제 19 회 한국정보처리학회 춘계학술발표대회 논문집 제 10 권 제 1 호 (2003. 5)

연결 리스트를 이용한 3 차원 비트맵 인덱싱의 공간 축약

이재민, 황병연
가톨릭대학교 컴퓨터공학과
e-mail : {likedawn, byhwang}@catholic.ac.kr

A Space Compression of Three-Dimensional Bitmap Indexing using Linked List

Jae-Min Lee, Byung-Yeon Hwang
Dept. of Computer Engineering, Catholic University of Korea

요 약

기존의 웹 문서나 컨텐츠들의 표현적 한계를 극복하기 위한 방안으로 메타 데이터에 관한 다양한 연구가 수행되어졌고 그 결과의 산물중에 가장 대표적인 것으로 XML을 들 수 있다. XML은 문서의 내용뿐 아니라 구조까지도 기술할 수 있는 장점을 통해 향후 정보 교환에 핵심적인 역할을 할 것으로 기대되어지고 있으며 이에 따라 XML 문서를 효율적으로 저장하고 검색하기 위한 다양한 연구가 진행되고 있다. BitCube는 Bit-wise 연산이 가능한 3차원 비트맵 인덱싱을 사용하여 XML 문서들의 구조적 유사성에 따라 클러스터링하고 사용자의 질의에 대한 처리를 수행하는 인덱싱 기법으로 그것의 빠른 성능을 입증하였다. 그러나 BitCube의 클러스터링은 XML 문서의 경로에 중점을 둔 것이므로 클러스터와 경로가 담고 있는 실제 단어들간에는 연관성이 없으므로 3차원 비트맵 인덱스는 하나의 평면을 제외한 모든 평면이 굉장히 높은 공간 사용량을 갖는 희소행렬이 된다. 본 논문에서는 늘어나는 방대한 문서의 양으로 인한 시스템의 성능 저하를 막고 안정적인 성능을 유지할 수 있도록 기존 연산의 성능을 저하시키지 않으면서 공간을 최소화 할 수 있는 연결 리스트를 설계하고 3차원 비트맵 인덱스를 연결 리스트로 재구성하는 방법을 제시한다.

1. 서론

기존 웹 문서 및 컨텐츠를 담고 있는 전자 문서들은 데이터의 표현 방법에 중점을 둔 사용이 간단한 방식으로 작성되어져 왔다. 이러한 문서들은 대부분 확장성이나 구조성, 다양한 데이터의 형태 및 검사 기능에 있어서 한계성을 가지고 있다. 이러한 문서들을 대상으로 질의를 수행하여 적절한 결과를 얻어내는 일은 쉽지 않다. 기존의 문서들은 구조적 정보들을 명확히 담고 있지 않음으로 기존 검색 시스템은 데이터 형태를 텍스트와 같이 특정한 형태만으로 한정함으로써 범위검색과 같은 연산을 사용하는 것에 많은 어려움이 있었다. 또한 사용자는 양질의 결과를 적절한 수준의 양만 얻어내길 원하는 것에 반해, 사용자가 원하는 양질의 수준을 인식할 수 없는 시스템은 자체적으로 검색 결과의 수준을 검사하여 각각의 문서에 순위를 주고 대량의 결과물들 속에서 사용자가 원하는 문서가 있기를 기대하는 것이 보편적이었다. 따라서 최근에 연구되는 검색 시스템에서는 문서들 안에 부가적인 정보를 담고 있는 메타데이터를 추가하여 위와 같은 문제점들을 해결하고자 노력하였다. 이러한 연구들이 진행되어지는 가운데 1998년에 W3C에 의해 표준으로 제정된 XML(eXtended Markup Language) [1]이 등장하게 되었다.

XML은 기존 문서 표현 방식과는 달리 자기 서술적인 (self-describing) 특징을 바탕으로 표현과 데이터, 구조를 분리시켜 데이터를 표현하고 교환하는 새로운 표준으로 부상하였다. 실제 소프트웨어 시장에서 XML은 XML/ EDI와 B2B, CRM(Customer Relationship Management)외에 다양한 e-business 솔루션들에서 전사적인 혹은 기업 대 기업간의 데이터 통합 및 효율적인 관리, 웹 기반 컨텐츠 관리, 비즈니스 통합 관리 등에 활용되어지고 있다. 이러한 XML의 활발한 연구 및 사용은 구조 정보를 담는 문서들을 빠르게 확산시키는 시발점이 되었고 이로 인하여 이러한 구조 정보를 잘 활용한 검색 방법의 필요성이 대두되게 되었다[2,3].

BitCube는 XML 문서의 저장과 검색을 위해 설계된 Bit-wise 연산이 가능한 1비트의 필드들로 구성된 3차원 비트맵 인덱스이다[4,5]. 이것은 문서의 태그들이 갖는 경로를 중심으로 문서와 클러스터간의 구조적 유사성을 판별하고 클러스터링한다. 그리고 사용자의 질의에 대해 정의된 연산을 통해 결과를 반환한다. 그러나 BitCube는 태그 자체의 의미인 경로를 중심으로 클러스터링을 수행함으로써 실제 태그

가 담고 있는 내용, 즉 문서의 실질적인 내용은 클러스터내의 문서들간에 전혀 연관성을 갖지 못한다. 이로 인해 3차원 비트맵 인덱스는 문서 ID와 경로 ID로 구성된 하나의 평면을 제외하고는 모든 평면이 필연적으로 희소행렬이 될 수 밖에 없으며 이것으로 인해 BitCube는 굉장히 높은 공간 사용량을 갖게 된다. 3차원 비트맵 인덱스를 구성하는 필드가 1비트의 작은 크기이지만 XML의 발전에 따른 빠른 확산 및 보급 속도와 이 기법이 XML 문서의 저장 및 검색을 위해 설계되었음을 고려할 때, 정보를 제공하기 위해 보유하는 문서의 양이 기하급수적으로 증가함으로써 발생하는 공간적 낭비와 이에 따른 처리 속도의 저하는 자명하다.

본 논문은 기존의 질의 처리를 위한 연산의 성능에 영향을 주지 않으면서 공간적인 낭비를 최소화할 수 있도록 연결 리스트를 설계한다. 또한 3차원 비트맵 인덱스를 연결 리스트로 재구성하여 늘어나는 방대한 문서량에도 안정적인 성능을 유지할 수 있는 방법을 제시한다.

2. BitCube와 연산들

본론에 들어가지 전에 3차원 비트맵 인덱싱에 대한 이해가 필요하므로 본 장의 1절과 2절을 통해 BitCube와 질의를 처리하기 위한 연산들에 대하여 간단히 소개한다[4,5].

2.1 BitCube

BitCube는 문서 ID와 경로 ID 그리고 단어 ID로 구성된 Bit-wise 연산이 가능한 1비트의 필드들로 구성된 3차원 배열이다. XML 문서는 (p, v)쌍의 집합으로서 정의되어 진다. 여기서 p는 root element로부터 기술된 element path(또는 ePath)이다. 그리고 v는 ePath에 대한 word 이거나 content를 나타낸다. Text-based 문서들을 핸들링하는 전형적인 방법들은 빈도수 테이블을 사용하거나 문서들에 대한 words를 나타내는 inverted(또는 signiture)파일을 사용하는 것이다. 그러나 XML 문서들은 XML elements(또는 XML tags)에 의해 표현되어지기 때문에 위의 전형적인 방법들은 효과적이지 못하다. XML 문서에 대한 BitCube는 다음과 같이 정의된다.

$$BitCube = (d, p, v, b)$$

여기서 d는 XML 문서를 뜻하고, p는 ePath를 뜻하고, v는 ePath에 대한 word 또는 content를 뜻하고 b는 BitCube에서 한 비트에 대한 값인 0 또는 1을 뜻한다. 결과적으로 Bitcube는 클러스터에 존재하는 모든 문서가 갖는 경로와 단어를 각각 한 비트의 정보로 인덱싱을 위한 3차원 공간에 모두 담고 있게 된다.

2.2 BitCube의 연산

BitCube는 사용자의 다양한 질의를 처리하기 위해 다음과 같은 연산을 수행한다.

① ePath Slice

특별한 ePath에 대해서 각각의 비트들은 조각 내어진다. 이 연산은 Path를 입력 값으로 받아들여서 그것과 관련된 words를 가지는 문서들의 집합을 리턴한다.

$P\_Slice(ePath)$ = { $(doc, word)$ |ePath는 doc안에서 사용된다. 그리고 word는 ePath와 연관된다.}.

이 Slicing의 결과물은 words의 집합을 가지고 있는 문서들의 집합을 나타내는 비트맵 인덱스이다.

② Word Slice

특별한 Word에 대해서 각각의 비트들은 조각 내어질 수도 있다. 이 연산은 word(search key)를 입력 값으로 받아들여서 문서들의 집합을 반환한다.

$W\_Slice(word)$ = { $(doc, ePath)$ |word는 doc안에서 차례로 사용되어지는 ePath와 연관된다.}.

결과물은 word와 관련된 ePath의 집합들을 가지는 문서들의 집합을 나타내는 비트맵 인덱스이다.

③ Document Project

BitCube의 각 행들은 추출(Projection)될 수 있다. 이 연산은 문서를 입력 값으로 받아들여서 그것들의 Epath와 연관있는 words를 가지는 ePath들의 집합을 반환한다.

$Project(doc)$ = { $(ePath, word)$ |doc안에 있는 전체 content와 ePath 쌍이다. }.

결과물은 그것들의 content(혹은 word)를 가지는 ePath들의 집합을 나타내는 비트맵 인덱스이다.

3. 3차원 비트맵 인덱싱의 공간 축약

기존 3차원 비트맵 인덱싱에서 XML 문서들은 삽입과 동시에 그 구조적 유사도에 따라 적절한 클러스터로 분류되고 클러스터들은 내부에 존재하는 모든 문서들의 구조 정보와 단어 정보에 근거하여 3차원 비트맵 인덱스를 구성한다. 그런데 문서를 분류하기 위해 사용하는 구조적 유사도는 경로에 중점을 두고 있으므로 문서의 실질적인 내용, 즉 경로가 담고있는 실제 문서의 내용과는 연관성이 없는 클러스터를 형성하게 된다. 예를 들어 'book'에 대해 기술하기 위해 만든 XML 문서는 'title', 'author', 'publisher', 'ISBN'과 같은 태그로 구성이 되는데 각각의 문서들은 그것이 갖는 태그로부터 경로를 추출하고 그것으로 구조적 유사성을 판별한다. 'book'이라는 범주로 분류된 클러스터내의 문서들은 대부분이 'title'이라는 태그를 갖겠지만 그 태그가 갖는 실제 문서의 내용인 책의 제목은 그것을 구성하는 단어들간에 공통점이 있을 수 없다. 컴퓨터를 소개하는 책의 제목과 요리를 소개하는 책의 제목에 공통적으로 빈번히 사용되는 단어는 극소수에 불구할 것이다. 또, 'author'나 'publisher'와 같은 태그도 마찬가지이다. 작가와 출판사가 세상에 유일하거나 극히 소수라면 문제가 되지않겠지만 실제로는 그렇지 않고 클러스터에는 굉장히 다양한 저자명과 출판사명이 들어갈 것이다. 'ISBN'의 경우, 모든 책들은 자신의 고유한 ISBN을 갖게 되므로 해당 클러스터의 3차원 비트맵 인덱스는 단 한 비트의 정보를 담기 위해 '문서 ID의 개수 × 경로 ID의 개수'만큼의 비트를 낭비하게 된다. 결과적으로 3차원 비트맵 인덱스에서 클러스터링을 위해 구성된 하나의 평면을 제외한 나머지 모든 평면들은 희소행렬이 된다.

다음에 오는 절들에서는 이것을 연결 리스트로 전환하는 방법을 구체적으로 제시한다.

3.1 연결 리스트의 설계

ePath Slice와 Document Project 연산을 수행하기 위해 리스트는 해당 노드의 단어 ID와 다음 단어ID를 가리키는 포인터가 있어야 한다. Word Slice 연산의 경우는 경로 ID와 다음 경로 ID를 가리키는 포인터가 필요하고 추가적으로 문서 ID와 다음 문서 ID를 가리키는 포인터가 있어야 한다. 그러나 Word Slice 연산이 각 문서 ID에 대하여 연결 리스트를 따라 순차적으로 수행되는 연산임을 고려할 때, '문서 ID의 개수 × 단어 ID의 개수'만큼의 헤드 노드들이 문서 ID와 다음 문서 ID를 가리키는 포인터를 갖는다면 다른 일반 노드는 그것을 포함하지 않도록 설계할 수 있다. 결과적으로 연결 리스트에서 헤드 노드는 12바이트의 공간을 사용하고 다른 일반 노드들은 8바이트의 공간을 사용한다. 그림 3.1은

연결 리스트를 구성하는 노드의 구조를 나타내며 우측에 있는 것이 일반 노드의 구조이고 좌측에 있는 것이 헤드 노드가 가져야 할 추가적인 필드들의 구조를 의미한다.

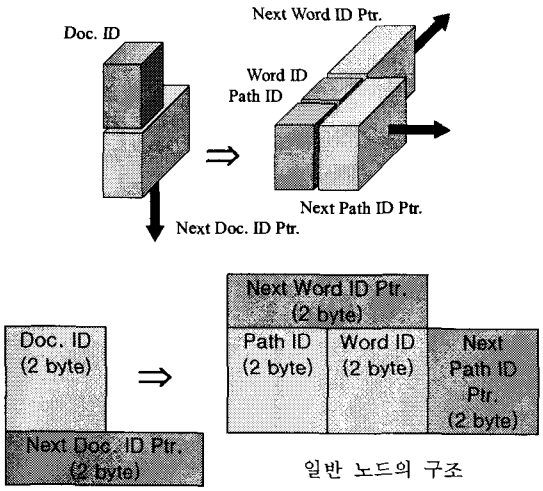

그림 3.1 노드의 구조

3.2 연결 리스트를 이용한 인덱스의 재구성

BitCube는 경로를 중심으로 클러스터링을 수행하므로 문서 ID와 경로 ID로 구성된 첫 평면은 매우 견고하게 구성된다. 이 평면을 'Base Bitmap'이라 한다. 새로 제안된 기법은 여기에 추가로 'Base Bitmap'과 대응하여 노드를 가리키는 포인터를 그것의 필드로 하는 'Doc. and Path Ptr. Reference Map'을 갖는다. 그리고 단어 ID에 따라 순차적으로 대응되는 노드를 가리키는 포인터를 필드로 하는 'Word Ptr. Reference List'를 갖는다. 이 두 개의 '포인터 참조 인덱스'를 사용하여 연결 리스트의 노드들에 접근한다. 일단 참조 인덱스를 사용해 연결 리스트에 접근하면 모든 연산은 어떠한 비교 연산도 수행할 필요없이 순차적으로 결과를 읽어들여서 반환한다. 그림 3.2는 본 논문에서 제안한 연결 리스트를 이용한 3차원 비트맵 인덱싱의 공간 축약의 전체적인 도식을 나타낸다.

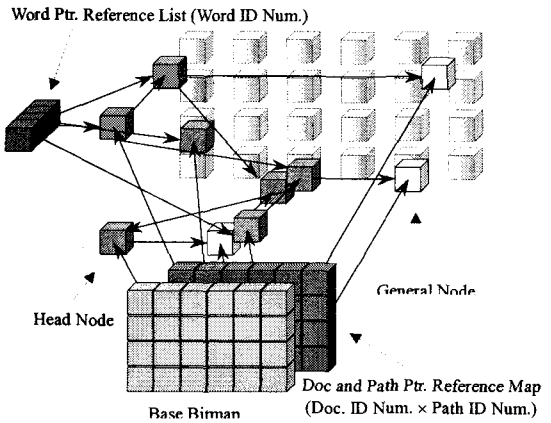

그림 3.2 연결 리스트를 이용한 3차원 비트맵 인덱싱

3.3 연결 리스트를 이용한 연산

BitCube에서 수행 가능한 모든 연산은 본 논문에서 제시한 방법을 사용하여 동일한 시간에 처리가 가능하다. 다음은 각각의 연산에 대한 알고리즘이다.

① ePath Slice

이 연산은 주어진 경로를 포함하는 문서 ID와 단어 ID로 구성된 리스트를 반환하는 연산이다. 제안한 방법에서 이 연산의 알고리즘은 다음과 같다.

```
Method P_Slice( ePath )
for i=0 to Path.ID.Num. - 1
  if  Base.Bitmap [0][i] = true
    if  Path.Name [i] = ePath
      for j=0 to Doc.ID.Num. - 1
        if  Base.Bitmap  [j][i] = true
          readWords(j, i)
      break

Method readWords(i, j)
node = ref.Map  [i][j]
n = Node.DocID
while node != null
    print n and Node.WordID
    node = node.nextWord
```

- *Path.ID.Num.* = 클러스터내에 존재하는 경로 ID의 개수
- *Base.Bitmap* = 문서 ID와 경로 ID로 구성된 2차원 인덱스
- *Path.Name* = *Base.Bitmap*의 인덱스에 대응되는 실제 경로 ID
- *Doc.ID.Num.* = 클러스터내에 존재하는 문서 ID의 개수
- *ref.Map* = *Base.Bitmap*의 인덱스에 대응되는 포인터를 필드로 하는 2차원 인덱스
- *Node.DocID* = 노드가 갖는 문서 ID
- *Node.WordID* = 노드가 갖는 단어 ID

② Word Slice

이 연산은 주어진 단어를 포함하는 문서 ID와 경로 ID로 구성된 리스트를 반환하는 연산이다. 제안한 방법에서 이 연산의 알고리즘은 다음과 같다.

```
Method W_Slice( Word )
for i=0 to Word.ID.Num. - 1
  if  Word.Name [i] = Word
    node = ref.List [i]
    while node != null
      temp = node
      readWords( Node.DocID , temp)
      node = node.nextDoc
  break

Method readWords(n, node)
while node != null
    print n and Node.PathID
    node = node.nextPath
```

- *Word.ID.Num.* = 클러스터내에 존재하는 단어 ID의 개수
- *Word.Name* = 실제 단어 ID의 1차원 배열
- *ref.List* = *Word.Name*의 인덱스와 대응되는 포인터를 필드로 하는 1차원 인덱스

- *Node.PathID* = 노드가 갖는 경로 ID

③ Document Project

이 연산은 주어진 문서가 갖는 모든 경로 ID와 단어 ID로 구성된 리스트를 반환하는 연산이다. 제안한 방법에서 이 연산의 알고리즘은 다음과 같다.

```
Method Project( Doc )
for i=0 to Doc.ID.Num. - 1
  if  Base.Bitmap [i][0] = true
    if  Doc.Name [i] = Doc
      for j=0 to Path.ID.Num. - 1
        if  Base.Bitmap [i][j] = true
          readWords(i, j)
      break

Method readWords(i, j)
node = ref.Map [i][j]
while node != null
    print Node.PathID and Node.WordID
    node = node.nextWord
```

- *Doc.Name* = *Base.Bitmap*의 인덱스에 대응되는 실제 문서 ID

4. 실험 및 결과

기존의 3차원 비트맵 인덱싱과 연결 리스트를 이용하여 공간 축약을 수행한 인덱스로 문서 수의 증가에 따른 공간 복잡도의 변화를 측정하였다. BitCube의 크기는 '문서 ID의 개수 × 경로 ID의 개수 × 단어 ID의 개수'개의 비트가 된다. 이에 반해 제안된 기법은 3.1절에서 언급한 것처럼 일반 노드인 경우 한 노드의 크기가 64비트(8바이트)가 되고 헤드 노드인 경우 96비트(8바이트)가 되므로 전체 크기는 '64 or 96 × 클러스터 내의 모든 단어의 개수 + 부가적인 공간'개의 비트가 된다. 여기서 '부가적인 공간'이란 연산을 위해 필요한 노드들을 가리키는 포인터를 필드로 하는 인덱스이며 그것은 3.2절에서 언급한 'Doc. and Path Ptr. Reference Map'와 'Word Ptr. Reference List'와 같은 '포인터 참조 인덱스'를 의미한다. 이 공간의 크기는 '문서 ID의 개수 × 경로 ID의 개수 × 16 (2바이트) + 단어 ID 의 개수 × 16(2바이트)'개의 비트가 된다. BitCube의 경우 경로 ID가 늘어날수록 실험에서 불리해지고 또 단어의 중복률이 낮아져도 불리해진다. 이에 따라 실험의 대상은 모두 대형 인터넷 서점인 'Amazon.com'사이트의 문서로 제한하여 문서의 경로 ID의 수가 늘어나는 것을 방지하고 문서의 분야도 'Books→ Subjects→ Science→ Biological Sciences→ Biology → General'로 제한함으로써 가능한 단어의 중복률이 높아질 수 있도록 실험을 수행하였다. 'Bestselling'의 순서로 문서를 삽입한 결과 9개 이상의 문서가 클러스터에 축적되면서 제안한 방법이 기존의 3차원 비트맵 인덱싱보다 낮은 공간 사용량을 지속적으로 유지했다. 이것은 XML 문서의 저장 및 검색을 위해 하나의 클러스터가 보유해야 할 문서의 실제적인 양을 고려할 때, 성능이 같다면 본 논문에서 제안된 방법이 공간의 효율적인 이용면에서 기존의 방법보다 뛰어나다는 것을 의미한다. 그림 4.1은 문서의 증가에 따른 두 기법의 공간 사용량의 변화를 나타내는 실험의 결과 그래프이다.

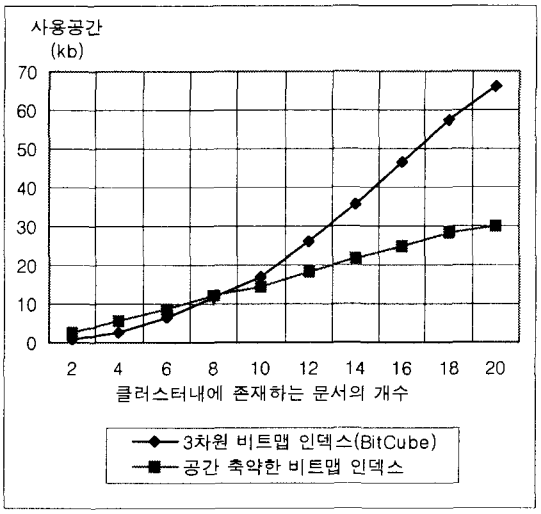

그림 4.1 문서량 증가에 따른 공간 사용량의 비교

5. 결론

기존의 BitCube에서 질의에 대한 빠른 처리를 위해 사용하던 3차원 비트맵 인덱싱은 매우 높은 공간복잡도를 갖는다. 본 논문에서는 같은 연산을 동일 시간에 처리하면서 공간 사용률을 줄일 수 있는 연결 리스트를 이용한 공간 축약에 관한 방법을 제안하였다. BitCube의 인덱스는 Bit-wise 연산이 가능한 한 비트로 구성되어 있다. 이에 비해 제안된 기법은 연결 리스트를 이용하므로 하나의 노드의 크기가 작게는 64배에서 많게는 96배의 크기를 갖게 되므로 클러스터 내에 존재하는 문서의 수가 적으면 오히려 더 많은 공간을 사용하게 된다. 그러나 실험 결과 클러스터내에 문서의 개수가 2개일 때, 약 4배 정도의 많은 공간을 사용하던 제안된 기법은 클러스터내에 문서의 개수가 9개를 넘어서자 기존의 3차원 비트맵 인덱싱보다 더 낮은 공간 사용량을 지속적으로 유지했다.

앞으로 시스템이 완전히 구현되지 않은 관계로 특정한 클러스터를 대상으로 실험을 수행할 수 밖에 없었던 점을 보완하여 전체 클러스터를 대상으로 성능을 평가할 수 있도록 구현에 힘쓸 것이며 기존 기법이 수행할 수 없었던 다양한 연산에 관한 연구를 계속하여 수행할 것이다.

참고문헌

[1] "Extensible Markup Language(XML) 1.1", W3C Candidate Recommendation, 2002

[2] J. Bosak, "XML, Java, and the Future of the Web", http://www.xml.com/pub/a/w3j/s3.bosak.html, 1997

[3] R. Bourret, "XML and Databaes", http://www.rpbourret.com/xml/XML/XMLAndDatabases.htm, 2002

[4] J. Yoon, V. Raghavan, and V. Chakilam, "BitCube: Clustering and Statistical Analysis for XML Documents", 13th International Conference on Scientific and Statistical Database Management, Virginia, July 18~20, 2001

[5] J. P. Yoon, V. Raghavan, V. Chakilam, and L. Kerschberg, "BitCube: A Three-Dimentional Bitmap Indexing for XML Documents", Journal of Intelligent Information System, Vol.17, pp.241~254, 2001