구조 유사도를 이용한 경로 기반의 색인 기법

김연혜* 이재민 황병연
*다음기술㈜ 응용기술팀
가톨릭대학교 컴퓨터공학과
*yhkim@ntechs.com
{likedawn, byhwang}@catholic.ac.kr

An Indexing Method based on the Path using Structure Similarity

Yun-Hye Kim* Jae-Min Lee Byung-Yeon Hwang
*Next Technologies, Professional Service
Dept. of Computer Engineering, Catholic University of Korea

요 약

기존의 웹 문서나 컨텐츠의 한계를 극복하기 위해 메타데이터에 대한 연구가 활발히 이루어진 가운데 그 산물로 등장한 XML은 현재 다양한 분야에서 그 활용에 관한 연구가 활발히 진행되고 있다. 그리고 그 중에서 XML문서 자체를 저장 및 검색하는 부분에 대한 연구도 많은 성과가 있었다. XML의 대표적인 특징은 기존의 다른 컨텐츠와는 달리 문서의 구조를 기술할 수 있다는 것이며 이런 구조적 정보는 활용 방법에 따라 XML문서의 다양한 처리에 있어 성능을 향상시키는 핵심적인 요소가 될 수 있다. 이에 본 논문에서는 기존의 비트맵 인덱스(Bitmap Index)를 확장하여 역파일 색인 방법과 결합시켜 P_INDEX를 제안하고, P_INDEX를 활용한 다양한 경로 중심의 검색 방법을 제시한다.

1. 서론

기존의 웹 문서나 컨텐츠들은 그 표현에 중점을 두고 있었으므로 기존 검색 시스템들은 주로 전문에서 추출한 요약문을 사용해 정보를 저장하고 문서나 컨텐츠에 순위를 매기는 방법을 사용하여 질의에 대한 처리를 수행하였다. 만약 "2000년 이후에 나온 개구리 소년에 대한 신문 기사"를 찾고자 한다면 '2000년', '개구리', '소년', '신문'과 같은 주요 단어 또는 단어들의 조합으로 사용자가 원하는 검색 결과를 얻으려고 노력하였다. 그러나 여기에는 여러 가지 문제점이 발생한다. 사용자가 원하는 적절한 결과를 얻기 위해서는 '2000년 이후'라는 조건을 수행하기 위해 날짜라는 특수한 데이터형을 가지며 범위검색을 허용하여야 한다. 또한 그 데이터의 출처는 신문 혹은 방송계이고 제목 혹은 내용에 가장 많이 나오는 단어가 '개구리'와 '소년'이라면 적절한 결과를 얻게 될 것이다. 그러나 문서들은 위와 같은 정보들을 자세히 담고 있지 않거나 기존 검색 시스템에서 데이터 형태를 텍스트와 같이 특정한 데이터 형태만으로 한정지음으로써 범위검색과 같은 연산을 사용하는 데 어려움이 많았다. 또한 사용자는 양질의 결과를 적당량만 얻어내길 원하는데 반해, 사용자가 원하는 양질의 수준을 모르는 시스템들은 자체적으로 검색 결과의 질을 검사하여 순위를 매긴 대량의 문서들에서 사용자가 원하는 문서들이 있길 바란다. 따라서 최근에 연구되는 검색 시스템에서는 문서들 안에 부가적인 정보를 담고 있는 메타데이터를 추가하여 위와 같은 문제점들을 해결하고자 노력하였다.

한편, XML의 등장으로 시스템은 데이터베이스의 테이블에서 정보를 추출하듯이 문서의 구조가 되는 태그에서 정보를 추출하는 것이 가능하게 되었다[1,2].

본 논문에서는 XML 문서에서의 구조 검색을 가능하게 하는 비트맵방식의 색인과 기존의 역파일 색인방식을 결합한 P_INDEX를 제안하였고 P_INDEX를 활용한 다양한 검색방법을 제시한다.

2. 관련연구

비트맵 인덱스는 같은 경로를 갖는 비율이 높은 문서들을 묶어 놓은 문서ID와 경로ID를 축으로 하는 Bit-wise 연산 가능한 필드로 구성된 2차원 배열이다. 문서ID는 인덱스를 구성하는 모든 XML 문서들 각각에게 주어진 식별자이며 경로ID는 문서내에 존재하는 내용을 갖는 태그의 전체 경로들 각각에게 부여되는 고유한 식별자이다. 경로ID를 갖는 경로는 ePath(요소 경로)라 하며 태그의 내용을 제외한 모든 조상 태그들의 조합된 이름을 사용한다.

어떤 ePath 가 비트맵 상의 거의 모든 문서 내에 존재한다면 "ePath는 대중적이다"고 말하며 대중성(popularity)을 표현하는 식은 다음과 같다.

$$pop(pi) = |pi| / [pi]$$

|pi|는 비트맵 pi의 크기, 즉 0 값이 아닌 값을 가지는 bit의 개수를 의미하고, [pi]는 비트맵 pi의 차수(cardinality), 0값을 가지는 bit의 개수와 0값이 아닌 값을 가지는 bit 개수의 합을 가진다. $pop(pi) \geq n\ (0 \leq n \leq 1$,주어진 실수)이라면 pi 는 n-popular bit라고 하고, $pop(pi) \leq m$ 일 때는 m-unpopular bit ($0 \leq m \leq 1$,주어진 실수)라고 한다. 대중성을 위한 임계치(threshold)가 n($0.5 \leq n \leq 1$)으로 주어졌을 때, ePath들은 다음과 같이 세 가지 경우로 나뉠 수 있다. (1) $pop(pi) \geq n$ 인 경우 "popular bits"라고 한다. (2) $pop(pi) \leq 1-n$ 인 경우 "weakening unpopular bits"라

고 하며 (3)$1-n < pop(pi) < n$ 인 경우 "strengthening unpopular bits"라고 한다.

클러스터 혹은 문서의 중심(center)은 분할된 XML 문서들의 집합 안에서 popular bits와 strengthening un-popular bits가 1로 구성되고 weakening unpopular bits가 0으로 구성된 비트 벡터(bit vector)이다.

xOR 가 exclusive or 연산이라고 할 때, 두 문서들 사이의 거리는 다음과 같다.

$$dist(di,dj) = |xOR(di,dj)|$$

두 개의 문서나 혹은 비트맵에서의 행들간의 유사도(Similarity)는 다음과 같다.

$$sim(di,dj) = 1-|xOR(di,dj)| / MAX([di],[dj])$$

$sim(di,dj) > \xi$ ($0 \le \xi \le 1$, 주어진 실수)라면 문서 di, dj 는 ξ - similar라고 한다.

비트맵 인덱스를 기반으로 하는 XML 문서 검색 시스템인 BitCube는 XQEngine, XYZFind와 같은 동종의 검색 시스템과의 실험을 통해 그 성능이 입증되었으며 문서의 수가 증가하는 것에 따르는 성능 저하에도 매우 안정적이라는 것이 증명되었다[3,4].

그러나 경로를 중심으로 문서의 클러스터링을 수행하는 비트맵 인덱스는 완전히 일치하는 경로에 대한 인식은 가능하지만 같은 의도로 작성된 경로가 갖는 구조적 유사성을 식별하지 못함으로써 정확한 클러스터링을 기대하기 어렵다. 그림 2.1은 비트맵 인덱스의 예이다.

PathID / DocID	1	3	4	7	9	...	N
1	1	1	1	1	1	...	1
2	1	0	1	1	1	...	0
4	0	1	1	0	1	...	1
5	0	0	1	1	0	...	1
7	1	0	1	1	0	...	0
Pop.	0.6	0.4	1	0.8	0.6	...	0.6
Cen.	1	0	1	1	1	...	1

Pop.≡ 대중성(popularity) , *Cen.*≡ 중심(center)

그림 2.1 비트맵 인덱스

3. P_INDEX

3.1 P_INDEX 구성

본 논문에서는 비트맵 인덱스와 기존의 역파일 색인방법을 결합하여 P_INDEX를 제안한다. P_INDEX는 시스템 내 모든 문서들의 ePath들을 보존하는 ePath Table과 문서와 ePath간의 관계를 보여주는 맵, 그리고 ePath가 의미하는 노드들의 시작위치와 종결위치를 나타내는 역파일을 갖는다. P_INDEX 전체 구성은 다음 그림 3.1과 같다.

ePath Table은 새로 입력된 XML 문서의 구조를 분석하여 존재하는 ePath들을 중복되지 않게 보존한다. ePath들은 고유한 ID로 구분된다[5].

문서와 ePath간의 관계를 보여주는 맵은 문서고유ID와 ePathID로 XY축이 구성된 맵으로써 비트맵과 달리 0과 0이 아닌 정수를 값으로 가지게 된다. 그림 3.1의 P_INDEX 구성 예제를 살펴보면 (D1, p9) = 0 값을 가지는데 이는 D1문서에는 p9를 의미하는 ePath가 존재하지 않는다는 것을 의미한다. 만약 모든 문서와 시스템내 모든 ePath들과의 맵을 소유하게 되면 맵은 전체적으로 희소행렬이 되므로 매우 큰 저장공간을 필요로 한다. 따라서 2장에서 살펴본 문서들간의 유사도를 계산하여 하나의 맵을 여러 개의 클러스터링 된 맵들로 분할하고 정보가 없는 열(ePathID)들을 삭제함으로써 효율적인 저장공간을 사용하도록 한다.

마지막으로 역파일은 ePath 노드가 실제 문서 내 시작위치와 종결위치를 가지고 있다. ePath노드의 시작위치와 종결위치 사이에는 데이터, 자손노드들이 위치하게 된다.

ePathID	ePath
p0	section.section
p1	section.section.section
...	...

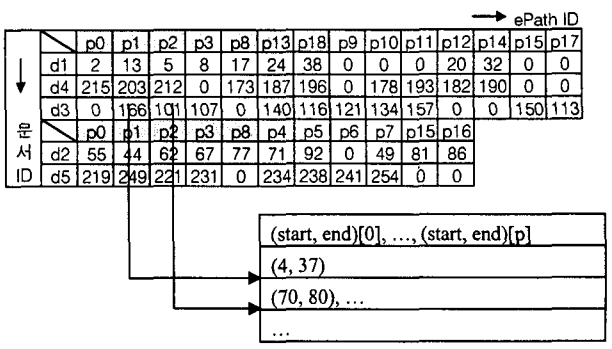

그림 3.1 P_INDEX 구성 예제

3.2 P_INDEX 검색 방법

P_INDEX를 사용하여 다음과 같은 검색이 가능하다. 다음과 같은 노드 구조를 검색할 때에는 ePath Table을 이용하여 결과를 구한다.

1. 특정 노드의 부모/자식노드 검색
2. 특정 노드의 조상/자손노드 검색
3. 특정 노드의 형제노드 검색

이를 확장하여 문서와 ePath간의 관계 맵을 이용하여 문서를 검색할 수 있다.

4. 특정 노드와 자식노드를 포함한 문서찾기
5. 특정 노드의 자손노드를 포함한 문서찾기
6. 특정 노드의 형제노드를 포함한 문서찾기

다음은 부모노드를 검색하는 알고리즘이다.

```
Method getParent(ePathID)
    If (getEPath(ePathID).cutElement(1) is not null)
        return ePath
    Else
        return null
  End If
```

- getEPath(ePathID) : ePath 클래스 메소드, ePathID로 ePath를 구할수 있는 메소드
- cutElement(int i) : ePath 클래스 메소드, 마지막 노드 삭제후 ePath 반환
- allEPaths : ePath 클래스 배열(모든ePath)
- resultEPathSet : ePath 클래스 배열(결과)

다음은 자식노드를 검색하는 알고리즘이다.

```
Method getChild(ePathID)
j = 0
For i = 0 to allEPaths
targetEPath = allEPaths[i].cutElement(1)
    If (getEPath(ePathID).equals(targetEPath))
        resultEPathSet[j++] = allEPaths[i]
    End if
End For
return resultEPathSet
```

다음은 조상노드를 검색하는 알고리즘이다.

```
Method getAncestor (ePathID)
j = 0;
For i = 1 to ePathDepth
    if (getEPath(ePathID).cutElement(1) is not null)
        resultEPathSet[j++] = (ePathID).cutElement(1)
    End If
End For
return resultEPathSet
```

다음은 자손노드를 검색하는 알고리즘이다.

```
Method getDescendants(ePathID)
j = 0
For i = 0 to allEPaths
sourceEPath = getEPath(ePathID)
targetEPath = allEPaths[i]
    If(sourceEPath.equals(targetEPath.substr(0,sourceEPath.lenth))
        resultEPathSet[j++] = allEPaths[i]
    End if
End For
return resultEPathSet
```

다음은 형제노드를 검색하는 알고리즘이다.

```
Method getBrothers(ePathID)
j = 0
For i = 0 to allEPaths
targetEPathID = allEPaths[i].getID()
    If(getParent(ePathID)).equals(getParent(targetEPathID))
        resultEPathSet[j++] = allEPaths[i]
    End if
End For
return resultEPathSet
```

문서들의 저장 시스템에서 직접적인 XML 문서 접근을 감소시키고 문서 입력시 전처리기에서 추출된 정보만을 가지고 여러 가지 구조 검색과 기존의 검색시스템에서 적용된 전문역파일 색인과의 결과셋을 합쳐서 내용-구조 혼합 P_INDEX를 이용한 검색 방법은 XML과 같은 구조화된 문서 검색도 가능하다. 그 외에도 요청하는 문서와 유사한 구조를 가진 문서들을 검색하는 연산을 수행할 수 있다.

그림 3.2는 실제 시스템에서 경로 관련 질의를 수행하는 예이다. 본 논문에서 제안한 기법을 사용하여 구현된 시스템은 기존 기법보다 매우 작은 인덱스 사이즈를 사용하면서도 다양하게 경로를 탐색할 수 있다.

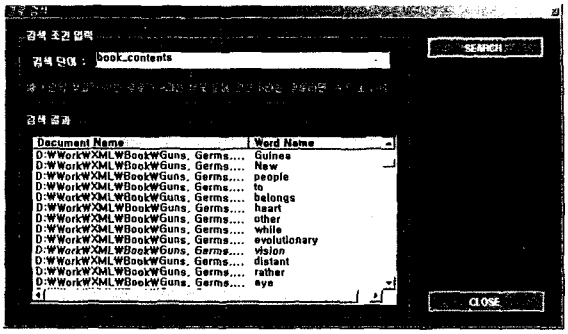

그림 3.2 경로 관련 검색의 예

4. 결론

본 논문에서 제안한 P_INDEX는 다음과 같은 특징을 가진다. 첫째, 다양한 구조 검색 가능: 구조 검색의 연산 알고리즘에 의해 다양한 구조 검색이 XML 문서를 직접 접근하지 않고도 연산되도록 한다. 둘째, 구조가 유사한 문서 검색 가능: 구조가 비슷한 문서를 찾아내는 연산이 가능해 진다. 이러한 구조 유사를 검색하는 방식은 이전에 시도하지 않았던 구조 검색 연산이다. 다양한 구조를 가질 수 있는 XML 문서들 중에서 원하는 문서와 내용과는 상관없이 구조가 비슷한 문서들을 추천하여 사용자에게 확장된 구조 검색을 할 수 있게 제공한다. 셋째, 신속한 구현과 성과 창출: 구조 검색을 사용하기 위해 기존 내용 검색 시스템을 완전히 다른 시스템으로 교체하는 것이 아니라 기존 내용 검색 시스템을 약간 변경시키거나 추가적인 모듈을 사용함으로써 전체 시스템을 구현할 때 소비되는 시간과 비용을 절감할 수 있게 된다.

향후 연구과제로는 XML 질의어 표준이 된 XQuery에서 요청하는 모든 요구사항을 만족하도록 개선하고, P_INDEX의 최적화를 위한 문서들의 삽입/수정/삭제시 문서 클러스터링에 대해서 개선하고자 한다.

5. 참고문헌

[1] R. Bourret, "XML and Databases", http://www.rpburret.com/xml/XMLAndDatabases.htm, 2003

[2] L. Feng, E. Chang, and T. Dillon, "A Semantic Network-Based Design Methodology for XML Documents", ACM Transaction on Information System, Vol.20, No.4, 2002

[3] 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

[4] 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

[5] M. Yoshikawa, T. Amagasa, T. Shimura, and S. Uemura, "XRel: A Path-Based Approach to Storage and Retrieval of XML Documents Using Relational Databases," ACM Transactions on Internet Technology, 1(1), 110-141, 2001