XML 데이타를 위한 효율적인 디스크 블록 할당 방법

(An Efficient Disk Block Allocation Method for XML Data)

김 정 훈 † 손 진 현 †† 정 연 돈 ††† 김 명 호 ††††
(Jung Hoon Kim) (Jin Hyun Son) (Yon Dohn Chung) (Myoung Ho Kim)

요 약 XML과 같은 준구조적 데이타가 많이 사용됨에 따라 이를 효과적으로 저장하고 관리하는 것이 중요해지고 있다. XML 데이타는 트리 형태로 모델링이 가능하며, 기본적으로 질의 처리는 트리를 탐색하는 방식으로 이루어진다. 본 논문에서는 XML 데이타를 디스크 블록에 저장하는 알고리즘을 제안한다. 제안하는 알고리즘은 트리의 각 노드마다 아래쪽에서 위쪽으로 숫자를 할당하며 그 숫자를 이용하여 디스크 블록에 노드들을 매핑한다. 제안하는 알고리즘은 접근 패턴 정보를 필요로 하지 않으며 어떠한 접근 패턴에 대해서도 좋은 성능을 보인다. 제안하는 방법의 몇가지 특성을 증명하고, 실험을 통해서 성능을 평가한다.

키워드 : XML, 준구조적 데이타, 디스크 저장 방법

Abstract With the recent proliferation of the use of semi-structured data such as XML, it becomes more important to efficiently store and manage the semi-structured data. The XML data can be logically modelled as a rooted tree e.g., the DOM tree. In order to process a query on the XML data, we traverse the tree structure. In this paper we present an algorithm that places the XML data to disk blocks. The proposed algorithm assigns a number to each node of the tree in a bottom-up fashion. Then, the nodes are allocated to disk blocks using the assigned number. The proposed algorithm does not need access pattern information, and provides good performance for any access pattern. The characteristics of the proposed method are presented with analysis. Through experiments, we evaluate the performance of the proposed method.

Key words : XML, Semistructured data, Disk storage method

1. 서 론

XML은 인터넷 환경에서 데이타 표현과 교환의 표준으로 자리잡아 가고 있으며 웹 기반 애플리케이션 분야에서 널리 적용되고 있으며 이에 따라 관리해야 하는 데이타의 양도 크게 증가하고 있다[1]. 따라서 이에 대한 효율적인 관리 방법이 필요하다. XML 등의 준구조적 데이타는 트리 형태로 모델링할 수 있다. 그림 1에 XML에서 사용되는 DOM 트리의 예를 보이고 있다[2]. 그림 1의 각 노드는 데이타 스키마나 데이타 자신을 의미하며, 노드 간을 연결하는 화살표는 DOM 트리에서의 접근 경로를 의미한다. 예를 들어, 'Pusan' 노드는 루트 노드인 'invoice'에서 'buyer', 'address' 노드를 통하여 접근할 수 있다.

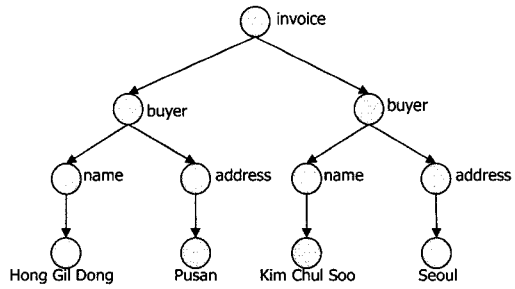


그림 1 XML 데이타

· 본 연구는 정보통신부 및 정보통신연구진흥원의 대학 IT연구센터 육성 지원 사업(IITA-2006-C1090-0603-0031)의 연구결과로 수행되었음

† 비 회 원 : 한국과학기술원 전산학과
kimjh@dbserver.kaist.ac.kr
†† 종신회원 : 한양대학교 컴퓨터공학과 교수
jhson@cse.hanyang.ac.kr
††† 종신회원 : 고려대학교 컴퓨터 · 통신공학부
ydchung@korea.ac.kr
†††† 종신회원 : 한국과학기술원 전산학과 교수
mhkim@dbserver.kaist.ac.kr

논문접수 : 2004년 2월 4일
심사완료 : 2007년 5월 25일

XML과 같은 준구조적 데이타에 대한 질의는 데이타 자체와 데이타 구조에 대한 조건을 같이 포함하는 경로 질의의 형식을 가진다[3-5]. 그리고, 경로 질의의 처리는 경로 정보를 사용하여 DOM 트리를 탐색하고 조건을 만족하는 노드들을 찾는 방식으로 이루어진다.

관리해야 하는 XML 데이타의 양이 크게 증가함에 따라 이를 디스크 등의 보조 기억장치에 저장하여 관리하는 것이 필요해졌다. XML 데이타를 디스크에 저장하는 방식으로는 크게 두 가지를 생각할 수 있다. 하나는 데이타를 일반적인 텍스트 파일 형식으로 저장하는 것이다. 이 경우 질의 처리를 하기 위해서는 XML 파일 데이타를 메모리로 읽어들여 파싱하는 과정이 필요하다. 다른 방법은 XML 데이타를 저장할 때 계층 구조 정보를 같이 저장하는 것이다. 이 경우에는 XML 문서를 반복적으로 파싱할 필요가 없으며 질의가 주어졌을 때 관련된 디스크 블록만을 읽고 처리하는 것도 가능하다. 본 논문에서는 DOM 구조를 유지하면서 XML 문서를 디스크에 저장하는 방법을 고려한다.

XML 문서를 저장하는 방법으로 지금까지 여러가지가 제안되어 왔다. 가장 기본적인 것으로 앞에서 이야기했듯이 XML 문서를 일반 텍스트 파일 형식으로 저장하는 방법이 있다.

다른 방법으로는 XML 문서의 스키마 정보를 이용하여 문서를 분할하여 저장하는 방식이 있다. Xindice의 경우 스키마 정보에서 얻을 수 있는 엘리먼트 명이나 애트리뷰트 명 등을 Key로 하여 B-트리를 구성하고 분할된 문서데이타가 저장된 레코드를 가리키도록 하고 있다[6]. 관계형 데이타베이스 분야에서도 이 방식에 대한 연구가 많이 이루어졌다[7-9]. 이들의 기본적인 아이디어는 문서의 스키마 정보로부터 데이타베이스 테이블을 생성하고 각 테이블에 문서 데이타를 분할하여 저장하는 것이다. 이 경우에는 문서 처리를 위한 파싱은 불필요하지만 전체 문서를 구성하는 작업의 경우 각 노드들이 저장된 레코드를 모두 읽어야 하므로 텍스트 파일 형식으로 저장한 경우보다 좋지 않은 성능을 보일 수 있다. 그리고, 새로운 문서 타입이 추가되거나 기존의 문서 타입이 변경될 경우 데이타베이스의 테이블을 새로 생성하거나 변경해 주어야 한다.

W3C에서는 DOM을 직렬화하기 위한 표준 API를 제안하고 있다[10]. 여기서는 DOM을 문자열로 변환하여 저장하며, DFS에 기반한 형태로서 처리 성능에 대해서는 특별히 고려하지 않고 있다.

Xyleme에서 사용하는 XML 저장소인 NATIX에서는 문서를 저장할 때 문서의 구조를 유지하면서 처리 성능을 개선하는 방법을 제안하였다[11]. NATIX에서는 하나의 레코드에 여러 개의 노드를 저장하며 새로운 노드가 추가될 경우 레코드를 분할하는 알고리즘을 도입하였다. 이 방법은 문서가 새로 추가될 경우 문서의 노드들을 탐색하는 순서에 따라 레코드에 저장되는 형태가 달라지게 되며 트리가 어느 한쪽으로 치우친 형태로 구성되면 좋지 않은 성능을 보이게 된다. (이 기법은 Timber 시스템[12]에서도 사용되는 저장 방식이다.)

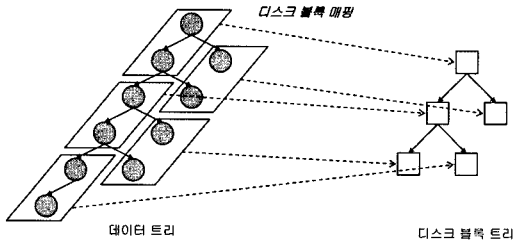
한편, OrientStore 시스템[13]에서는 스키마(Schema)를 통해 시맨틱 정보를 추출하여, 이를 XML 데이타의 저장에 활용하는 방식이 제안되었다. 스키마 상에서 반복되는 부분이 '*' 혹은 '+'로 표시되며, 이는 동일한 형태의 레코드 형태가 반복되는 경우에 많이 사용되기에, 이러한 부분들을 동일 디스크 블록에 할당하려는 휴리스틱을 사용할 수 있다. 하지만, 이 방법은 스키마 정보를 필요로 하고, 시맨틱 정보의 정확성에 따라 성능의 차이가 나타나며, 최적성 여부를 판단할 수 없다는 단점을 지닌다.

논문의 나머지 부분은 다음과 같이 구성된다: 2장에서는 이 문제에 대한 연구 동기를 언급하고, 3장에서는 본 논문에서 논의할 문제를 정의한다. 4장에서는 제안하는 디스크 블록 매핑 방법에 대하여 설명하고 5장에서 제안하는 방법의 특성을 분석한다. 6장에서는 실험을 통하여 제안하는 방법의 성능을 평가하고 7장에서는 결론을 서술한다.

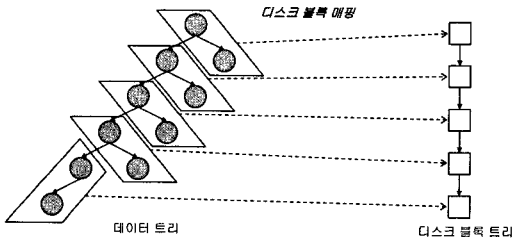
2. 연구 동기

XML 문서를 디스크 블록에 매핑하는 방식에 대해서 질의 처리 비용이나 공간 비용과 같은 평가 요소를 생각할 수 있다. 일반적으로 질의 처리 비용은 디스크 블록의 접근 회수로 측정되며 공간 비용은 XML 문서를 저장하기 위하여 필요한 디스크 블록의 개수로 측정한다. 최근에 보조 디스크 저장 장치의 비용이 상당히 낮아졌기 때문에 예전에 비해서 공간 비용에 대한 요구사항이 크지 않다. 따라서, 본 논문에서는 디스크에 저장된 XML 문서에 대한 질의 처리 비용의 개선에 중점을 둔다.

그림 2는 디스크 블록 매핑 방법에 따라 질의 처리 비용이 다르게 나타나는 것을 보이고 있다. XML DOM 트리 등과 같이 준구조적 데이타를 트리 형태로 표현한 것을 데이타 트리(DT)라 하고 이를 디스크 블록에 매핑한 결과가 트리 형태로 표현될 경우 이를 디스크 블록 트리(DBT)라 언급할 것이다. 그림 2(a)와 같이 디스크 블록 매핑을 할 경우 DT의 어떤 노드라도 최대 3개의 디스크 블록을 통하여 접근이 가능하다. 반면에 그림 2(b)의 경우에는 5개의 디스크 블록을 거쳐서 접근할 수 있는 경우도 있다. 본 논문에서는 최소의 높이를 갖는 디스크 블록 트리를 생성할 수 있는 효율적인 디스크 블록 매핑 방법을 제안한다.



(a) 디스크 블록 매핑 예 1



(b) 디스크 블록 매핑 예 2

그림 2 디스크 블록 매핑의 예

3. 문제 정의

이번 장에서는 몇 가지 용어를 정의하고 디스크 블록 매핑 문제를 서술한다.

정의 1. XML DOM 트리 등과 같이 준구조적 데이타를 트리형태로 모델링한 것을 데이타트리라 하며 T_{Data}로 표시한다.

T_{Data}의 각 노드 n_i에 대하여 3개의 파라미터를 정의한다.

T_i	n_i를 루트로 하는 부트리
$c(i)$	n_i로부터 T_i의 단말 노드를 접근하기 위하여 필요한 디스크 블록 접근 횟수 중최대값
S_i	n_i를 저장하기 위하여 필요한 공간의 크기
B	디스크 블록의 크기

일반적으로 디스크 블록은 하나 이상의 노드를 저장할 수 있다고 생각할 수 있다. 따라서 항상 $B \geq S_i$이 성립한다고 본다. $c(i)$는 부트리 T_i의 루트 노드 n_i에서 T_i 내의 노드들을 접근하기 위하여 필요한 디스크 블록 접근 회수 중 최대값을 의미한다. 본 논문에서는 임의의 노드 n_i에 대해서 $c(i)$를 최소화하는 매핑 방법을 제안한다.

정의 2. DOM 트리로 모델링된 XML 데이타에 대하여 디스크 블록 매핑 문제는 T_{Data}의 임의의 노드 n_i에 대하여 $c(i)$를 최소화하는 디스크 블록 매핑 방법을 찾는 것이다.

4. 디스크 블록 매핑 방법

본 논문에서 제안하는 방법은 트리 넘버링과 디스크 블록 매핑의 두 단계로 구성된다. 트리 넘버링 단계에서는 T_{Data}의 각 노드 n_i에 양의 정수(매핑 인디케이터)를 할당한다. 그리고나서 각 노드의 매핑 인디케이터 값을 이용하여 노드들을 디스크 블록으로 매핑한다. 4.1절과 4.2절에서 각 단계에 대하여 설명할 것이다.

4.1 트리 넘버링

트리 넘버링 단계에서는 T_{Data}의 각 노드 n_i에 매핑 인디케이터 $m_i(\geq 1)$를 할당한다. 이 과정은 트리의 아래쪽에서 위쪽으로 진행된다. 즉, 트리의 단말 노드부터 매핑 인디케이터 값이 결정되며 T_{Data}의 루트 노드의 매핑 인디케이터 값이 마지막으로 결정된다. 각 노드의 매핑 인디케이터 값을 결정하는 규칙은 다음과 같다.

규칙 1. 노드 n_i가 단말 노드일 경우: m_i 값은 1이 된다.

규칙 2. 노드 n_i가 내부 노드이며 n_i의 모든 자손 노드 n_j에 대하여 m_j값이 결정되어있는 경우:

(1) m_j값 중 가장 큰 값을 $MMI(n_i)$라 하자.

(2) $s_i+\sum s_j$ (n_j는 n_i의 자손 노드이며 $m_j = MMI(n_i)$를 SUM_i라 하자. 이 값과 디스크 블록 크기 B를 비교하여 $SUM_i \leq B$이면 m_i값으로 $MMI(n_i)$를 할당한다. 그리고, $SUM_i > B$이면 m_i값으로 $MMI(n_i)+1$을 할당한다.

규칙 1은 각 노드의 매핑 인디케이터 값을 결정하기 위한 초기 조건을 의미하며, 규칙 2에서는 n_i를 자손 노드들과 같은 디스크 블록에 할당할 수 있는지 여부를 판단하여 매핑 인디케이터 값을 결정한다.

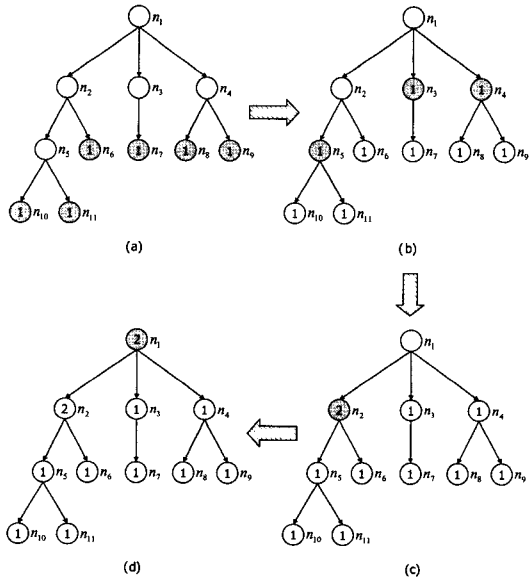


그림 3 트리 넘버링의 예

그림 3에서 디스크 블록 크기 B는 3이며 각 노드를 저장하기 위하여 필요한 공간의 크기는 모두 1이다. 즉 $s_i = 1(1 \le i \le 11)$이며 각 디스크 블록은 최대 3개의 노드를 저장할 수 있다. 그림 3에서 매핑 인디케이터 값은 원 안의 숫자로 표시하였다. 처음으로 모든 단말 노드에 매핑 인디케이터 값으로 1을 할당한다. 그리고나서 규칙 2에 따라 내부 노드들의 매핑 인디케이터 값을 결정해 간다. n_3에서 $MMI(n_3)$, SUM_3는 각각 1과 $2(=s_3+s_7 = 1+1)$이므로 매핑 인디케이터 값 m_3는 1이 된다(규칙 2의 $SUM_3(=2) \le B(=3)$인 경우). n_4의 경우 $MMI(n_4)$, SUM_4는 각각 1과 $3(=s_4+s_8+s_9=1+1+1 \le B(=3))$이므로 m_4는 1이 된다. n_5의 경우 $MMI(n_5)$, SUM_5가 각각 1과 $3(=s_4+s_8+s_9=1+1+1 \le B(=3))$이므로 m_5는 1이 된다. 그림 3(b)에서 지금까지의 넘버링 과정을 보이고 있다. n_2의 경우 $MMI(n_2)$, SUM_2가 각각 1과 $5(=s_2+s_5+s_6+s_{10}+s_{11} > B(=3))$이므로 m_2는 $2(MMI(n_2)+1)$가 된다. 이것이 그림 3(c)에 나타나있다. n_1의 경우 $MMI(n_1)=2$이고 $SUM_1=2(=s_1+s_2) \le B(=3)$이므로 m_1은 2가 된다. 이것이 그림 3(d)에 나타나있다.

트리 넘버링 알고리즘을 그림 4에 정리하였다.

```
1. Apply Rule 1 to the leaf nodes of T_Data.
2. WHILE(exist any node whose mapping indicator is not decided)
       Apply Rule 2
   /* All nodes' mapping indicators are decided. */
```

그림 4 트리 넘버링 알고리즘

4.2 디스크 블록 매핑

디스크 블록 매핑은 루트 노드에서 단말 노드 방향으로 진행되며 그 과정에서 앞의 트리 넘버링 단계에서 각 노드마다 할당한 매핑 인디케이터 값을 사용한다. 4.1절에서 언급한 바와 같이 한 노드와 그 노드의 자식 노드가 동일한 매핑 인디케이터 값을 가질 경우 하나의 디스크 블록에 할당되며, 이것이 디스크 블록 매핑의 기본적인 방법이다. 한 노드가 선택되면 그 노드와 동일한 매핑 인디케이터 값을 가진 자손 노드들을 하나의 디스크 블록에 할당한다. 그리고, T_{Data}에서 아직 매핑이 되지 않은 노드 중 가장 높은 위치에 있는 노드를 다음에 매핑할 노드로 선택한다. T_{Data}의 루트 노드에서 시작하며, 모든 노드가 디스크 블록에 할당될 때까지 이 과정을 반복한다. 그림 5에 디스크 블록 매핑 알고리즘을 정리하였다.

그림 6을 통하여 디스크 블록 매핑 알고리즘이 동작하는 예를 보일 것이다. 그림 6에서 T_{Data}는 15개의 노드로 구성되며(n_i, $1 \le i \le 15$), 매핑 인디케이터 m_i가 4.1절에서 설명한 방법에 따라 이미 결정되어 있다. 우선

```
1. WHILE(exist any node that is not allocated to a disk block)
   {
      1.1. Arbitrarily select one of the nodes that are in the highest level in
      T_Data and are not allocated to any disk block yet.
         /* Let the selected node be n_i */
      1.2. Find all descendant nodes of n_i that have the same mapping
      indicator as n_i.
      1.3. Allocate n_i and the descendant nodes found in step 1.2 to the same
      disk block.
   }
/* All nodes are allocated to disk blocks. */
/* Finally, a disk block tree can be constructed. */
```

그림 5 디스크 블록 매핑 알고리즘

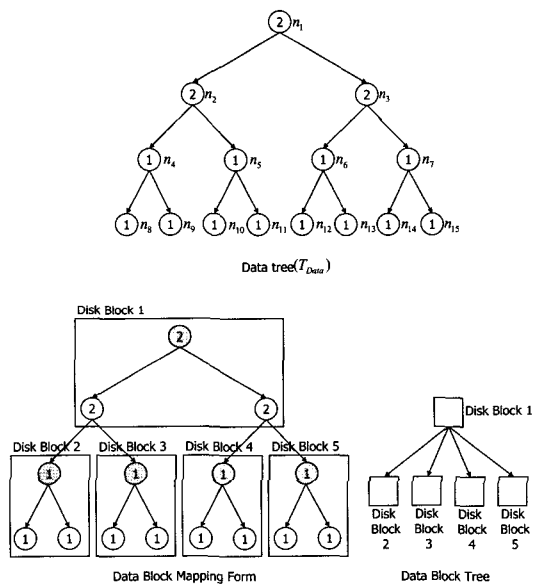


그림 6 디스크 블록 매핑의 예

T_{Data}에서 가장 높은 위치의 노드인 루트 노드 n_1이 선택된다. $m_1=2$이고 이와 동일한 매핑 인디케이터 값을 가진 노드는 n_2와 n_3이다. 따라서 n_1, n_2, n_3가 하나의 디스크 블록에 할당된다. 이제 다음으로 선택할 수 있는 노드는 n_4, n_5, n_6, n_7이다. 노드를 선택하는 순서는 최종 결과에 영향을 주지 않는다. 여기서는 n_5를 선택한다고 하자. $m_5=1$이므로 n_5, n_{10}, n_{11}이 하나의 디스크 블록에 할당된다. T_{Data}의 모든 노드가 디스크 블록에 할당될 때까지 이와 같은 과정을 반복하면 그림 6의 디스크 블록 트리와 같은 결과를 얻게 된다.

5. 분석

본 장에서는 4.1절에서 언급한 매핑 인디케이터의 특

성을 알아본다. 그림 6을 살펴보면 n_i의 매핑 인디케이터 값 m_i는 루트 노드가 n_i인 부트리 T_i에 속한 노드들을 탐색하기 위하여 필요한 디스크 블록 접근 회수 중 최대값이 된다는 것을 알 수 있다. 그림 6에서 부트리 T_2를 보면 부트리의 루트노드 n_2에서 T_2내의 노드들을 탐색하기 위하여 최대 2개의 디스크 블록 접근(디스크 블록 2와 3)이 필요하다는 것을 알 수 있으며, 이 값은 m_2와 같다. 3장에서 우리는 $c(i)$를 "부트리 T_i의 노드들을 접근하기 위한 디스크 블록 접근 회수 중 최대값"으로 정의하였으며, 디스크 블록 매핑 방법에 따라서 여러 가지 값을 가질 수 있다. 이제 각 노드의 매핑 인디케이터 값이 $c(i)$의 가능한 최소값이 된다는 것을 증명할 것이다. 이를 통하여 본 논문에서 제안하고자 하는 디스크 블록 매핑 방법이 임의의 노드 n_i에 대하여 $c(i)$를 최소화하는 방법이라는 것을 알 수 있다.

$c(i)$의 최소값을 $c(i)_{\min}$이라 하자.

관찰 1. n_j가 n_i의 자식 노드일 때 $c(i)_{\min} \geq c(j)_{\min}$이 성립한다.

관찰 2. n_i가 단말 노드일 때 $c(i)_{\min}$은 1이며 n_i가 내부 노드일 때 $c(i)_{\min}$은 $\max(c(j)_{\min})$ 또는 $\max(c(j)_{\min})+1$이 된다(n_j는 n_i의 자식 노드).

증명. n_i가 단말 노드일 때 $c(i)_{\min}$이 1인 것은 명백하다. n_i가 내부 노드일 때에는 n_i가 자식 노드 중 일부와 같은 디스크 블록에 할당되거나 전혀 다른 디스크 블록에 할당되는 두가지 경우를 생각할 수 있다. 첫번째 경우에는 $c(i)_{\min}$이 $\max(c(j)_{\min})$이 되며 두번째 경우에는 $c(i)_{\min}$이 $\max(c(j)_{\min})+1$이 된다.

관찰 3. 임의의 노드 n_i에 대하여 $m_i \geq c(i)_{\min}$이 항상 성립한다.

Proof) 본 논문에서 제안하는 방법에서는 각 노드의 $c(i)$값이 m_i가 된다. 따라서 위의 식이 성립하는 것은 명백하다.

정리 1. 임의의 노드 n_i에 대하여 $m_i = c(i)_{\min}$이 항상 성립한다.

Proof) n_i가 단말 노드일 경우 다음은 명백하다.

$$m_i = c(i)_{\min} = 1 \tag{1}$$

n_i가 내부 노드일 경우를 생각해 보자.

n_i의 임의의 자손 노드 n_j에 대하여 다음이 성립한다고 가정하자.

$$m_j = c(j)_{\min} \tag{2}$$

4.1절에서 설명한 규칙 2에 의하여 m_i는 $MMI(n_i)$ 또는 $MMI(n_i)+1$이 된다.

$m_i = MMI(n_i)$일 때 n_i의 자손 노드 중 $m_i = m_j$인 노드 n_j가 존재한다.

식 (2)에 의하여,

$$m_i = m_j = c(j)_{\min} \tag{3}$$

그리고, 관찰 1과 관찰 3에 의하여 다음 식이 성립한다.

$$m_i \geq c(i)_{\min} \geq c(j)_{\min} \tag{4}$$

식 (3)과 (4)에 의하여 다음 식이 성립한다.

$$m_i = c(i)_{\min} = c(j)_{\min} \tag{5}$$

$m_i = MMI(n_i)+1$인 경우에는 식 (2)와 관찰 1과 3에 의하여 $c(i)_{\min}$은 $MMI(n_i)$ 또는 $MMI(n_i)+1$이 된다. $c(i)_{\min} = MMI(n_i)+1$일 경우 m_i값은 $c(i)_{\min}$인 것은 명백하다. 그리고, $c(i)_{\min} = MMI(n_i)$라는 것은 n_i가 자손 노드 중 일부와 하나의 디스크 블록에 할당된다는 것을 의미한다. 이것은 $m_i = MMI(n_i)+1$이라는 것과 모순이 된다. 따라서 $c(i)_{\min}$은 항상 $MMI(n_i)+1$이 되며, 다음이 성립한다.

$$m_i = c(i)_{\min} \tag{6}$$

식 (1), (5), (6)에 의하여 정리가 성립한다는 것을 알 수 있다.

6. 성능 평가

이 장에서는 실험을 통해, 제안하는 방법의 성능을 평가한다. 메인메모리 1GB의 Dual Pentium-4 프로세서를 사용하는 리눅스 시스템 상에서 실험하였으며, 실험 환경 설정은 다음과 같다.

6.1 실험 설정

실험은 2가지 종류로 구성하였다. [실험 1]은 임의의 특정 노드를 접근하는데 필요한 디스크 블록의 평균 접근 회수를 구하는 것이며, [실험 2]는 XMark라는 벤치마크에서 제공하는 XQuery 질의를 처리하는데 필요한 최대 및 평균 디스크 블록 접근 회수를 측정하는 것이다.

[실험 1]에서 사용한 데이타 집합은 *Docbook*[13], *NITF*[14], 그리고 *XMark*[15]이며, [실험 2]에서는 *XMark* 데이타 집합을 사용하였다. 각 데이타 집합은 다양한 크기의 문서 100개로 구성되어 있으며, 평균 크기 5MB, 최대 크기 10MB로 구성되어 있다. 실험에서 사용한 디스크 블록의 크기는 2KB, 4KB, 8KB, 16KB, 32KB이다.

제안하는 방법과 비교를 위해 함께 실험한 방법들은 (i) 너비 우선 탐색 형식을 사용한 데이타 할당 방법(Breadth First Allocation: *BFA*), (ii) 깊이 우선 탐색 형식을 사용한 데이타 할당 방법(Depth First Allocation: *DFA*), 그리고 (iii) *NATIX* 방법이다. 실험 2에서 사용한 질의는 그림 7에서 설명하였다. 다양한 질의 유형을 고려하여 선택한 질의들로서, Q1은 속성에 대한 필터를 포함한 질의이며, Q2는 텍스트에 대한 필터 조건을 포함하고 있다. Q3은 조상-자손 관계를 사용

Q1	**FOR** $b **IN** document("auction.xml")/site/open_auctions/open_auction **WHERE** $b/bidder/personref[@person="person18829"] **BEFORE** $b/bidder/personref[@person="person10487"] **RETURN** <history> $b/reserve/text() </history>
Q2	**COUNT(FOR** $i **IN** document("auction.xml")/site/closed_auctions/closed_auction **WHERE** $i/price/text() >= 40 **RETURN** $i/price)
Q3	**FOR** $p **IN** document("auction.xml")/site **RETURN** count($p//description) + count($p//annotation) + count($p//email);
Q4	**FOR** $p **IN** document("auction.xml")/site/people/person **LET** $a := **FOR** $t IN document("auction.xml")/site/closed_auctions/closed_auction **WHERE** $t/buyer/@person = $p/@id **RETURN** $t **RETURN** <item person=$p/name/text()> **COUNT** ($a) </item>

그림 7 실험 2에서 사용한 XQuery 집합

하는 질의이며, 마지막으로 Q4는 조인을 필요로 하는 질의이다.

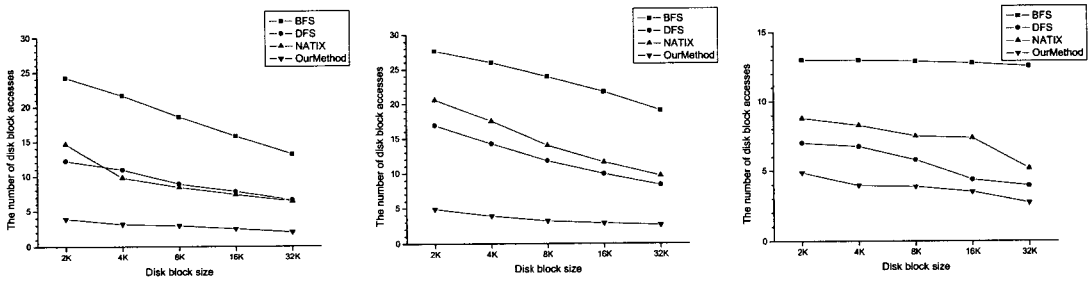
6.2 실험 결과

그림 8은 실험 1에서 임의의 노드 접근에 필요한 디스크 접근 회수의 최대값을 나타내며, 그림 9는 평균 값을 나타낸다. 모든 데이타 집합에 대해 제안하는 방법이 가장 적은 디스크 접근 회수를 보이고 있으며, 특히 제안하는 방법의 경우 최대값과 평균값의 차이가 매우 작음을 알 수 있다.

그림 10은 XMark 데이타 집합을 사용하여 XQuery 질의를 처리하는데 필요한 디스크 접근 성능을 측정하는 실험 2의 결과이다. 제안하는 방법의 성능이 다른 방법에 비해 뛰어나며, 디스크 블록의 크기가 큰 경우에 더욱 유리함을 알 수 있었다. 이는 클러스터링 효과가 디스크 블록의 크기가 클 때 더욱 유용함을 의미한다.

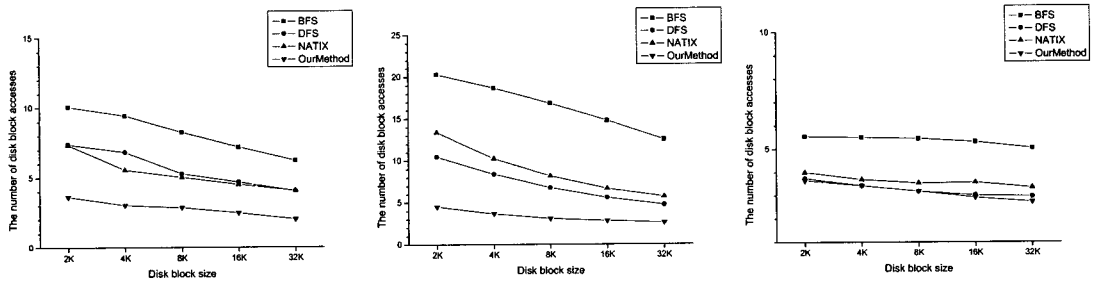
7. 결 론

방대한 양의 XML 데이타는 디스크 등의 보조 기억



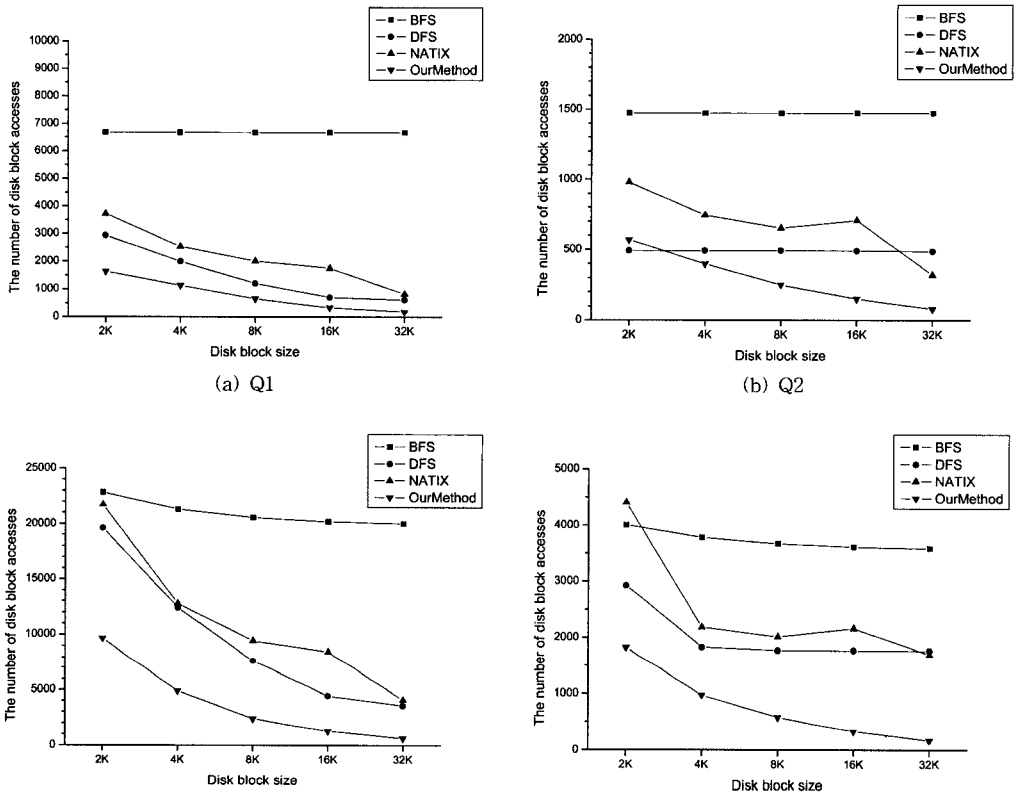
(a) Docbook (b) NITF (c) XMark

그림 8 실험 1 - 임의의 노드 접근에 필요한 최대 블록 접근 회수



(a) Docbook (b) NITF (c) XMark

그림 9 실험 1 - 임의의 노드 접근에 필요한 평균 디스크 접근 회수



(a) Q1 (b) Q2

(c) Q3 (d) Q4

그림 10 실험 2 - 질의 Q1~Q4 처리에 필요한 디스크 접근 회수 (Dataset: XMark)

장치에서 효율적으로 관리되어야 하며 디스크 블록에 저장하는 방식에 따라 성능차이가 매우 커질 수 있다. 본 논문에서는 데이타 트리를 디스크 블록 트리로 매핑하는 방법을 제안하였으며, 모든 노드에 대해서 트리의 높이가 최소가 된다는 것을 보였다. 제안하는 방법은 트리 넘버링과 디스크 블록 매핑의 두단계로 구성된다. 트리 넘버링 단계에서는 데이타 트리의 아래쪽에서 위쪽으로 각 노드마다 매핑 인디케이터 값을 할당한다. 그리고, 디스크 블록 매핑 단계에서는 앞 단계에서 할당한 매핑 인디케이터 값을 이용하여 노드들을 디스크 블록으로 매핑한다. 그리고, 실험을 통하여 제안하는 방법이 좋은 성능을 보여주는 것을 확인하였다.

제안하는 방법에서 생성하는 디스크 블록 트리는 높이가 최소이며 밸런싱된 형태를 갖는다. 따라서, 어떠한 노드를 찾더라도 적은 횟수의 디스크 블록 접근을 필요로 한다. 또한, 경로 질의의 경우에도 조건을 만족하는 노드를 찾는 것이 핵심적인 기능이 되므로, 본 논문에서 제안하는 방법을 사용하면 보다 나은 성능을 얻을 수 있었다.

본 논문에서 제안하는 할당기법은 XML 데이타만을 고려하고 있다. 데이타의 저장 위치를 빠르게 찾고자 하는 목적으로 색인을 사용할 경우, 색인을 통해 데이타의 저장 위치를 확인한 후, 해당 데이타를 디스크에서 읽게 되므로, 본 논문의 할당 기법은 데이타를 읽는데 필요한 디스크 접근 시간을 줄일 수 있다. 향후 연구로서, 색인 정보가 XML 데이타와 구분없이 같이 저장되는 색인 기법의 사용에 대한 고려가 필요하다. 이 경우, 색인과 XML 데이타를 동시에 고려하여 디스크에 할당하는 방법을 연구할 필요가 있다. 더 나아가 질의 처리 알고리즘에 최적화 될 수 있는 방법에 대한 고찰도 필요할 것이다.

참 고 문 헌

[1] Tim Bray, et. al., Extensible markup language (XML) 1.0 second edition W3C recommendation. Technical Report REC-xml-20001006, World Wide Web Consortium, 2000.

[2] W3C. Document Object Model (DOM), http://www.w3.org/DOM, Feb. 2000.

[3] Alin Deutsch, et. al., XML-QL: a query language for XML. In Proc. Of the Int. WWW Conf., 1999.

[4] James Clark, et. al., XML Path Language(XPath) Version 1.0, http://www.w3c.org/TR/xpath
[5] Don Chamberlin, et. al., XQuery: A Query Language for XML, W3C working draft. Technical Report WD-query-20010215, World Wide Web Consortium, 2001.
[6] Apache Xindice, http://xml.apache.org/xindice/
[7] D. Florescu, D. Kossman, Storing and Querying XML Data using an RDBMS. IEEE Data Engineering Bulletin 22(3), 1999.
[8] J. Shanmugasundaram et al. Relational Databases for Querying XML Documents: Limitations and Opportunities, In Proc. of VLDB, 1999.
[9] 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.
[10] Johnny Stenback, Andy Heninger, DOM Level 3 Load and Save Specification Version 1.0, http://www.w3.org/TR/2003/WD-DOM-Level-3-LS-20030619/
[11] C.-C. Kanne, G. Moerkotte, Efficient storage of XML data, Technical Report 8/99, University of Mannheim, 1999.
[12] H. V. Jagadish et al., "TIMBER: A Native XML Database," The VLDB Journal, 11:274-291, 2002.
[13] X. Meng, D. Luo, M. L. Lee, J. An, "OrientStore: A Schema Based Native XML Storage System," In proceedings of VLDB conference, pp. 1057-1060, 2003.
[14] DocBook Technical Commitee, DocBook, http://www.docbook.org/
[15] International Press Telecommunications Council, News Industry Text Format(NITF), http://www.nitf.org/
[16] XMark - An XML Benchmark Project, http://www.xml-benchmark.org

김 정 훈
1997년 2월 한국과학기술원 전산학과 학사. 1999년 2월 한국과학기술원 전산학전공 석사. 1999년 3월~현재 한국과학기술원 전산학전공 박사과정 재학중. 관심분야는 XML, Distributed Systems, Database Systems 등

손 진 현
1996년 서강대학교 전산학과 학사. 1998년 한국과학기술원 전산학과 석사. 2001년 한국과학기술원 전자전산학과 박사 2001년 9월~2002년 8월 한국과학기술원 전자전산학과 박사후 연구원. 2002년 9월~현재 한양대학교 컴퓨터공학과 교수

정 연 돈
1994년 2월 고려대학교 전산과학과 학사 1996년 2월 한국과학기술원 전산학과 석사. 2000년 8월 한국과학기술원 전자전산학과 전산학전공 박사. 2000년 9월~2001년 8월 한국과학기술원 정보전자연구소 Post-Doc. 연구원. 2001년 9월~2003년 2월 한국과학기술원 전자전산학과 전산학전공 연구교수. 2003년 3월~2006년 2월 동국대학교 컴퓨터공학과 교수. 2006년 3월~현재 고려대학교 컴퓨터·통신공학부 교수. 관심분야는 XML, Mobile/Broadcast Databases, Sensor Networks, Spatial Databases, Database Systems 등

김 명 호
1982년 서울대학교 컴퓨터 공학과(학사) 1984년 서울대학교 컴퓨터 공학과(석사) 1989년 Michigan State Univ. 전산학(박사). 1989년~현재 한국과학기술원 전산학전공 교수. 1995년 Univ. of Virginia 방문 교수. 관심분야는 Database, Distributed Systems, Workflow, Multimedia, OLAP, Data Warehouse