효율적인 XML문서의 필터링을 위한 SFilter 설계 및 구현

장복선[0]*, 손기락*
한국외국어대학교 컴퓨터 및 정보통신공학과
boksun77@orgio.net , ksohn@hufs.ac.kr

Design and Implementation of SFilter for Efficient Filtering of XML Documents

Boksun Jang[0], Kirack Sohn
*Dept. of Computer & Information Telecommunication Engineering,
Hankuk University of Foreign Studies

요 약

효율적인 문서 교환을 위해 의미 있는 태그를 사용하는 XML문서가 인터넷상에서 널리 사용되고 있다. XML문서에 대한 정보를 추출하기 위해 많은 질의어가 사용되고있지만 특히 정규 경로 표현에 있어 임의 이동이 쉽고 질의 표현이 쉬운 *XPath*[2]가 사람들에게 각광을 받고 있다. 이 연구에서는 *XPath*[2]를 이용하여 사용자 질의를 등록하고, 등록된 질의를 이용하여 효율적으로 XML문서를 필터링하기 위한 방법을 제안한다. 본 논문에서는 *NiagaraCQ*[1]와 같이 XML문서 정보를 이용하여 사용자에게 계속적으로 XML 문서를 제공하는 Continuous Query 시스템에 사용된 SFilter를 설계하고 구현하였다.

1. 서 론

인터넷의 발전과 더불어 네트워크 상에서의 정보의 교류가 많아짐에 따라 XML(eXtensible Markup Language)는 인터넷상에서의 효율적인 문서 교환을 위한 표준으로서 중요한 위치를 차지하게 되었다. 기존의 HTML은 구문검사를 할 수 없을 뿐만 아니라 구조적인 내용 제공을 할 수 없어서 문서 교환에서는 부족한 점이 많았다. 하지만 XML은 HTML Tag로서는 표현할 수 없는 문서의 구조를 자유롭게 기술할 수 있을 뿐만 아니라 문서에 들어있는 정보를 식별할 수 있기 때문에 전자 상거래 환경에서 XML이 문서 교환 표준으로 많은 관심을 받고 있다.

이제는 XML은 정보교환을 위한 목적으로 연구되고 있을 뿐만 아니라 XML을 이용한 효율적인 문서 관리와 분류, 사용자에게 정보를 제공하는 서비스 개발에 이르는 많은 연구가 진행되고 있다. 구조화되고 확장성 있게 내용정보를 제공할 수 있는 XML문서로부터 필요한 정보를 추출해 내기 위해, XML구조에 맞게 많은 질의 언어들이 개발되었다. XML정보 추출을 위해 개발된 질의언어는 다음과 같다. : *Lore, XML-QL, XML-GL, Quilt, XQL, XQuery, XPath*

*XPath*는 정규 경로 표현에 의해서 임의의 경로로 이동이 쉽고, 질의 표현이 간단하고 쉬워서 누구나 쉽게 이해하고 사용할 수 있다는 장점을 가지고 있다. 본 논문에서는 *XPath*를 사용하여 XML문서를 효율적으로 필터링하는 SFilter를 제안하고, SFilter의 설계 및 구현에 대해 기술한다.

2. 관련연구

XPath는 XML문서의 정보를 Element의 계층적인 구조 경로로 표현할 수 있다. 모든 Element node는 루트로부터 파생되고 각각의 Element node는 문자 데이터와 애트리뷰트를 가질 수 있다. XML문서는 시작태그와 종료태그를 가지고 트리형식으로 되어있으므로 그로부터 Element node의 범위나 깊이 레벨 정보를 표현할 수 있다.

전통적으로 필터링 시스템은 "Bag-of-words"의 기반을 가진 정보 검색 기술에서부터 개발되어왔다. 하지만 최근 정보의 양이 급증함에 따라 데이터베이스 개발자들도 관계형 데이터베이스 질의어와 XML 질의어를 이용하여 이와 비슷한 기능을 수행하게끔 만든 *NiagaraCQ*와 같은 *Continuous Query System*을 개발하였다. *NiagaraCQ* 시스템은 폭포수처럼 문서를 질의처리에 따라 여러 단계를 거쳐서 결과가 나오는 처리 방식을 사용한다. *NiagaraCQ* 시스템에서 사용된 질의 처리 주안점은 비슷한 질의를 그룹으로 묶어 크게 먼저 분류하고 점점 작게 분류되도록 하여 작업부하를 줄이도록 한데 있다.

이와 비슷한 시기에 개발된 *XFilter*[3]의 주안점은 XML Data Stream이 들어올 때 *Finite State Machines*(FSM)라는 인덱스를 사용하여 효율적으로 경로를 표현하고 이를 처리하는 것을 개발하는데 있다. FSM은 모든 Element의 경로표현을 각 State로 연결하므로, 질의 처리에 있어 자연스럽고 효과적인 표현방법이다. 문서가 들어오며 state에 적합한 Element가 발견되면 문서 내용에 따라 질의에 해당되는 마지막 state에 도달하여 그 문서가 질의를 만족시킨다는 것을 알릴 수 있다. FSM은 많은 구조적인 질의를 동시에 처리하는데 탁월한 성능을 보였지만, 비슷한 질의를 처리하는 부분에서 작업부하를 줄이지 못하는 단점이 있다.

그리고 2년 후에 이런 *XFilter*의 단점을 보완하여 여러 질의를 하나의 *Nondeterministic Finite Automaton*(NFA)로 묶은 *YFilter*[4]를 발표하였다. NFA는 사용자 질의를 표현하기 위해 필요했던 state에서 질의의 공통 Element부분을 집합으로 묶어 하나의 state로 만들어 state의 수를 줄였다. 뿐만 아니라, Automata를 이용하여 수행 속도를 빠르게 하고, 수행 작업을 분담하여 성능을 향상 시켰다. 하지만 각각의 Element가 각각의 state로 표현되므로 경로가 길면 state의 낭비가 일어날 수 있다.

본 논문에서는 XML 문서의 효율적인 필터링을 위해 *Suffix Tree*를 활용하여 위에서 제안했던 *YFilter*의 문제점을 보완한 SFilter를 제안한다.

3. SFilter의 구성

XML 문서 필터링은 크게 필터링 질의 트리 생성부분과 문서 처리부분으로 나누어 생각할 수 있다. 필터링 질의 트리는 사용자가 얻고자하는 XML 문서 경로에 대한 질의를 등록하여 문서 필터링의 기본이 되는 필터링 트리가 생성된다. 문서 처리부분은 입력된 XML문서가 정보제공을 위해 사용자가 등록한 질의 중에 어떤 질의를 만족하는지에 대한 처리를 수행하게 된다.

3.1 사용자 질의 등록

사용자는 *XPath*를 이용하여 질의를 등록한다. 사용자가 등록한 질의는 *JavaCC*(Java Compiler Compiler)[9]에 의해 파싱(Parsing)처리 하였다. 파싱된 결과는 *XPath* 경로 표현에 맞게 *Suffix Tree*를 구성하였다. 본 논문에서 사용한 *XPath* 경로 표현은 다음과 같이 분류할 수 있다.

- // Title
- / Book / Paper
- / Book / * / Title
- / Book // Author

본 논문에서 제안되는 필터링 알고리즘은 *Suffix Tree* 노드를 사용하여 필터링 트리를 구성하였다. *Suffix Tree* 노드의 구조는 (표 1)과 같다.

Start_Path	Parsing 된 Element중 노드를 구성하는 경로 표현 중 첫 번째 Element
Suffix_Path	첫 번째 Element를 제외한 나머지 경로 정보
Level_Info	문서의 트리구조 중 Root에서부터 노드에 포함된 경로의 깊이 정보
Query_List	경로 정보에 의해 노드에 해당되어지는 사용자 질의 정보
Left_Child	하위 트리를 구성하는 노드
Right_Sibling	같은 레벨의 트리를 구성하는 노드

표 1 *Suffix* 트리 노드의 구조

각 Element 간에 계층적인 구조를 표현하기 위해 "/" 와 "//" 연산 기호를 사용하였다. 우선 연산기호 "//"를 사용하는 "절대경로"에 대해 알아보자. "절대경로"에서는 요소(Element)의 위치에 대한 제약이 없으므로 요소 이름만으로 조건을 만족시킨다. "//" 이미지를 갖는 트리 노드를 따로 구성하여 어떤 경우에라도 비교 연산을 할 수 있도록 한다.

다음으로 "단순경로"를 생각할 수 있다. "단순경로"의 경우 루트에서 시작되어 "/" 연산자 사이의 두 요소의 관계가 "*부모-자식*"관계이므로 같이 사용될 가능성이 상당히 높다. 트리 노드를 구성할 때 부모와 자식을 각각 다른 노드로 구성하지 않고 하나의 node로 묶어 트리를 구성한다. Element 내용을 비교하여 질의를 처리하므로 작업 부하를 상당히 줄일 수 있다.

경로에 "Wild Character"가 들어있는 경우도 생각해야 한다. 이 경우는 "Wild Character"에 대해 파싱 과정부터 따로 설정된 값으로 트리 노드를 구성한다. 그리고 연결관계(*Level* 차이)가 "/*/"로 표시되어 있으므로 "부모-자식" 관계가 아니라, 트리의 깊이 수준(Depth Level)에 관하여 처리를 유보하여 추가 정보를 조건으로 제공하여 "Wild Character"에 대한 처리를 하였다.

위와 비슷한 방법으로 "//"도 처리 할 수 있다. 질의 예제 *Book//Author*는 *Book*의 하위 트리에 속한 노드이면서 *Author*이라는 이름을 가지기만 하면 그 조건이 충족되므로 그 경로나 깊이 수준(Depth Level)의 정보에 따라 질의가 처리되지 않고 처리가 유보되어 어떤 경우에라도 요소이름 비교연산 될 수 있도록 초점을 맞추었다.

이와 같은 방법으로 등록된 질의를 사용하여 (그림 1)과 같이 트리를 구성하였다.

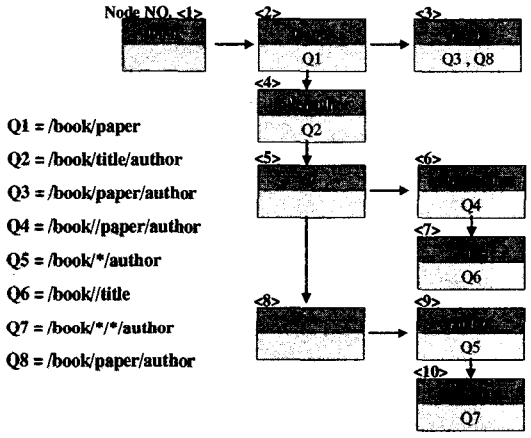

그림 1 XPath 질의와 관련 *Suffix tree*

3.2 Substring Decomposition

주어진 *XPath*로 표현되는 질의에 포함된 Element들의 순서는 $s=t_1 \cdot t_2 \cdot t_3 \cdots t_n$ 처럼 문자열로 정의될 수 있다. 정의된 문자열은 다른 문자열과 *Suffix Tree* 구성 알고리즘을 이용하여 공통된 부분 문자열을 찾아 낼 수 있다. 앞부분에서 찾아진 공통부분 문자열은 각각의 문자열 Element가 하나의 node를 구성하고, 나머지 부분의 서로 다른 문자열은 서로 다른 노드로 만들어져 트리를 구성하게 된다. 이는 하나의 Element가 하나의 노드로 만들어져 트리를 구성하는 것이 아니라 다수의 Element가 하나의 문자열로 정의되어 하나의 노드로 구성됨으로써 트리의 크기가 작아지고, 작업 수행 속도가 빨라진다.

(그림 2)는 질의가 입력됨에 따라 노드가 변화하는 Suffix *Tree*를 보여주고 있다. (그림 2)에서 보여지듯이 앞쪽의 공통된 부분은 남고, 나머지 서로 다른 부분에 대하여 다른 노드가 생성되어 *Suffix Tree*를 구성하게 된다. 이는 예전 *YFilter*에서 하나의 Element가 하나의 노드로 구성될 때보다 입력되는 문서의 질의 만족을 확인하는 작업에서의 부하를 줄이고, 노드의 낭비를 막음으로써 메모리를 효율적으로 관리할 수 있는 효과를 가져온다.

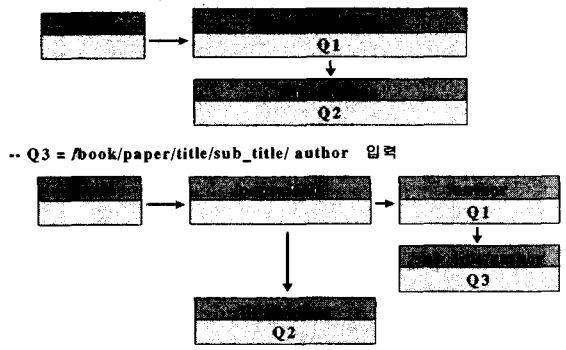

그림 2 XPath 질의 입력에 따른 *Suffix tree*의 변화

3.3 XML 문서 필터링 처리

XML 문서 입력을 위한 파싱(Parsing) 처리는 *SAX parser*[9]를 이용하였으며, 문서의 시작 태그와 종료 태그에 따른 이벤트 중심의 처리를 하였다. XML문서는 이들 태그에 의해 트리 구조를 갖게 되므로 관련된 하위 트리로의 검색으로 검색의 폭을 줄여나갈 수 있다.

(그림 3)은 문서 입력이 수행될 때의 필터링 시스템의 작동 원리를 보여주고 있다. 필터링 트리는 (그림 1)에서의 *Suffix Tree*를 사용하였다.

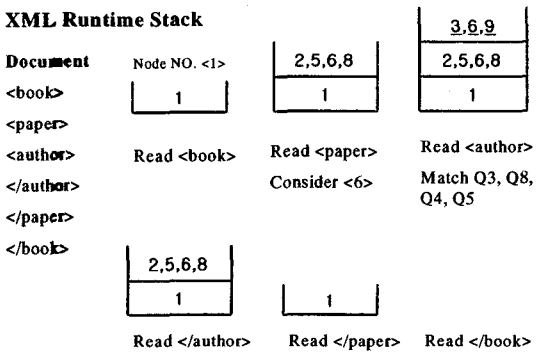

그림 3 XML Runtime Stack 상태변화

XML 문서가 입력되면서 시작 태그에 의해 이벤트가 발생된다. 문서에서 <*book*>을 읽어들이면 start-of-element 이벤트에 의해 필터링 엔진이 시작된다. 이 엔진은 현재 우선 스택에 들어있는 활성화된 노드가 무엇인지 확인한 후에 스택이 비어있을 경우 구성된 트리에서 현재 읽어들인 Element의 Content와 일치하는 것이 있는지 확인하고, 관련있는 노드만을 스택에 넣는다.

다음 Element <*paper*>를 읽어 들이면 스택이 비어있지 않으므로 스택에 들어있는 활성화된 노드만을 고려하여 비교연산을 하게 된다. <1>의 하위 트리에서 <*paper*>와 관련있는 노드를 다시 스택에 넣게 된다. 하지만 절대경로인 "//"의 경우 그 하위 트리 노드까지 고려하여야한다. 그 하위 트리중에 조건을 만족하는 노드가 있다면 그 노드까지 스택에 넣어야한다. "//" 노드는 다음 Element가 들어올 때 다시 고려되어야 하므로 스택에 남겨둔다.

<author>를 읽어들이면, 검색된 관련 노드의 Query_list가 비어있지 않으므로 Matching_list를 구성하게된다. 검색된 노드에 들어있던 Query_list를 Matching_list에 추가하게 된다. 바로 위에서 고려되었던 <6>의 경우, Start_Path뿐만 아니라 Suffix_Path까지 일치하는지 확인하여 Matching_list에 추가하게된다.

4. 결론 및 향후 연구

본 논문에서는 기존에 발표된 필터링 시스템의 특징에 대해 살펴보고, 이들의 문제점을 보완하여 XML문서의 구조적인 특성을 이용하여 XPath 질의에 맞는 필터링 트리를 구성하고, 이를 처리하는 필터링 방법을 제안하고, 구현하였다. 하지만 트리를 구성하는 과정에서 질의의 개수가 많아짐에 따라 트리가 거대해지고 질의 입력 처리 또한 늦어지는 것을 알 수 있었다.

향후 연구 계획으로는 XML문서 처리 과정에서 작업 부하를 줄이고 처리를 효율적으로 하기 위하여 병렬처리 방법을 도입하여 대용량 데이터 처리에 대해 기능적으로 향상 된 필터링 엔진으로 개발하여 *Continuous Query System*에 상업적으로 활용 가능한 엔진을 구현하는 것이다.

5. 참고문헌

[1] J. Chen et al. NiagaraCQ, A Scalable Continuous Query System for Internet Databases. In ACM SIGMOD Conf., May 2000

[2] W3C. XML Path Language (*XPath*) 1.0., http://www.w3.org/TR/*XPath*, November 1999

[3] M. Altinel, M. J. Franklin. Efficient Filtering of XML Documents for Selective Dissemination of Information. In VLDB Conf., September 2000

[4] Yanlei Diao, Peter Fischer, Michael J. Franklin, YFilter: Efficient and Scalable Filtering of XML Documents. In ICDE Conf., February 2002

[5] Y.Diao, M. J. Franklin. NFA-based Filtering for Efficient and Scalable XML Routing. Technical Report, USB/CSD-1-1159, October 2001

[6] M. K. Aguilera, R. E. Strom, D. C. Sturman etc., Matching Events in a Content-based Subscription System. In Proc. of ACM PODC, May 1999

[7] S. babu, J. Widom, Continuous Query over Data Streams. In Stanford Univ. http://dbpubs.stanford.edu/pub/2001-9

[8] Y. Chen, K. Lwin, S. Williams, Continuous Query Processing and Dissemination. In Berkely, http://www.cs.berkely.edu

[9] Web Gain , JavaCC : Java Compiler Compiler, http://www.metamate.com/products/java_cc/

[10] David Megginson, SAX : A Simple API for XML, http://www.megginson.com/ SAX/