학부 부교수
(E-mail : hskoo@cju.ac.kr)
††† 제주대학교 컴퓨터공학과 부교수
(E-mail : sjlee@cheju.ac.kr)

대한 번호부여 방식은 XML 데이터가 삽입이 되면 엘리먼트들의 구조적 관계가 변경되어 노드번호들의 재구성을 야기시킨다. 따라서 갱신이 빈번한 XML 문서의 경우는 심각한 성능저하를 야기시킬 수 있다. 이러한 문제를 해결하기 위해 최근에 제안된 확장 프리오더(extended preorder) 방식[3]에서는 노드 삽입연산에 의해 발생되는 기존의 노드번호 재조정 수를 줄이기 위하여 초기에 노드번호를 할당할 때 향후 노드삽입을 위한 여유번호 공간을 포함하여 노드번호를 부여한다. 그러나 이 방식은 삽입장소와 삽입량을 모르는 상태에서 사전에 고정된 여유공간을 확보, 할당하기 때문에 번호공간 활용도가 떨어진다. 대량의 여유공간을 확보한 경우에도 동일장소에 반복 삽입이 발생하면 몇 회 지나지 않아 여유공간을 초과하게 되고 이는 인접 노드중 여유공간을 갖고 있는 맨 처음 노드를 인식하고 그 사이에 존재하는 기존노드들에 대해 노드번호 재조정을 발생시킨다. 따라서 노드 삽입비용보다 재조정비용이 과다하여 전체 시스템의 성능을 저하시킨다.

본 논문에서는 저장된 XML 문서에 대해 노드삽입시 재조정되는 노드번호의 수를 최소화하기 위해 새로운 노드 넘버링 방식 ENN(Extensible Node Numbering)을 제안한다. 이 방법은 가변길이 문자열을 이용한 버켓단위의 노드번호를 사용하여 사전에 필요한 모든 노드번호를 확보하지 않고 일정량의 여유공간(버켓크기)을 확보한 후 향후 삽입에 의하여 노드번호가 필요할 경우 버켓내 오프셋(offset)을 이용하여 추가로 노드번호를 확보한다. 오프셋이 부족할 경우 동적으로 새로운 버켓단위의 노드번호를 생성, 확보함으로써 사전 번호공간 확보의 단점을 개선한다. 노드삽입에 따른 재조정 노드 수는 최악의 경우 한 개 버켓내 노드 수로 제한된다. 본 논문의 구성은 먼저 2장에서 기존에 연구된 노드번호 부여 방식을 살펴보고, 3장에서 본 논문에서 제안하는 ENN 방식을 설명한다. 그리고 4장에서는 제안방식에 따른 성능 분석 및 평가를 내리고 5장에서 결론을 맺는다.

2. 노드번호 부여 방식

XML 문서의 구조적 특성을 표현하기 위해 XML 엘리먼트들이나 문자열 노드들의 위치를 기반으로 한 노드번호 부여방식에 대한 많은 시도가 있었다 [1,3,5,10,11]. Dietz의 번호부여 방식(Dietz's numbering scheme)[1]은 XML 트리를 각각 preorder와 postorder로 순회하여 각 노드당 <preorder, postorder>를 부여하며, 이 값을 노드의 레벨(level)과 함께 사용하여 두 노드간 구조적 관계를 결정한다. 하나의 노드 v가 노드 u의 조상이라면 preorder(v) < preorder(u) 그리고 postorder(v) > postorder(u)를 만족한다. 그리고 v가 u의 부모라면 추가적으로 level(v)+1 = level(u)를 만족한다.

[5]에서는 노드번호를 문서에서 노드가 위치한 시작태그와 끝태그의 절대 바이트 오프셋 값들의 쌍을 가지고 표현함으로써 노드들의 구조적 관계를 범위 포함관계로 결정하였다. 하나의 노드 v가 노드 u의 조상이라면 start(v) < start(u) < end(v)를 만족한다. 그리고 v가 u의 부모라면 추가적으로 level(v)+1 = level(u)를 만족한다.

[10]에서는 노드번호로 바이트 단위의 오프셋 대신 워드단위의 오프셋을 사용함으로써 문자열 노드의 길이가 변경될 때 노드번호 재조정의 필요성을 제거하였다. 그러나 여전히 삽입되는 엘리먼트들로 인한 기존 노드번호의 갱신은 불가피하다.

최근에 연구된 확장 프리오더 방식[3]은 시작 태그와 끝 태그에 노드번호를 부여할 때 나중에 삽입될 노드들의 노드번호들을 위해 여유 오프셋 번호를 포함하도록 노드번호를 할당하는 방식이다. 이 방법은 XML 문서 트리를 preorder로 순회하여 각 노드에 대해 <order, size>값을 부여한다. order는 여유 번호공간을 갖는 preorder값이며 size는 향후 추가될 자손들을 위한 여유 번호공간의 크기이다. 하나의 노드 v가 노드 u의 조상이라면 order(v) < order(u) ≤ order(v) + size(v)를 만족한다. 그리고 v가 u의 부모라면 추가적으로 level(v) +1 = level(u)를 만족한다. 이 방식은 여유 번호공간을 미리 고정 확보함으로써 여유 공간내의 노드삽입인 경우에는 기존 노드번호들이 재조정되지 않으나 다음과 같은 문제점을 가지고 있다.

- 삽입이 발생될 장소와 삽입량을 모르는 상태에서 미리 여유 번호공간을 고정적으로 확보하기 때문에 번호공간 활용측면에서 효율성이 낮고, 과다한 여유 번호공간의 확보는 노드번호 저장공간의 낭비를 야기한다.
- 여유 번호공간을 초과하는 노드 삽입이 발생할

경우 인접노드들 중 여유공간을 확보하고 있는 맨 처음 노드 식별을 위한 노드 탐색과 중간에 존재하는 노드번호 값들에 대해 재조정이 발생할 수 있다. 최악의 경우는 삽입지점 이후의 모든 노드번호들을 재조정해야 한다.

- 특정노드의 조상 노드들은 인접 노드들과는 달리 디스크 상에서 물리적으로 다른 페이지에 위치할 가능성이 높다. 여유공간을 초과하는 노드 삽입시 추가 여유공간 확보를 위해 삽입지점의 조상 노드번호 값 재조정을 필요로 하는 경우가 발생한다. 이 때 조상노드 값 재조정 비용은 인접 노드번호 값 재조정비용보다 크다. 최악의 경우 1개의 노드번호 갱신마다 1페이지 갱신을 필요로 하게 될 것이다.

본 논문에서는 이러한 확장 프리오더의 문제점을 개선하기 위해 새로운 노드번호 구조를 설계, 제안하였다. 제안방식은 노드 삽입시 재조정 대상 노드수를 줄이기 위해 사전에 고정크기의 여유공간을 확보하는 측면에서는 기존의 연구와 유사하나 여유공간을 초과하는 노드삽입이 발생할 때 노드번호를 추가로 확보하는 방법이 다르다. 확장 프리오더 방식은 여유공간이 부족하면 인접한 노드번호들을 탐색하고 해당 노드번호들을 재조정하는 사전 고정 번호공간 확보방식이다. 반면 제안한 방식은 여유공간이 부족할 경우 이웃한 두 버켓번호 사이 값을 갖는 새로운 버켓번호를 동적으로 생성하여 확보하는 동적 가변 번호공간 확보방식이다.

3. ENN(Extensible Node Numbering)

이번 장에서는 논문에서 제안한 노드번호 부여방식의 기반이 되는 XML 문서 모델, 노드번호 구조, 버켓번호 인코딩, 노드번호 부여 방법 및 알고리즘을 설명한다.

3.1 XML 문서 모델

XML 문서는 순서 트리(ordered tree)로 표현된다. XML 문서트리의 노드들은 엘리먼트 노드, 애트리뷰트 노드, 그리고 텍스트 노드로 구성된다.

XML 문서트리의 각 노드들을 깊이우선순위(depth-first traversal) 방식으로 방문하며, 방문할 때마다 번호를 부여한다. 엘리먼트는 두 번씩 방문되며, 부여받은 번호는 범위를 표현한다. 애트리뷰트와 텍스트 노드는 별도의 값을 부여받지 않고 소속된 엘리먼트 노드의 시작 태그에 해당하는 노드번호 값을 상속받는다. 그러므로 두 노드간의 구조적인 포함관계는 노드간의 범위 포함관계로 결정할 수 있다.

3.2 노드번호 구조

ENN 노드번호 부여 방법에서 하나의 노드번호는 그림 1에서와 같이 (버켓번호,오프셋)으로 구성되며 이는 데이타베이스 관리 시스템의 화일 관리자에서 레코드 식별자 RID = (페이지식별자, 오프셋)와 유사한 형태를 갖는다. 버켓번호는 세부분으로 구성되며 각 용도는 다음과 같다.

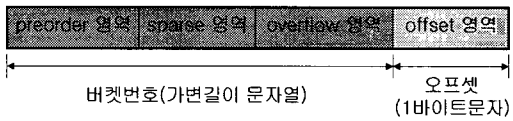


그림 1. ENN 노드번호의 구조

- preorder 영역:

초기 XML 문서 저장시 각 노드에 번호를 순차적으로 부여하기 위해 사용됨.

- sparse 영역:

동적으로 새로운 버켓번호를 생성하기 위해 추가적으로 사용하는 영역으로 확장 허용 횟수 범위내에서 사용됨.

- offset 영역:

sparse 영역을 모두 소모하였을 때 새로운 노드번호 생성을 위해 사용됨.

- overflow 영역:

버켓번호에 소속된 offset 영역이 부족한 경우 새로운 버켓번호를 위해 추가로 사용됨.

버켓번호는 문자열 값을 사용하여 순차적으로 표현되며 오프셋은 버켓내 순서번호로 초기값은 0으로 설정된다. 노드 삽입으로 인접한 두 버켓번호간에 새로운 버켓번호가 필요할 경우 동적으로 생성되는 새로운 버켓번호는 문자열 값을 기준으로 순서상 두 버켓번호 사이에 위치해야 한다. 이를 위해 버켓번호는 가변길이 문자열을 사용하며 특히 3.3절에서는 문자열을 이용한 버켓번호의 효율적 표현을 위하여 인코딩 방법을 자세히 설명한다.

3.3 버켓번호 인코딩

노드번호는 XML 문서내에 사용된 엘리먼트들에 대하여 유일 값을 가져야 한다. A~Z, a~z, 0~9 값을 이용한 고정길이 문자열 형식의 버켓번호 인코딩은 표현범위에 비해 필요한 문자열의 길이가 길다. 따라서 본 논문에서는 최상위 바이트에 길이 표시 비트들을 가지는 가변길이 문자열과 바이트 내 모든 비트 값을 사용하는 인코딩 방식을 사용한다. 그림 2는 길이비트가 2일 때 인코딩된 버켓번호 문자열과 대응되는 10진값, 그리고 환산식을 제시하고 있다.

3.4 노드번호 부여 방법

논문에서 제안하는 ENN 방식은 초기 XML 문서를 저장할 때 XML 문서트리를 깊이-우선 탐색방식에 따라 순회하며 엘리먼트들에 대해 2번씩 방문한다. 이때 노드번호는 preorder 영역만을 사용하는 버켓번호 형태로 순차 할당되며 소속 offset 영역의 오프셋은 0으로 초기화된다. 그림 3은 예제 XML 문서의 XML 트리에 대해 제안한 노드번호 방식으로 노드번호를 부여한 결과를 나타낸 것이다. 한편 XML 문서트리의 노드(엘리먼트)들을 저장할 때 시작태그에 해당하는 노드번호를 기준으로 B+_tree를 생성함으로써 XML 문서내 노드들이 도큐먼트 순서(document order)를 유지하도록 할 수 있다.

향후 노드 삽입이 발생할 경우 확장허용 횟수 내에서 동적으로 sparse 영역을 추가 사용하여 즉 preorder 영역과 sparse 영역으로 구성된 새로운 버켓번호를 사용하여 이웃한 두 버켓번호의 사이 값을 갖도록 순차 할당하고 오프셋은 0으로 초기화한다(버켓단위 할당). sparse 영역이 모두 소모되어 확장허용 횟수에 도달되면 새로운 버켓번호를 생성하지 않고 이미 부여된 버켓번호에 소속된 offset 영역의 현재 오프셋 다음 값부터 순차적으로 부여한다(오프셋 단위 할당). 동일 장소의 반복 삽입으로 인하여 잔여 오프셋이 부족할 경우 overflow 영역을 추가로 사용하여 preorder 영역, sparse 영역, overflow 영역으로 구성된 새로운 버켓번호를 생성하여 버켓단위로 할당한다. 향후 추가삽입은 offset 영역을 사용하는 오프셋 단위로 할당이 이루어진다. 따라서 노드 삽입에 의하여 재조정이 필요한 대상 노드들은 노드 삽입지점 이후 삽입지점과 동일한 버켓번호을 갖는 버켓내 존재하는 노드번호들로 한정된다.

그림 4는 그림 3의 XML 문서트리에 노드 삽입이 발생함에 따라 삽입 노드들의 노드번호와 기존 노드

```
<books>
  <book id= 1>
    <author> <name>김혜수</name>
             <tel>02-734-1111</tel>
    </author>
    <caption> <title> 가을날의 동화</title>
              <vol>150</vol>
    </caption>
  </book>
  <book id= 2>
    <author> <name>박민수</name>
             <tel>02-744-4455</tel>
    </author>
    <caption> <title>추억일기</title>
              <vol>100</vol>
    </caption>
  </book>
</books>
```

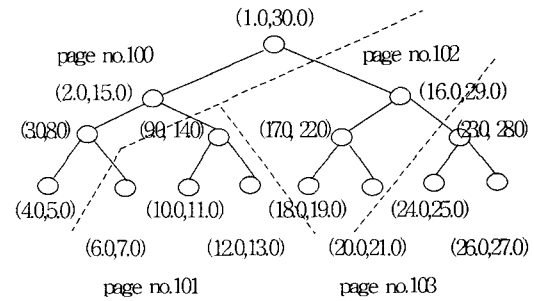


그림 3. 예제 XML 문서와 제안 방법에 의해 노드번호가 부여된 XML 트리

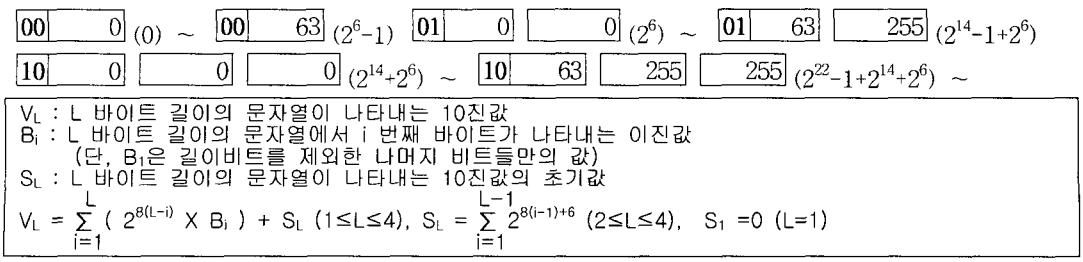


그림 2. 가변길이 문자열로 인코딩된 버켓번호 값과 대응 10진값

번호들의 값이 재조정되는 과정을 나타내고 있다.

확장허용 횟수(l_level)를 2, 오프셋의 최대값(offset_max)을 8이라 가정한다. 노드번호 중간에 존재하는 공백은 실제로는 존재하지 않고 다만 이해를 돕기 위해 사용하며 검정 색 노드들은 새로 삽입된 노드들을, 사각 노드번호들은 재조정된 노드번호들을, 밑줄 친 노드번호는 삽입지점의 노드번호를 나타낸다. 그림 4(a)는 그림 3의 11.0 노드번호 뒤에 3개의 노드가 삽입된 상태를 나타내고 있다. 현재 확장횟수(c_level: 0)가 확장허용 횟수 2보다 작으므로 sparse 영역을 추가 사용하여 순차적으로 새로운 버켓번호를 부여한다. 그림 4(a)의 11 4.0 노드번호 뒤에 노드 3개가 삽입될 경우에 sparse 영역을 추가 사용하여 버켓단위의 노드번호가 순차 부여되며 그림 4(b)는 이를 나타낸다. 그림 4(c)는 현재 확장횟수가 확장허용횟수와 같은 상황 즉 sparse 영역이 모두 소모된 상황에 11 4 3.0 노드번호 뒤에 3개의 노드가 삽입된 경우이다. 11 4 3.0 노드번호의 오프셋 다음 값인 1부터 순차적으로 오프셋을 할당한다(오프셋 단위 할당). 계속해서 그림 4(c)의 11 4 3.4 노드번호 뒤에 노드 3개를 삽입한다고 가정하자. 이 경우는 11 4 3.4 노드번호의 버켓번호 '11 4 3' 내 잔여 오프셋 갯수 2(=8-6)가 필요한 오프셋 갯수 6(=3*2)보다 작고 '11 4 3' 버켓번호의 다음 버켓번호 '11 4 4' 사이에 빈 버켓번호가 없는 상황이다. 따라서 overflow 영역을 추가 사용하여 버켓단위로 노드번호를 순차 할당한다. 그리고 삽입지점의 노드번호 11 4 3.4 이후의 노드번호 중 동일한 버켓번호를 가진 노드번호들(11 4 3.5, 11 4 3.6)은 재조정(11 4 3 7.0, 11 4 8.0)된다. 그림 4(d)는 이러한 결과를 나타내고 있다. 그림 4(e)는 그림 4(d)의 11 4 3 3.0 노드번호 뒤에 2개의 노드가 삽입되는 경우이다. 버켓번호 '11 4 3 3' 내 잔여 오프셋 갯수 8은 필요한 오프셋 갯수 4보다 크므로 삽입노드들의 노드번호는 11 4 3 3.0 노드번호의 오프셋 다음 값인 1부터 순차적으로 부여된다.

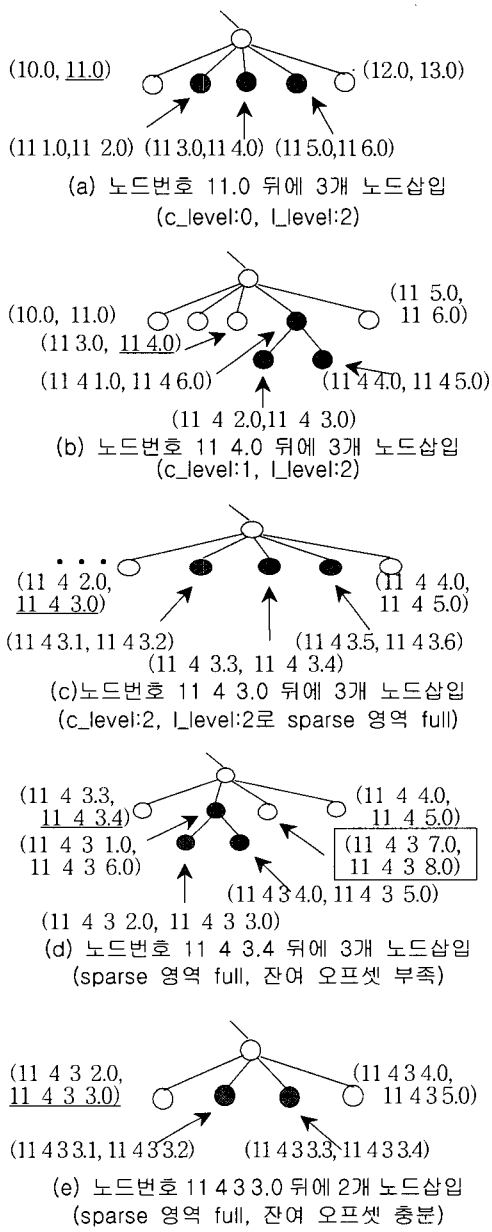


그림 4. 제안 방법에 의한 노드번호 부여 과정

3.5 노드번호 부여 알고리즘

그림 5(a)의 알고리즘에서 삽입지점 노드의 끝 태그에 해당하는 노드번호(cur_nid2)와 탐색순서상 바로 인접한 노드번호(next_nid) 사이의 여분의 버켓번호 갯수(Nrem)와 삽입노드들을 위해 요구되는 총 노드번호 수(Nneed)를 먼저 구한다. 삽입지점 노드번호 위치에 잔여 버켓번호가 존재하면(Nrem > 0) 잔여 번호를 사용하여 삽입노드들에 대해 할당한다(Redundant_Bucket_Numbering). 번호를 부여할 노드가 남아 있으면(Nneed>0) last_nid의 현재 확장 레벨(c_level)과 확장 허용레벨(l_level)을 비교한다. sparse 영역이 남아 있으면(c_level < l_level) sparse 영역을 추가 사용하여 노드번호를 할당한다(Sparse_

```
Input:삽입노드들이 문서 순서로 정렬된 태그 list(input_tree), 삽입지점 노드의 끝 태그 노드번호 값(cur_nid2)
Output:노드번호들이 할당된 XML 입력트리(input_tree), 재조정 노드번호 list
Extensible_Node_Numbering
{ index <- 1 ; last_nid <- cur_nid2 ; next_nid <- cur_nid2 노드번호 바로 다음의 노드번호 값 ;
  Nrem<- Bucket_no(next_nid) - Bucket_no(cur_nid2) -1 ;
  Nneed <- the number of the nodes in the input tree * 2 ;
  if(Nrem>Ø) then Redundant_Bucket_Numbering(input_tree, last_nid, next_nid, Nrem, Nneed);
  if(Nneed > Ø )
  then if(c_level < l_level) then Sparse_Area_Numbering(input_tree, last_nid, next_nid, Nneed)
       else { Noffset <- offset_max - Offset(last_nid) ; //잔여 offset 수
              if(Noffset ≥ Nneed ) then { Offset_Area_Numbering(input_tree, last_nid, Nneed);
                                          cur_nid2와 동일 버켓 값을 갖는 재조정 노드번호 list 반환;}
              else { Overflow_Area_Numbering(input_tree, last_nid, next_nid, Nneed) ;
                     cur_nid2와 동일 버켓 값을 갖는 재조정 노드번호 list 반환;  }
}
```

(a) 메인 알고리즘

```
Redundant_Bucket_Numbering(input_tree, last_nid, next_nid, Nrem, Nneed)
{ if (Nrem ≥ Nneed)
  then { last_nid와 next_nid 사이의 여분 버켓번호를 이용하여 순차적으로 input_tree[]에 노드번호 값 할당 ;
         Nneed <- Ø ; }
  else { last_nid와 next_nid 사이의 여분 버켓번호를 이용하여 순차적으로 input_tree[]에 노드번호 값 할당 ;
         Nneed <- Nneed - Nrem ; last_nid <- 맨 마지막에 할당된 노드번호;
         index <- 맨 마지막에 할당된 노드번호를 갖는 input_tree의 인덱스 +1 ;   }
}
Sparse_Area_Numbering(input_tree, last_nid, next_nid, Nneed)
{ if (length(Bucket_no(last_nid)) < length(Bucket_no(next_nid)) and
     Bucket_no(next_nid)의 마지막 바이트 값이 X'01))
  then input_tree[index].bucket <- Bucket_no(last_nid) & X'00 & X'01
  else input_tree[index].bucket <- Bucket_no(last_nid) & X'01 ;
  input_tree[index].offset <- Ø ;  index <- index +1 ;
  for i = 1 to Nneed -1 do
  { input_tree[index+i].bucket <- input_tree[index+i-1].bucket의 값 1증가 ;
    input_tree[index+i].offset <- Ø ; }
  index <- index + Nneed -1 ;
}
Offset_Area_Numbering(input_tree, last_nid, Nneed)
{ input_tree[index].bucket  <- Bucket_no(last_nid) ;
  input_tree[index].offset  <-  Offset_no(last_nid) + 1 ; index <- index +1 ;
  for i = 1 to Nneed -1 do
  { input_tree[index+i].bucket <- Bucket_no(last_nid) ;
    input_tree[index+i].offset <- input_tree[index+i-1].offset + 1 ; }
  last_nid <- input_tree[index+Nneed] ;  index <- index + Nneed -1 ;
}
Overflow_Area_Numbering(input_tree, last_nid, next_nid, Nneed)
{ if ( length(Bucket_no(last_nid)) < length(Bucket_no(next_nid)) and
      Bucket_no(next_nid)의 마지막 바이트 값이 X'01))
  then input_tree[index].bucket <- Bucket_no(last_nid) & X'00 & X'01
  else input_tree[index].bucket <- Bucket_no(last_nid) & X'01 ;
  input_tree[index].offset <- Ø ;  index <- index +1 ;
  for i = 1 to Nneed -1 do
  { input_tree[index+i].bucket <- input_tree[index+i-1].bucket의 값 1증가 ;
    input_tree[index+i].offset <- Ø ; }
  last_nid <- input_tree[index+Nneed] ;  index <- index + Nneed -1 ;
}
```

(b) 서브 알고리즘

그림 5. ENN 방식의 노드번호 부여 알고리즘

_Area _Numbering). sparse 영역이 남아 있지 않으면(c_level=l_level) 더 이상 확장이 불가능하므로 offset 영역이나 overflow 영역을 이용하여 순차적으로 노드번호를 할당한다. 이 때 잔여 오프셋 수(Noffset)가 필요한 노드 수보다 충분하면 offset 영역을 이용하여 오프셋 단위로 노드번호를 할당하고(Offset_Area_Numbering) 작으면 overflow 영역을 이용하여 버켓단위로 노드번호를 할당하게 된다(Overflow_Area_Numbering).

4. 성능 분석 및 평가

이번 장에서는 본 연구에서 제안한 방식(ENN)과 확장 프리오더(EP)[3] 기법을 비교하기 위해 두가지 삽입유형, 즉 다른장소 반복삽입 유형과 동일장소 반복삽입 유형으로 나누어 디스크 저장공간의 소요량과 삽입에 따른 디스크 페이지 갱신비용에 대한 성능평가를 실시하였다. 다른장소 반복삽입 유형은 매 삽입마다 다른 장소에 삽입되는 경우이고 동일장소 반복삽입 유형은 매 삽입마다 직전 삽입장소와 동일한 지점에 삽입되는 경우이다. 성능평가에 사용된 변수는 표 1과 같다.

4.1. 저장공간 비용

다른장소 반복삽입 유형에서 k개 노드를 총 I회 삽입할 때 노드번호들에 대한 누적 저장공간 크기의 비용 모델은 식(1), 식(2)와 같다.

$$EP = ((n+s)*2+\alpha)*N+((n+s)*2+\alpha)*k*I \quad (1)$$

$$ENN=\sum_{i=1}^{N}((n_i+1)*2+\alpha)+\sum_{p=1}^{I}\sum_{j=1}^{k}((m_p+1+s_j)*2+\alpha) \quad (2)$$

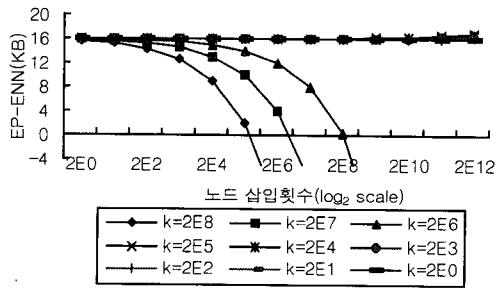
다른장소 반복삽입에서 ENN은 버켓번호만 사용되며 이때 m_p값은 n_i 값들 중의 하나이므로 m_p 대표값은 n′로 간주할 수 있다. 따라서 식(2)는 식(3)과 같이 계산된다.

$$ENN=((n'+1)*2+\alpha)*N+((n'+1+s')*2+\alpha)*k*I \quad (3)$$

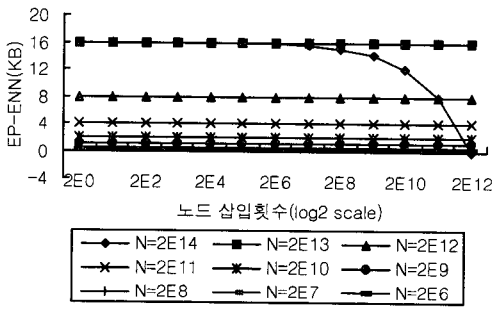
표 1. 성능평가를 위한 변수

항목	내 용	항목	내 용
n	초기 XML 문서의 노드번호 순차 할당을 위해 부여된 바이트 수	N	초기 XML 문서내 노드 개수
s	초기 노드번호 순차 할당시 향후 삽입을 위해 부여된 바이트 수	k	1회 삽입노드 개수
m_p	p회째 삽입시 삽입지점 노드의 버켓번호 바이트 수	I	총 삽입횟수
s_j	ENN방식에서 1회 삽입시 j번째 노드의 sparse영역 바이트 수	n′	n_i의 대표값(상수)
B	버켓내 최대 삽입가능한 노드 개수(버켓크기)	s′	s_j의 대표값(상수)
C1r	1페이지 read를 위해 필요한 디스크 접근 횟수	β	재조정 노드수
C1w	1페이지 write를 위해 필요한 디스크 접근 횟수	T	1페이지 바이트수
Br_p	버켓내 존재하는 노드들의 read 페이지 수	S	ENN방식에서 확장 허용 횟수
Bw_p	버켓내 존재하는 노드들의 재조정을 위한 write 페이지 수		
α	하나의 튜플에서 노드번호 한쌍을 제외한 나머지 필드의 평균바이트 수		
n_i	ENN방식에서 초기 문서트리에 할당된 i번째 노드번호의 버켓번호 길이		
H_p, H'_p	EP, ENN 각 방식에서 p회째 삽입시 삽입점 노드검색을 위해 참조한 페이지 수		
J_p, J'_p	EP, ENN 각 방식에서 p회째 삽입시 삽입점 노드의 부모노드 검색을 위해 참조한 페이지 수		
$K1_p$	EP방식에서 p회째 삽입시 여유공간내 즉 노드조정없이 노드를 삽입한 페이지 수		
$K1'_p$	ENN방식에서 p회째 삽입시 sparse 영역만을 사용하여 노드를 삽입한 페이지 수		
$K2_p$	EP방식에서 p회째 삽입시 기존 노드조정을 통해 확보한 여유공간내 노드를 삽입한 페이지 수		
$K2'_p$	ENN방식에서 p회째 삽입시 sparse 영역을 초과, 노드를 삽입한 페이지 수		
R_p	EP방식에서 여유공간을 확보하기 위하여 순차적으로 참조한 페이지 수		
U_p	EP방식에서 기존노드에 대해 재조정을 수행한 페이지 수		
Ar_p	EP방식에서 여유공간 확보를 위해 부모(조상)노드를 참조한 페이지 수		
Aw_p	EP방식에서 기존노드 재조정을 위해 부모(조상)노드를 갱신한 페이지 수		

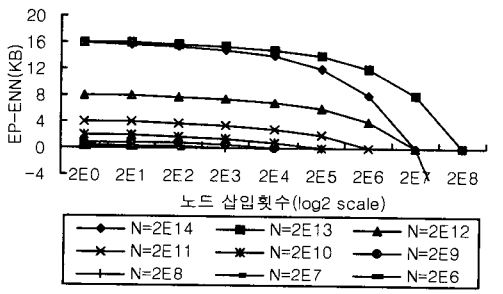
그림 6은 n=2, s=2, S=2일 때 초기 문서의 크기 N의 값과 1회 삽입노드 개수 k값에 따른 I회 다른 장소 반복삽입시 두 기법의 누적 저장공간의 차이 'EP - ENN' 값을 보여주고 있다. 그림 6(a)에서 k $\leq 2^5$일 때는 횟수에 관계없이 항상 ENN이 더 우수한 결과를 보이나 k > 2^5일 때는 총 삽입노드 수 k * I가 대략 N범위내의 삽입횟수에 대해서 ENN이 우수함을 알 수 있다. 그림 6(b), (c)는 N값이 클수록 초기 저장공간 차이가 더 커짐을 보이고 있다. 한편 동일 장소에 k개 노드를 총 I회 반복 삽입했을 때 EP의 소요 저장공간은 식(1)과 동일하고 ENN의 저장공간 비용은 식(4)와 식(6)으로 나누어 계산할 수 있다.



(a) N= 2^{13}, k= $2^0 \sim 2^8$



(b) $k=2^2$, $N=2^6 \sim 2^{14}$



(c) $k=2^6$, $N=2^6 \sim 2^{14}$

그림 6. n=2, s=2, S=2 일 때 I회 다른 장소 반복 삽입시 저장공간 차이

(I $\leq$S or (I>S and $\lfloor B/k \rfloor = 0$) 일 때)

$$ENN = \sum_{i=1}^{N}((n_i + 1) * 2+\alpha)+\sum_{p=1}^{I}\sum_{j=1}^{k}((m_p +1 + p-1 + s_j) * 2+\alpha) \quad (4)$$

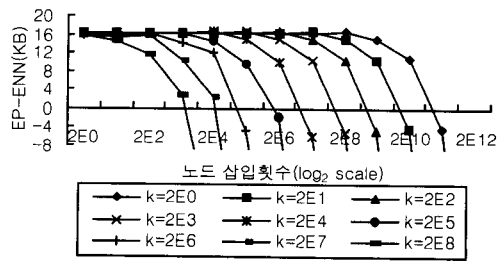
여기서 $m_1 = m_2 = \ldots = m_I$이고 최악의 삽입지점은 n_i의 i=N일 때이므로 $m_p = n_N$ 이다. 따라서 식(4)는 식(5)와 같이 계산된다.

$$ENN = ((n'+1)*2+\alpha)*N+((n_N+(I+1)/2+s')*2+\alpha) * k * I \quad (5)$$

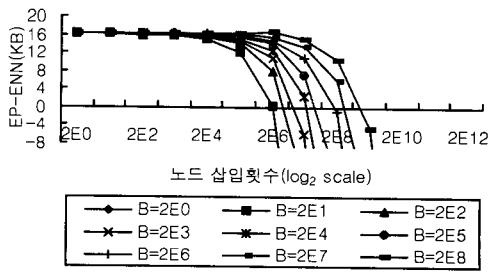
(I > S and $\lfloor B/k \rfloor \neq 0$ 일 때)

ENN은 먼저 sparse 영역을 모두 사용하고 난 후 offset 영역과 overflow 영역을 사용한다.

$$ENN=\sum_{i=1}^{N}((n_i+1)*2+\alpha)+\sum_{p=1}^{S}\sum_{j=1}^{k}((m_p+1+p-1+s_j)*2+\alpha)+\sum_{p=1}^{I-S}\sum_{j=1}^{k}((m_p+1)*2+\alpha)+\sum_{p=1}^{W}(\sum_{r=1}^{V}(S+p-1+s_r)*2) \quad (V=(\lfloor B/k \rfloor +1)*k,\ W=\lfloor (I-S)/(\lfloor B/k \rfloor +1) \rfloor) \quad (6)$$



(a) $k=2^0 \sim 2^8$, $B=2^8$, $N=2^{13}$



(b) $B=2^0 \sim 2^8$, $k=2^2$, $N=2^{13}$

그림 7. n=2, s=2, S=2 일 때 동일장소 I 번 반복 삽입시 저장공간 차이

$s_1 \sim s_V$의 대표값을 s″라 하면 식(6)은 식(7)과 같이 구해진다.

$$ENN=((n'+1)*2+a)*N+((n_N+(S+1)/2+s')*2+a)*k*S+((n_N+1)*2+a)*k*(I-S)+(S+(W+1)/2-1+s'') * 2 * V * W \quad (7)$$

그림 7(a)에서 총 삽입 노드수 k * I가 초기 문서의 노드 수 N에 가까워질 때 ENN의 성능은 떨어지나 작은 값 k(1회 삽입노드 갯수)의 반복삽입시 길어진 노드번호로 인해 증가된 공간에도 불구하고 대략 $k*I < 2^{11}(=N/2^2)$인 범위내에서 우수한 성능을 보이고 있다. k가 클수록 k * I 값은 조금 더 늘어난다. 그림 7(b)는 버켓의 크기가 클수록 더 많은 삽입횟수에 대해 우수한 성능을 나타내고 있다.

4.2 갱신 비용

이미 저장된 XML 문서에 대해 노드 삽입이 발생할 때의 갱신비용을 비교하기 위해 노드삽입에 따른 디스크 접근 횟수만을 비교항목으로 선정하였다. 확장 프리오더(EP)와 제안방식(ENN) 모두 노드 삽입량이 여유공간보다 작은 경우는 각 기법의 정의에 의해 재조정되는 노드는 없다. 그러나 여유공간을 초과하는 노드 삽입인 경우 EP 방식은 인접 노드번호에 대해 필요한 공간을 확보할 때까지 순회하고 목표 노드번호 사이에 있는 모든 노드번호를 재조정한다.

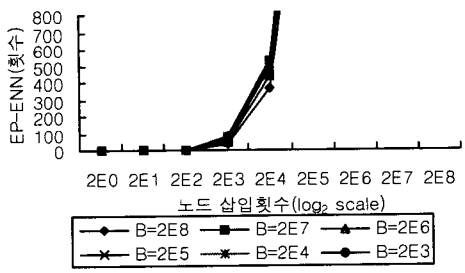
한편 ENN 방식에서 여유공간(sparse 영역)을 초과하는 노드 삽입은 offset 영역과 overflow 영역을 사용하게 된다. 이 때 overflow 영역을 사용하여 부여된 노드번호는 노드번호 길이가 확장되기 때문에 삽입에 따른 재조정은 발생하지 않는다. ENN 방식의 노드번호 재조정은 여유공간을 초과하는 삽입이 발생할 때 offset 영역을 사용하여 부여된 노드들에 대해서만 발생된다. 따라서 최악의 경우 재조정 노드수는 버켓내 최대로 존재할 수 있는 노드번호 수 즉 버켓크기가 된다. 다른장소 반복삽입 유형인 경우에 EP와 ENN 모두 재조정노드가 없기 때문에 동일장소 반복삽입 유형에 대해서만 성능 평가를 실시하였다.

동일장소 반복삽입 유형에서 k개 노드를 I회 삽입했을 때 노드번호들에 대한 디스크 접근 비용은 식(8), 식(10)과 같다.

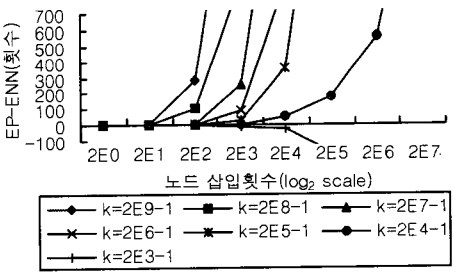
$$EP = \sum_{p=1}^{I} (H_p*C1r+J_p*C1r+K1_p*C1w+R_p*C1r+Ar_p*C1r+U_p*(C1r+C1w)+Aw_p*(C1r+C1w)+K2_p*C1w) \quad (8)$$

식(8)의 계산을 간단히 하기 위해 A_{rp}와 A_{wp}가 0이 되도록 트리형태가 아닌 단말노드 형태의 노드들만 삽입이 이루어진다고 가정하며 C1r=1, C1w=1로 설정한다. 그리고 $(k+1)^x = 2^{8*s-1}$ (x는 정수)가 되도록 1회 삽입노드 수 k를 설정하고 매 회 삽입시 삽입노드들은 재조정 노드 수를 최소화하기 위해 여유공간을 균등하게 분할한다고 가정한다. 따라서 노드 삽입은 기 확보된 여유공간이나 추가 확보된 여유공간 중 어느 한쪽에만 발생하며 재조정 노드수는 '여유공간 확보를 위해 검색한 노드수 - 1'이 된다. 따라서 식(8)은 식(9)와 같이 계산된다.

$$EP = \sum_{p=1}^{I} (H_p + J_p + K_p + 3 * U_p) \quad (9)$$



(a) $k=2^5-1$, $B=2^3 \sim 2^8$



(b) $k=2^3-1 \sim 2^9-1$, $B=2^8$

그림 8. n=2, s=2, S=2, $N=2^{13}$일 때 동일장소 I 번 반복삽입시 디스크 접근횟수 차이

한편, 제안방식의 갱신비용은 정의에 의해 식(10)과 같이 계산될 수 있다.

$$ENN = \sum_{p=1}^{I} (H'_p*C1r+J'_p*C1r+K1'_p*C1w+ Br_p*C1r+Bw_p*C1w+K2'_p*C1w) \quad (10)$$

여기서, $p \le S$일 때는 $K1'_p=K'_p$, $K2'_p=0$이고 $p>S$일 때는 $K1'_p=0$, $K2'_p=K'_p$가 된다. 식(10)은 식(11)과 같이 간략화 된다.

$$ENN = \sum_{p=1}^{I} (H'_p +J'_p +K'_p +Br_p +Bw_p) \quad (11)$$

식(9)와 식(11)에서 $H_p = H'_p$, $J_p = J'_p$라 가정하면 EP와 ENN 방식의 저장공간 차를 구하는 식은 식

표 2. EP-ENN 식 요소 값

요소 항목	요소 값
K_p	$\lceil ((n+s) * 2 +\alpha) * k /T \rceil$
U_p	$(p \le I')\ U_p =0$, $(p > I')\ U_p = \lceil ((n+s) * 2 +\alpha) * \beta /T \rceil$, β: 재조정노드수(표3)
K'_p	$((p \le S)$ or $(p > S$ and $\lfloor B/k \rfloor = 0))\ \lceil ((n_N+p+s') * 2 +\alpha) * k /T \rceil$, $((p > S$ and $\lfloor B/k \rfloor \ne 0)$ and $R\ne0)\ \lceil ((n_N+1) * 2 +\alpha) * k /T \rceil$, $((p > S$ and $\lfloor B/k \rfloor \ne 0)$ and $R=0)\ \lceil ((n_N+S+Q+ s') * 2 +\alpha) * k /T \rceil$
Br_p	$((p \le S)$ or $(p > S$ and $\lfloor B/k \rfloor = 0))\ 0$, $((p > S$ and $\lfloor B/k \rfloor \ne 0)$ and $R\ne0)\ \lceil ((n_N+1) * 2 +\alpha) * (R-1) * k /T \rceil$, $((p > S$ and $\lfloor B/k \rfloor \ne 0)$ and $R=0)\ \lceil ((n_N+1) * 2 +\alpha) * \lfloor B/k \rfloor * k /T \rceil$
Bw_p	$((p \le S)$ or $(p > S$ and $\lfloor B/k \rfloor = 0))\ 0$, $((p > S$ and $\lfloor B/k \rfloor \ne 0)$ and $R\ne0)\ \lceil ((n_N+1) * 2 +\alpha) * (R-1) * k /T \rceil$, $((p > S$ and $\lfloor B/k \rfloor \ne 0)$ and $R=0)\ \lceil ((n_N+ S+Q + s'') * 2 + \alpha) * \lfloor B/k \rfloor * k /T \rceil$
	Q= (p-S)/($\lfloor B/k \rfloor$ +1)의 몫, R= (p-S)/($\lfloor B/k \rfloor$ +1)의 나머지, $I' = \lfloor (\log_{(k-1)} 2^{8*s-1}) \rfloor$ $n_N = n$, $s' \le \lceil (\log_2 k + 3) /8 \rceil$, $s'' \le \lceil (\log_2 k * (\lfloor B/k \rfloor +1) + 3) /8 \rceil$

표 3. p값에 따른 재조정 노드 갯수

t	i	조건	재조정 노드수(β)	e_a
t=0	i=0	r=k+1	$(q -i(k+1)+1)(k+1)^2$	1
		r≠k+1	r(k+1)	
	$1 \le i \le k$	r=i-1 ∧ $q\ne q_t$	$(q -i(k+1))(k+1)^2$	0
		r=i-1 ∧ $q=q_t$	$Q'(q -i(k+1))(k+1)^2$	
		r≠i-1 ∧ r < (i-1)	(k+2+r-i)(k+1)	
		r≠i-1 ∧ r > (i-1)	(r-i)(k+1)	
t=1	i=k+1	r=k+1	$(q -i(k+1))(k+1)^2$	0
		r≠k+1	r(k+1)	
	$k+2 \le i \le 2k+1$	r=i'-1 ∧ $q\ne q_t$	$(q -i(k+1) -1)(k+1)^2$	0
		r=i'-1 ∧ $q=q_t$ ∧ i ≠ 2k+1	$(Q'-(k+1))(q -i(k+1) -1)(k+1)^2$	
		r=i'-1 ∧ $q=q_t$ ∧ i = 2k+1	$Q'(q -i(k+1) -1)(k+1)^2$	
		r≠i'-1 ∧ r < (i'-1)	(k+2+r-i')(k+1)	
		r≠i'-1 ∧ r > (i'-1)	(r-i')(k+1)	
t=m (2≤m<k+2)	i=m(k+1)	r=k+1 ∧ k < m-2	$(q -i(k+1) -m+1)(k+1)^2$	(k < m-2) 0, (k ≥ m-2) 1
		r=k+1 ∧ k ≥ m-2	$(q -i(k+1) -m+2)(k+1)^2$	
		r≠k+1	r(k+1)	
	mk+m+1 ≤ i ≤ (m+1)k +m	r=i'-1 ∧ $q\ne q_t$	$(q -i(k+1) -m+1)(k+1)^2$	(mk+m+1 ≤ i <m(k+2)) 0, (m(k+2)≤ i ≤ (m+1)k +m) 1
		r=i'-1 ∧ $q=q_t$ ∧ i ≠ (m+1)k+m	$(Q'-m(k+1))(q -i(k+1) -m+1)(k+1)^2$	
		r=i'-1 ∧ $q=q_t$ ∧ i = (m+1)k+m	$Q'(q -i(k+1) -m+1)(k+1)^2$	
		r≠i'-1 ∧ r < (i'-1)	(k+2+r-i')(k+1)	
		r≠i'-1 ∧ r > (i'-1)	(r-i')(k+1)	

p'= p-I', q=p'/(k+2)의 몫, r=p'/(k+2)의 나머지, t=i/(k+1)의 몫,
Q'=p'/((k+1)*(k+2))의 몫, R'=p'/((k+1)*(k+2))의 나머지
i : $Q'(k+1)(k+2) + Q' \le p' < (Q'+1)(k+1)(k+2)$ 일 때 i= Q',
$Q'(k+1)(k+2) \le p' \le Q'(k+1)(k+2)+Q'-1$ 일 때 i= Q'-1
i': i/(k+1)의 나머지, q' = i/(k+2)의 몫, $q_t = q_e - e_a = (i+1)(k+1)+q' - e_a$

(12)와 같이 구해진다.

$$EP - ENN = \sum_{p=1}^{I} ((K_p + 3 * U_p) - (K'_p + Br_p + Bw_p)) \quad (12)$$

식(12)의 K_p, U_p, K'_p, Br_p, Bw_p의 값은 표 2, 표 3를 이용하여 계산된다.

그림 8은 n=2, s=2, S=2, $k=2^5-1$, $B=2^8$, $N=2^{13}$, α =56, T=1024 일 때 총 I회 동일장소 반복삽입시 EP와 ENN 방식의 디스크 접근회수의 차이를 보여주고 있다. 그림 8(a)는 노드 삽입횟수가 증가할수록 제안 방식의 성능이 더욱 더 우수함을 보이고 있으며 또한 $\lfloor B/k \rfloor$ 값이 작을수록 더 나은 성능을 나타낸다. 그림 8(b)에서 일정한 B의 값에 대해 k의 값이 클수록 디스크 누적접근회수 차이는 급증하여 제안방식이 우수한 성능을 갖고 있음을 알 수 있으며 작은 값의 k에 대해서는 저조한 성능을 보이나 실험결과 작은 값의 k에 대해서 B의 값 조정을 통해 우수한 결과를 얻을 수 있었으며 k값이 우수한 결과를 나타내는 개략적인 B의 범위는 $k=2^2-1$ 인 경우 2^4, $k=2^3-1$인 경우 2^6, $k=2^4-1$ 이상인 경우 2^8까지가 해당된다. 그리고 n의 값과는 거의 무관함을 보이고 있다.

결과적으로 논문에서 제안한 ENN 방식은 저장공간 측면에서 대략 $k * I < N$인 삽입횟수에 대해 우수한 결과를 보이고 있으며 동일장소 반복삽입인 경우 길어진 노드번호로 인한 검색비용이 증가함에도 불구하고 초기 문서 노드 수를 초과하는 삽입횟수에 대해 우수한 갱신 성능을 보이고 있다. 작은 값의 k에 대해 많은 횟수의 반복 삽입이 일어날 때 증가하는 검색비용이 예상보다 크지 않음을 실험을 통해 알 수 있었다. B의 값을 k에 대한 유효 범위내의 $B > k$인 적절한 값으로 설정하면 노드번호 길이는 R=0인 경우만 증가하며 $\lfloor B/k \rfloor$+1 회마다 2 * k 바이트 씩 증가한다. 따라서 $T=2^{10}$, k=3 일 경우 1024/3(≒ 342) 회마다 디스크 검색이 한번 더 일어난다.

제안방식의 우수한 성능중의 하나는 반복삽입 횟수에 따른 노드삽입 비용 편차가 작다는 것이다. EP의 노드삽입비용은 재조정 노드 수에 따라 좌우되는데 재조정 노드 수의 편차가 ENN에 비해 상당히 크다. $k=2^4-1$일 때 $p=2^4+3$(즉 $p'=2^4$)번째 삽입시 EP의 재조정 노드수 β는 $(k+1)^2 = 2^8$이고 $p=2^4*(2^4+1)+3$인 경우 $\beta=(2^4)^3$이 된다. 반면 ENN의 재조정 노드수는 항상 설정된 버켓크기 값(최대 설정치:2^8)이하로 제한된다. 따라서 동일장소 반복삽입시 특정 삽입횟수 시점에 따라 갱신 속도가 지연되는 현상을 방지할 수 있다.

5. 결론 및 향후 연구과제

XML 문서의 갱신은 이미 저장된 노드번호들에 대해 재조정 작업을 야기시켜 XML 문서관리 시스템의 성능을 저하시키는 요인이 되고 있다. 본 논문에서는 XML 문서 갱신으로 인한 기존 노드번호들의 재조정 범위를 최소화하기 위해 새로운 노드 넘버링 방식인 ENN(Extensible Node Numbering)을 제안하였다. ENN 방법은 가변길이 문자열을 사용하여 삽입시점에서 노드들에 대해 필요한 만큼의 버켓번호들을 동적으로 생성, 부여하며 향후 삽입이 발생될 때 오프셋을 순차 할당하는 방식을 채택함으로써 해당 버켓내 노드들만 재조정되도록 하였다. 저장공간 소요량은 최악의 경우 즉 동일장소 반복삽입 유형인 경우 초기 문서의 노드 수 대비 작은 양의 노드를 삽입할 때는 ENN이 우수하나 초기문서의 노드 수에 근접할수록 저장공간이 늘어난다. 그러나 디스크 접근횟수는 길어진 노드번호로 인해 증가된 페이지 수에도 불구하고 초기문서 노드 수를 초과하는 삽입에 대해서도 EP에 비해 우수한 성능을 갖는다는 것을 실험을 통해 알 수 있었다. 그리고 초기 문서 크기 이상의 삽입이 발생할 때 문서 재구성을 고려하면 이 결과는 더욱 더 유용한 결과라 평가되며 특히 일정한 노드 수를 동일장소에 반복 삽입할 경우 삽입시점에 관계없이 삽입비용 편차를 버켓크기 이하로 감소시킨 결과는 매우 의미가 있다고 평가된다.

그리고 본 실험에서 계산된 실험치는 노드삽입에 따른 조상(부모) 노드의 갱신을 고려하지 않았으나 실제상황에서는 조상(부모) 노드의 갱신이 발생한다. 따라서 EP의 갱신 비용은 실험에서 구해진 갱신 비용 이상이 소요되어 제안방식은 더욱 더 좋은 성능을 나타낼 것이다.

향후 연구로 동일 장소에 많은 횟수의 반복 삽입 및 삭제 연산 후 길어진 노드번호의 효율적인 처리를 위해 버켓번호 재사용 방법에 대한 추가 연구를 수행할 예정이다.

참 고 문 헌

[1] Paul F. Dietz, "Maintaining order in a linked list," *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*, pp. 122-127, 1982.

[2] 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," *Proceedings of IEEE ICDE*, pp.141-152, 2002.

[3] Q. Li and B. Moon, "Indexing and Querying XML Data for Regular Path Expressions," *Proceedings of VLDB*, pp.361-370, 2001.

[4] J. Shanmugasundaram, K. Tufte, C. Zhang, G. He, D. J. DeWitt, and J. F. Naughtton, "Relational Databases for Querying XML Documents: Limitations and Opportunities," *Proceedings of VLDB*, pp302-304, 1999.

[5] M. Yoshikawa et al., "XRel: A Path-Based Approach to Storage and Retrieval of XML Documents Using Relational Databases," *ACM Trans. on Internet Technology*, Vol. 1, No. 1, pp 110-141, 2001.

[6] N. Bruno, N. Koudas, and D. Srivastava, "Holistic Twig Joins: Optimal XML Pattern Matching," *Proceedings of ACM SIGMOD*, pp.310-321, 2002.

[7] S. Chien, Z. Vagena, D. Zhang, V. Tsotras, and C. Zaniolo, "Efficient Structual Joins on Indexed XML Documents," *Proceedings of VLDB*, pp 263-274, 2002.

[8] J. Shanmugasundaram, E. Shekita, R. Barr, M. Carey, B. Lindsay, H. Pirahesh, and B. Reinwald, "Efficiently Publishing Relational Data as XML Documents," *Proceedings of VLDB*, pp. 65-76, 2000.

[9] D. Florescu and D. Kossman, "Storing and Querying XML Data using an RDBMS," *IEEE Data Engineering Bulletin*, Vol. 22, No. 3, pp 27-34, 1999.

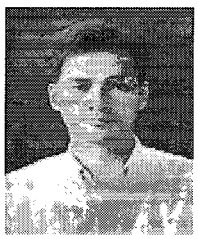
[10] C. Zhang et al., "On Supporting Containment Queries in Relational Databases Management Systems," *Proceedings of ACM SIGMOD*, pp. 425-436, 2001.

[11] I. Tatarinov, "Stroring and Querying Ordered XML Using a Relational Database System," *Proceedings of ACM SIGMOD*, pp. 204-215, 2002.

박 충 희

1989년 인하대학교 전산학과 졸업(이학사)
1994년 인하대학교 대학원 전산공학과 졸업(공학석사)
2002년~현재 제주대학교 컴퓨터공학과 박사과정

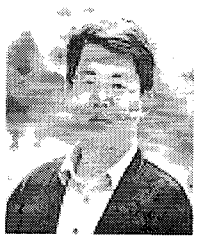
관심 분야 : XML 데이타베이스, Web database, GIS



구 흥 서

1985년 인하대학교 전산학과 졸업(이학사)
1989년 인하대학교 대학원 전산학과 졸업(이학석사)
1993년 인하대학교 대학원 전산학과 졸업(이학박사)
1994년~현재 청주대학교 정보기술공학부 부교수

관심 분야 : XML 데이타베이스, 컨텐트관리시스템(CMS), 기업정보포탈(EIP)



이 상 준

1984년 중앙대학교 컴퓨터 공학과 졸업(공학사)
1989년 중앙대학교 대학원 컴퓨터 공학과 졸업(공학석사)
1992년 중앙대학교 대학원 컴퓨터 공학과 졸업(공학박사)
1993년~현재 제주대학교 컴퓨터 공학과 부교수

관심 분야 : Heuristic Algorithm, Web Database, Agent, XML응용