XPath 처리를 위한 효과적인 XML 저장 시스템의 설계 및 구현

김경원° 손기락
한국외국어대학교 컴퓨터 및 정보통신공학과
kwstone@daps.hufs.ac.kr , ksohn@hufs.ac.kr

Design and Implementation of an Efficient XML Storage System for XPath Processing

Kyungwon Kim° Kirack Sohn
Dept. of Computer & Information Telecommunication Engineering,
Hankuk University of Foreign Studies

요 약

XML은 웹 문서의 표준이며, 네트워크 상의 문서교환 포맷으로 자리잡아가고 있다. 이에 따라, 많은 XML데이터를 저장하고 관리하는 방법에 대한 연구가 활발하게 진행되고 있다. 저장 및 관리 해야할 XML문서의 양이 증가함에 따라 XML데이터의 검색을 효율적으로 지원할 수 있는 저장구조가 필요하게 되었다. 따라서, 본 논문에서는 XML Path 기반의 검색질의에 효과적으로 대응할 수 있는 XML 저장 시스템을 설계 및 구현 하고자 한다.

1. 서 론

XML은 1996년 W3C(World Wide Web Consortium)에서 제안한 것으로서, 네트워크 상에서 구조화된 문서를 전송 가능하도록 설계된 표준화된 텍스트 형식이다. 이는 인터넷에서 기존에 사용하던 HTML의 한계를 극복하고 SGML의 복잡함을 해결하는 방안으로써 사용자가 새로운 태그를 정의할 수 있도록 자기서술적 능력을 추가한 것이다.

이런 XML의 특성 때문에 네트워크 상에서 문서교환 표준으로 많은 사람들에게 각광받고 있고, 그 사용이 점점 늘어가는 추세이다. XML문서 형식의 사용이 늘어남에 따라 많은 양의 XML데이터를 저장, 관리해야할 필요성이 대두되었고, 지금도 대용량 XML데이터를 저장하고, 효율적인 검색을 지원할 수 있는 XML 저장 시스템에 대한 연구가 활발하게 진행되고 있다.

또한, 문서의 구조와 의미를 표현할 수 XML문서로부터 필요한 정보를 추출하기 위해 문서 특성에 맞는 많은 질의어들이 개발되었다. 그 중 XPath는 Element와 Attribute를 이용한 정규 경로에 관한 표현이 쉬워서 누구나 쉽게 이해할 수 있고 쉽게 습득할 수 있다. 뿐만 아니라, 임의 경로로의 이동이 쉬워서 많은 XML 검색엔진에서 사용되고 있다.

본 논문에서는 XPath 질의를 효과적으로 처리할 수 있는 저장 구조를 제안한다. 또한, XML데이터를 SAX Parser를 통하여 본 논문에서 제안한 저장구조에 저장하고, 이 데이터에서 XPath 질의를 SQL 질의로 변환하여 수행한 결과를 XML문서 형태로 반환하는 시스템의 설계 및 구현을 기술한다.

2. 관련연구

2.1. Edge

Edge[2]는 XML 데이터를 Edge관점에서 접근하여 다음과 같이 하나의 Table로 표현한다.

Edge (Source, Ordinal, Target, Label, Flag, Value)

Source는 Node를 표현하고, Ordinal은 Node의 반복횟수를 나타낸다. Target은 이 Element의 부모 Node를 가르킨다. 이를 이용하여 Node들의 Tree 구조를 표현할 수 있다. Label은 node의 이름을 표현하고, Flag는 "ref"와 "val" 두 종류의 값을 가지게 되며, 해당 Node가 Value를 가지는지를 표현하게 된다.

Edge는 Parsing된 XML 데이터를 위와 같은 구조로 저장하게 된다. Edge는 간단한 구조로 구성이 되어 있어서 구현이 간단한 반면에, XML Path 기반의 질의를 처리하기 위해서 많은 Join연산을 필요로 한다.

2.2. XRel

XRel[3]은 XML 데이터를 Node관점에서 접근하여 다음과 같이 4개의 Table로 표현한다.

Path (PathID, Pathexp)
Element (DocID, PathID, Start, End, Ordinal)
Text (DocID, PathID, Start, End, Value)
Attribute (DocID, PathID, Start, End, Value)

Path Table은 XML 데이터를 Parsing하면서 발생하는 모든 Path를 표현하게 된다. 예를 들면, (1, "/namecard/phone")와 같이 Node가 가질 수 있는 Path를 모두 표현해 준다. Element Table은 Path Table을 참조하여 Node의 정보를 표현한다. Start와 End는 XML 데이터를 Parsing하는 과정에서 Parsing된 순서에 따라 번호를 부여하는 DFS-Numbering 기법[4]을 이용하여 Node들의 Tree구조를 표현하게 된다. Text Table과 Attribute Table은 각각 Element와 Attribute의 Content와 Value를 표현한다.

XRel은 Path Table의 Pathexp를 이용하여 대부분의 XML Path 기반의 질의의 처리를 Edge보다 효과적으로 처리할 수 있다. 하지만, "*"나 "//" 표현이 포함된 질의를 처리하기 위해

서는 Start와 End를 이용한 여러개의 θ-Join연산을 필요로 한다.

2.3. XParent

XParent[5]는 앞에 소개한 Edge와 XRel의 장점을 결합하여 만든 구조로써, XML 데이터를 다음의 5개의 Table로 표현한다.

LabelPath (ID, Len, Path)
DataPath (Pid, Cid)
Element (PathID, Did, Ordinal)
Data (PathID, Did, Ordinal, Value)
Ancestor (Did, Ancestor, Level)

LabelPath, Element, Data Table은 XRel의 Table들과 유사하다. XRel에서 사용된 DFS-Numbering 기법 대신에 DataPath Table을 이용하여 Node들의 Tree 구조를 표현하였다. 또, XParent의 특징적인 것은 Ancestor Table을 이용하여 각 Node들의 모든 조상들을 표현한다는 것이다. 이를 이용하여 질의처리의 성능을 향상시킬 수 있다. 하지만, Ancestor Table의 Record 수는 Element의 개수에 비례해서 증가하게 되므로 많은 양의 XML 데이터를 처리하게 될 경우 질의처리의 성능을 저하 시키는 원인이 될 수도 있다.

3. XPath 처리를 위한 효과적인 저장구조

관련연구에서 살펴본 몇 가지 XML 저장구조에는 몇가지 문제점들이 있다. 본 논문에서는 이러한 문제점들을 보안하여 XML Path기반의 질의처리의 성능을 향상시킬 수 있는 XML 저장구조를 다음과 같이 제안한다.

Path (PathID, Path, Length)
SubPath (ID, PathID, StartLevel, EndLevel, SubPath)
Element (EID, PathID, Parent, Ord)
Data (EID, PathID, Ord, Value)

Path Table은 (표 1)과 같이 표현되며, Node가 가질 수 있는 모든 Path를 저장한다. Path는 Node가 가질 수 있는 전체 Path를 모두 표현하고, Length는 표현된 Path가 몇 개의 Node로 구성되었는지를 표현한다.

Element Table은 (표 3)과 같이 표현되며, Node의 기본 정보를 저장한다. PathID는 Path Table의 PathID를 참조하고, Parent는 현 Node의 부모 Node의 EID를 가진다. Ord는 Node의 반복 횟수를 표현한다.

Data Table은 (표 3)과 같이 표현되며, Element나 Attribute들의 Value를 저장한다. EID는 Element Table의 EID를 참조하여 Value의 부모 Node를 표현한다. Value는 현 Node의 Vale를 표현한다.

본 논문에서 제안한 저장구조 중 가장 특징적인 부분은 SubPath Table이다. SubPath Table은 (표 2)에서 볼 수 있듯이 Path Table의 Recoed들 각각을 다시 Node별로 분리하여 가질 수 있는 모든 조합을 만들어서 표현하는 방법이다. 각각 분리된 Node들을 조합하여 만든 SubPath들은 Path Table의 PathID를 참조하고, 조합된 SubPath들의 각각 시작하는 Node의 Level과 끝나는 Node의 Level을 가지고 있다.

이 SubPath를 이용하면 관련연구에서 살펴본 저장구조들로는 표현하기 힘든 XML Path 기반의 질의의 처리 성능을 향상시킬 수 있다.

기존 저장구조에서는 "*"나 "//"를 포함하는 XPath질의를 SQL 질의로 변환하여 처리할 경우, 많은 수의 Join 연산을 이용하거나 Cost가 높은 연산을 이용하여 처리하게 된다. 이런 경우에, 본 논문에서는 "*"나 "//"를 기준으로 하여 Path를 분리하고, 각각 분리된 Path들을 SubPath Table의 SubPath들과 비교하여 분리된 Path들의 공통적인 PathID를 찾아내게 된다. 몇 개의 Equi-Join만을 이용하여 "*"나 "//"를 포함하는 XPath 질의를 SQL 질의로 변환이 가능하기 때문에 질의의 처리 성능을 향상할 수 있다. (그림 2)는 XParent와 본 논문에서 제안한 저장구조를 이용하여 "*"를 포함하는 XPath 질의를 처리하기 위해 변환된 SQL 질의이다.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<namecard>
<name eng="kkw">kwstone</name>
  <company>HUFS</company>
  <phone>
      <office>031-332-4456</office>
      <hp>017-652-8894</hp>
  </phone>
</namecard>
```

그림 1. (표 1,2,3)을 위한 Smaple XML Document

표 1. Path Table

PathID	Path	Langth
1	/namecard	1
2	/namecard/name	2
3	/namecard/name/@eng	3
4	/namecard/company	2
5	/namecard/phone	2
6	/namecard/phone/office	3
7	/namecard/phone/hp	3

표 2. SubPath Table

ID	Pi	SL	EL	SubPath	ID	Pi	SL	EL	SubPath
1	1	1	1	/namecard	15	5	2	2	/phone
2	2	1	1	/namecard	16	5	1	2	/namecard/phone
3	2	2	2	/name	17	6	1	1	/namecard
4	2	1	2	/namecard/name	18	6	2	2	/phone
5	3	1	1	/namecard	19	6	3	3	/office
6	3	2	2	/name	20	6	1	2	/namecard/phone
7	3	3	3	/@eng	21	6	2	3	/phone/office
8	3	1	2	/namecard/name	22	6	1	3	/namecard/phone/office
9	3	2	3	/name/@eng	23	7	1	1	/namecard
10	3	1	3	/namecard/name/@eng	24	7	2	2	/phone
11	4	1	1	/namecard	25	7	3	3	/hp
12	4	2	2	/company	26	7	1	2	/namecard/phone
13	4	1	2	/namecard/company	27	7	2	3	/phone/hp
14	5	1	1	/namecard	28	7	1	3	/namecard/phone/hp

표 3. Element Table & Data Table

EID	PathID	Parent	Ord
1	1		1
2	2	1	1
3	3	2	1
4	4	1	1
5	5	1	1
6	6	5	1
7	7	5	1

EID	PID	Ord	Value
2	2	1	kwstone
3	3	1	kkw
4	4	1	HUFS
6	6	1	031-332-4456
7	7	1	017-652-8894

```
-- 본 논문의 Schema
select d1.Value
from SubPath sp1, SubPath sp2, Data d1
where sp1.SubPath = '/namecard'
  and sp2.SubPath = '/hp'
  and sp2.StartLevel = sp1.EndLevel + 2
  and sp1.PathID = sp2.PathID
  and d1.PathID = sp2.PathID;

-- XParent의 Schema
select d1.Value
from LabelPath p1, Data d1
where p1.Path like '/namecard/%/hp'
  and p1.len = 3
  and p1.ID = d1.PathID;
```

그림 2. XPath("/namecard/*/hp")를 변환한 SQL

4. 성능 실험 및 분석

본 논문에서 제안한 저장구조와 기존 저장구조를 XPath 질의를 이용하여 처리성능을 비교하였다. 사용된 XML데이터는 DBLP Bibliography XML Record(121M)를 이용하였고, 성능 실험에는 700MHz Dual CPU, 1GB Memory, 56GB HDD, Server를 이용하였다. 질의는 아래의 7가지 XPath 질의를 사용하였다.

Q1. /dblp/www
Q2. /dblp/article/title/sub/sup/i
Q3. //title/sub
Q4. //article//sup
Q5. /dblp/www/author[2]
Q6. /dblp/article/[author = 'Frank Manola']
Q7. /dblp/article[//year = '1994']/cdrom

성능 실험 결과는 (그림 3)과 같다.

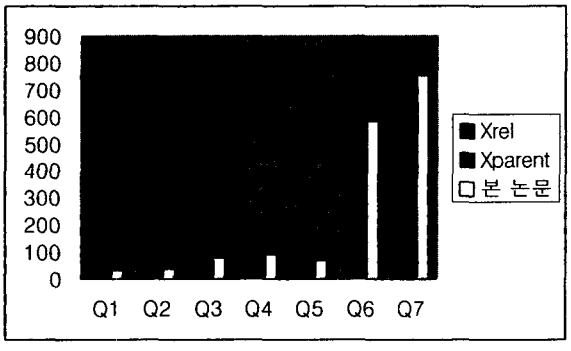

그림 3. 성능비교 그래프

(그림 3)의 성능실험 결과에서 볼 수 있듯이 본 논문에서 제안한 저장구조를 이용한 경우가 Q3, Q4, Q7과 같이 "*"나 "//"를 포함하는 XML Path 기반의 질의 처리에 대하여 좋은 성능을 보여주었다.

6. XML 저장 시스템 구성

본 논문에서 제안한 저장구조를 이용한 XML 저장 시스템의 전체적인 구성은 (그림 4)와 같다.

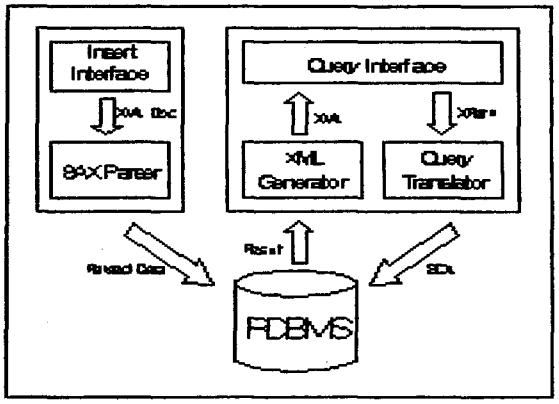

그림 4. 시스템 구성도

Insert Interface가 XML문서를 SAX Parser를 이용하여 Parsing 후, RDB의 본 논문에서 제안한 저장구조에 저장한다. Query Interface를 통하여 들어온 XPath질의는 Query Translator를 이용하여 SQL질의로 변환되고, 변환된 SQL질의를 RDB에 처리한다. 결과로 반환된 데이터들은 XML Generator를 통하여 XML문서 형태로 출력된다.

7. 결론 및 향후 연구

본 논문에서는 기존에 발표된 대표적인 XML 저장구조에 대하여 살펴보고, 이들의 문제점을 보안하고, XML Path 기반의 질의 처리의 성능을 향상시킬 수 있는 효과적인 저장구조를 제안하였다. 또한, 기존의 XML 저장구조와 본 논문에서 제안한 XML 저장구조를 구현하고, 실험을 통하여 XML Path기반의 질의의 처리 성능을 비교분석 하였다. 마지막으로, 본 논문에서 제안한 XML저장 구조를 이용하여 XML 문서를 RDB에 저장하고, XPath 질의를 SQL 질의로 변환하여 결과를 XML 문서 형태로 제공하는 시스템을 설계 및 구현하였다.

향후 연구 과제로는 다양한 XPath 질의에 대하여 변환된 SQL 질의를 최적화하여 질의 처리성능을 개선하는 방법에 대한 연구가 필요하다.

참고문헌

[1] Tim Bray, Jean Paoli, C.M.Sperberg-McQueen, and EveMaler. Extensible Markup Language (XML) 1.0 second editon W3C recommendation. Technical Report REC-xml-20001006, World Wide Web Consortium, October 2000.

[2] Florescu, Kossmann, "A performance evaluation of alternative mapping schemes for storing xml data in a relational database", Technical report., 1999.

[3] Yoshikawa, Amagasa, "XRel : A path-based approach to storage and retrieval of XML documents using relational database", ACM Transactions on Internet Technology., 2001.

[4] 이용석, 손기락, "XML문서 저장 시스템 설계 및 구현", 정보과학회 학술발표 논문집(I), 1998.

[5] Haifeng Jiang, Hongjun Lu, Wei Wang, Jeffrey Xu Yu, "XParent: An Efficient RDBMS-Based XML Database System", IEEE., 2002.