스트리밍 XML 데이터에 대한 빠른 트윅 질의 처리 기법*

류병걸, 박상현, 하종우, 이상근
고려대학교 정보통신대학 컴퓨터통신공학부
e-mail:{smart123, condols, okcomputer, yalphy}@korea.ac.kr

Fast Twig Query Processing for Streaming XML Data

Byung-Gul Ryu, Sang-Hyun Park, Jong-Woo Ha, SangKeun Lee
Division of Computer and Communication Engineering, Korea University

요 약

스트리밍 XML 데이터는 고정된 저장소에 유지되지 않고 사용자 측으로 계속적으로 데이터가 전송된다는 특성을 지닌다. 이러한 스트리밍 XML에 대한 질의 처리를 위해서는 효과적인 메모리 관리와 빠른 질의 처리 성능이 요구된다. 최근 최소한의 메모리 사용으로 효과적으로 트윅 질의를 처리하기 위한 기법인 StreamTX가 제안되었으나 반복적인 질의 처리 알고리즘 호출로 인해 불필요한 질의 처리 시간이 발생한다. 따라서, 본 논문에서는 이러한 불필요한 질의 처리 시간을 줄이기 위해 실시간으로 질의와 무관한 노드를 제거하여 보다 효과적인 질의 처리를 수행 기법을 제안한다. 제안된 기법은 기존 연구와 유사한 메모리 사용량을 가지면서도 빠른 질의 처리 속도를 가짐을 성능평가를 통해 검증한다.

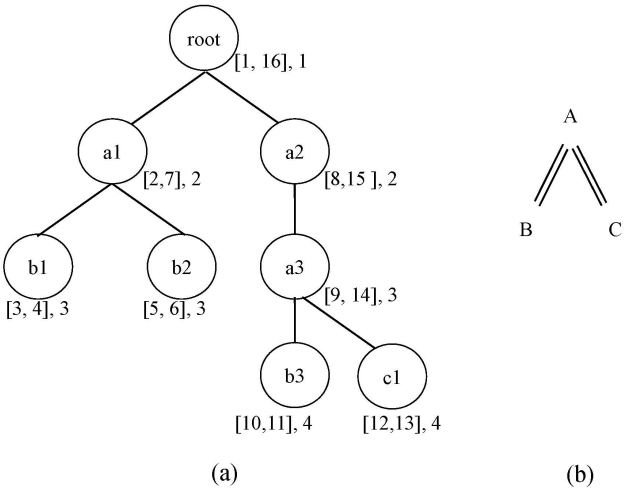
1. 서론

XML은 정보의 표현 및 교환의 표준으로서 문서 구조, 내용 및 출력 형식이 각각 분리되어 전체 문서를 정의하므로 문서 구조의 재사용, 출력 형식의 유연성 및 문서구조에 대한 검색과 같은 다양한 특성을 제공하며 이로 인해 XML 관련 기술의 확산은 더욱 가속화 되고 있다.

XML 분야의 가장 기본적이고 핵심적인 분야는 XML로 표현된 정보로부터 관심있는 정보만을 빠르고 효과적으로 얻기 위한 질의 처리 기술이다. 질의 처리를 위한 기존 대부분의 연구는 XML을 데이터베이스 시스템과 같은 고정된 저장소에 미리 색인된 상태로 데이터를 항상 유지하면서 XPath [1] 혹은 XQuery [2] 같은 XML 질의 언어를 통해 필요한 정보를 얻었다. 그러나 고정된 저장소에 XML 데이터를 유지하여 항상 접근 가능한 환경과는 달리 스트리밍 XML 환경은 고정된 저장소가 존재하지 않으며 추가 색인정보 등이 없이 가공되지 않은 형태의 XML 데이터가 연속적으로 사용자측으로 전송된다는 특성을 지닌다. 따라서 스트리밍 XML로부터 원하는 데이터만을 얻기 위해서는 실시간으로 다양한 형태의 사용자 질의에 대한 결과를 효과적으로 가져와야 한다.

이때, 표현 및 처리가 복잡한 질의를 표현할 수 있고, 다수 노드를 추출할 수 있는 장점을 가진 트윅 패턴 질의를 지원하기 위해서는 스트림상의 XML 태그정보를 관리하는 메모리 사용을 최소화하고 질의 처리과정의 계산 복잡도를 줄여 빠른 데이터 추출을 가능하게 하는 것이 중요하다.

일반적으로 XML 문서는 그림 1(a)와 같은 트리 형태로 표현되며 XML 문서에 대한 트윅 패턴 질의는 그림 1(b) 와 같이 표현된다. 주어진 트윅 질의에 상응하는 노드는 각각 (a_2, b_3, c_1), (a_3, b_3, c_1) 이다.



(그림 1) 예제 XML 문서와 트윅 질의

최근 제안된 StreamTX[3]는 유선상의 트윅 질의 처리

* 이 논문은 2009년도 정부(교육과학기술부)의 재원으로 한국과학재단의 지원을 받아 수행된 연구임(No. 2009-0077925)

알고리즘을 스트리밍 XML에 적용한 최초의 알고리즘이다. StreamTX는 TwigStack[4] 알고리즘을 기반으로 의사결정표(decision table)에 기반한 스트리밍 XML 질의 처리 기법을 제안하였다. 그러나 StreamTX는 각 스트리밍 노드에 대해 반복적으로 질의 처리 알고리즘을 사용하기 때문에 주어진 질의에 맞는 답을 찾기 위한 불필요한 처리시간이 발생한다. 따라서, 본 논문에서는 이러한 불필요한 질의 처리과정을 줄임으로서 질의 처리 시간을 최소화하기 위한 새로운 기법을 제안한다.

본 논문의 구성은 다음과 같다. 1장의 서론에 이어서 2장에서는 기존의 관련 연구 및 본 연구의 동기를 살펴본다. 3장에서는 제안하는 스트리밍 XML 질의 처리 기법을 기술한다. 4장에서는 제안기법과 StreamTX의 성능평가를 비교 분석하며, 5장에서는 향후 연구에 대해 살펴보고 결론을 내린다.

2. 관련연구

고정된 저장소의 XML 문서를 대상으로 트윅 질의를 처리하기 위한 초기연구에서는 질의에 존재하는 각각의 경로를 처리한 후 이를 통합하는 과정을 통해 최종 질의 결과를 추출하였다[5, 6, 7]. 그러나 이때 발생하는 불필요한 임시 결과와 비교 연산으로 인해 질의 처리 시간이 늘어나게 되는데 이러한 단점을 보완하여 처리시간을 줄이기 위한 연구 또한 계속적으로 진행되었다[4, 8, 9, 10].

그러나 이러한 연구들은 XML 문서가 고정된 저장소에 있을 경우에는 효과적이나 스트리밍 XML 환경에 적용할 경우 메모리 점유율이 높고 질의 처리 속도가 느려진다는 문제점을 가진다[11, 12, 13, 14]. 따라서, 스트리밍 XML 환경에서 효과적으로 질의를 처리하기 위한 기법들이 제안되었는데 대표적으로 StreamTX[3]는 스트리밍 XML 환경에서 최소한의 메모리를 사용하여 트윅 질의를 처리하기 위한 기법이다. StreamTX는 스트림 상으로 입력되는 XML 노드를 처리함에 있어 질의처리를 위한 저장소인 큐와 트윅 결과 매칭을 위한 저장소인 스택을 사용하여 스트림 노드의 특성을 고려한 의사 결정표(decision table)와 TwigStack 알고리즘을 기반으로 질의 처리 기법을 제안하였다.

그러나 StreamTX는 주어진 질의에 따라 질의 처리 알고리즘을 반복적으로 수행함으로서 불필요한 질의 처리 시간이 발생한다. 따라서, 이를 극복하기 위해 본 논문에서는 큐와 스택 두 개의 자료구조를 사용한 StreamTX와 달리 스택 자료구조만을 사용하여 StreamTX와 유사한 메모리 사용량을 가지면서도 불필요한 질의 처리 시간을 줄일 수 있는 기법을 제안한다.

3. XPruner : 스택기반 스트리밍 XML 프루닝

3.1 StreamTX 알고리즘 검토

그림 2는 스트리밍으로 전송되는 XML 문서(그림 1(a))에 대한 트윅 질의(그림 1(b))를 수행할 경우 StreamTX상에서 질의 처리시 큐에 존재하는 노드와 알고리즘 호출횟수를 나타낸다.* StreamTX는 기존에 제안된 기법들에 비해 메모리 관리는 효율적이나 여전히 불필요한 질의 처리 시간을 가지게 된다. StreamTX는 스트림상의 XML 노드 이벤트가 발생할 때마다 TwigStack 기반의 getNext 알고리즘을 실행한다.

	<a_1>	<b_1>	</b_1>	<b_2>	</b_2>	</a_1>	<a_2>
Q_a	a_1	a_1	a_1	a_1	a_1	a_1	a_2
Q_b	-	b_1	b_1	b_1, b_2	b_1, b_2	b_1, b_2	-
Q_c	-	-	-	-	-	-	-
실행 횟수	3	3	3	3	3	9	3

	<a_3>	<b_3>	</b_3>	<c_1>	</c_1>	</a_3>	</a_2>
Q_a	a_2, a_3	a_2, a_3	a_2, a_3	a_2, a_3	-	-	-
Q_b	-	b_3	b_3	b3	-	-	-
Q_c	-	-	-	c_1	-	-	-
실행 횟수	3	3	3	18	3	3	3

(그림 2) StreamTX 상에서의 질의 처리

예를 들어, 그림 2에서 <a_1><b_1>의 노드 시작 이벤트로부터 최종 </a_3></a_2>의 노드 끝 이벤트까지 getNext 알고리즘은 이벤트가 발생할 때 마다 주어진 질의의 노드 수만큼 반복적으로 계속 호출되며 그 총 횟수는 63회이다. 실제 위 예제에서 getNext 알고리즘은 <c_1> 노드의 시작 이벤트가 발생하였을 경우에만 필요하다. 하지만, StreamTX는 질의 결과 연산을 위해 필요하지 않는 이벤트가 발생할 경우에도 노드 수만큼 getNext 알고리즘이 실행되기 때문에 실제 질의 처리시간은 증가하게 된다.

3.2 XPruner 구조 및 스트리밍 XML 질의처리

StreamTX에서의 반복적인 질의 처리 알고리즘 호출로 발생하는 불필요한 질의 처리 시간을 줄이기 위해 본 논문에서는 스트리밍 XML 상에서 효율적인 질의 처리를 위한 XPruner를 제안한다.

Algorithm *XPruner(d)*

```
Input: an XML document d
Output: results matching the query
01: while(¬ (empty(d)))
02:     e = Current(d)
03:     if(isQueryNode(e)
04:         if(isStartEvent(event))
05:             if(isRootNode(e))
06:                 pushStack(Se,e,"","");
07:             end if
08:             if(findParentStack(e))
09:                 upDateParentStack(e)
10:                 pushStack(Se,e,e.pointerToParent(e),"")
```

* 지면관계상 StreamTX에서 큐의 노드상태에 따른 getNext 처리과정 및 스택상의 노드 처리과정은 생략하였다.

```
11:          end if
12:          if(¬empty(∀ Stack))
13:              for(qi∈ leafNodes)
14:                  outPut(qi)
15:              end for
16:          end if
17:          if(isLeafNode(e))
18:              if(trueParent(e))
19:                  outPut(qi)
20:              end if
21:          end if
22:      end if
23:      if(isEndEvent(event))
24:        if(e≠ leafNodes)
25:              count = pointToChildPoint(e);
26:              if(count != 0)
27:                  for i=1 to count
28:                      popDesStack(e.child)
29:                  end for
30:              end if
31:              pop(Se)
32:          end if
33:      end if
34:   end if
35: end while
```

본 논문에서 제안한 알고리즘은 3단계로 구분된다.

- **첫째, 스택 삽입**

  단계 5에서 단계 11까지는 시작 이벤트가 발생한 노드를 스택에 삽입한다. 노드의 시작 이벤트가 발생하면 알고리즘은 노드의 삽입을 위해 주어진 트윅 질의의 만족 여부를 상위 스택의 엘리먼트와 비교하여 만족할 경우에만 스택에 삽입을 한다.

- **둘째, 스택 삭제**

  단계 23에서 단계 33은 끝 이벤트가 발생한 노드를 스택에서 삭제한다. 노드의 끝 이벤트가 발생하면 이 벤트가 발생한 노드가 말단 노드가 아니면 현재 노드와 자손 노드까지 스택에서 삭제한다.

- **셋째, 결과 출력**

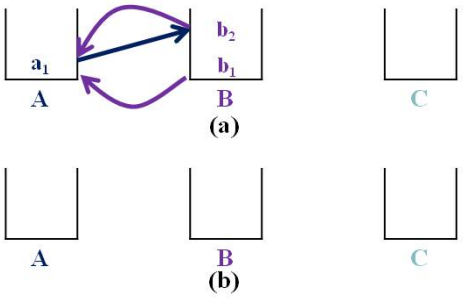
  단계 12에서 단계 21은 주어진 트윅 질의에 만족하는 노드가 스택에 들어있을 경우 결과값을 출력한다. 말단 노드가 들어온 후 스택에 주어진 트윅 질의의 결과값이 있을 경우에는 단말 노드를 삭제하고 결과값을 보여준다.

	<a_1>	<b_1>	</b_1>	<b_2>	</b_2>	</a_1>	<a_2>
S_a	a_1	a_1	a_1	a_1	a_1	-	a_2
S_b	-	b_1	b_1	b_1, b_2	b_1, b_2	-	-
S_c	-	-	-	-	-	-	-
실행 횟수	1	1	1	1	1	1	1

	<a_3>	<b_3>	</b_3>	<c_1>	</c_1>	</a_3>	</a_2>
S_a	a_2, a_3	a_2, a_3	a_2, a_3	a_2, a_3	a_2, a_3	A_2	-
S_b	-	b_3	b_3	b3	-	-	-
S_c	-	-	-	c_1	-	-	-
실행 횟수	1	1	1	2	1	1	1

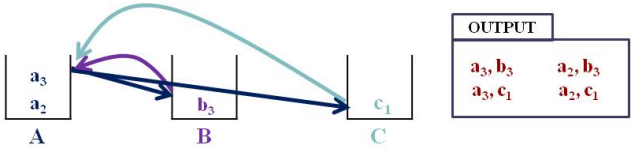
(그림 3) XPruner의 질의 처리 실행

그림 3은 그림 1의 XML 문서와 트윅 질의에 대하여 XPruner가 실행하는 과정을 나타낸다. 예를 들어 <a_1> 시작 이벤트가 발생하였을 때 a_1이 질의의 루트 노드인지를 확인한다. a_1이 주어진 질의의 루트 노드 A와 같기 때문에 스택 A에 삽입한다. <b_1>의 시작 이벤트가 발생하였을 때 b_1은 질의의 루트 노드가 아니기 때문에 부모 스택 A의 a_1과의 관계가 주어진 질의와 맞는지 확인한다. b_1은 조건을 만족하기 때문에 스택 B에 삽입된다. </b_1> 끝 이벤트가 발생하였을 때 b_1은 질의의 말단 노드와 일치하기 때문에 스택에 계속 유지한다. 이와 같이 진행 후 마지막에 </a_1> 끝 이벤트가 발생하게 되면 스택 A에서 a_1이 제거되고 a_1을 참조하고 있는 b_1, b_2 역시 스택 B에서 제거된다. 그림 4(a)는 </a_1> 끝 이벤트가 발생하기 전의 스택 상태이고 그림 4(b)는 끝 이벤트가 발생하여 알고리즘이 실행된 스택 상태이다.



(그림 4) 알고리즘 진행에 따른 스택 상태

<c_1>의 시작 이벤트가 발생하게 되면 모든 스택안에는 최소 하나 이상의 노드가 존재하게 된다. 따라서, 주어진 질의에 맞는 답 <a_3, b_3, c_1>과 <a_2, b_3 c_1>을 반환 하고 말단 노드 b_3와 c_1은 각 스택에서 삭제된다. 그림 5는 <c_1> 시작 이벤트가 발생했을 때의 스택 상태와 트윅 질의의 결과를 반환하고 있다.



(그림 5) 알고리즘 결과값 반환

마지막으로 </a_3>와 </a_2> 끝 이벤트가 발생하면 스택 A의 노드 a_3와 a_2가 삭제된다. 결론적으로, 그림 3에서 보여지는 것처럼 XPruner는 이벤트가 발생한 노드에 관해서만 필요한 알고리즘만을 실행하기 때문에 StreamTX 보다 불필요한 시간을 줄일 수 있다.

4. 성능평가

XPruner의 성능 평가를 위해서 본 논문에서는 실제 XML 데이터인 DBLP와 TreeBank를 사용하였다[15]. DBLP와 TreeBank의 데이터 특징은 표 1과 같으며 실험에 사용된 질의는 표 2와 같다. 각 데이터에 대해 각각 3개의 트윅 질의를 사용하였으며 StreamTX와 질의 처리

성능을 비교하였다. 또한, 본 실험에서는 StreamTX와 마찬가지로 XPruner에서도 최대 메모리 허용량을 5M로 할당하였다. XPruner의 메모리 점유율은 StreamTX와 거의 유사한 성능을 가지기 때문에 메모리 사용 측면에서는 최대 메모리를 5M로 할당하여도 XPruner 또한 주어진 트윅 질의를 처리하는데 문제가 없다

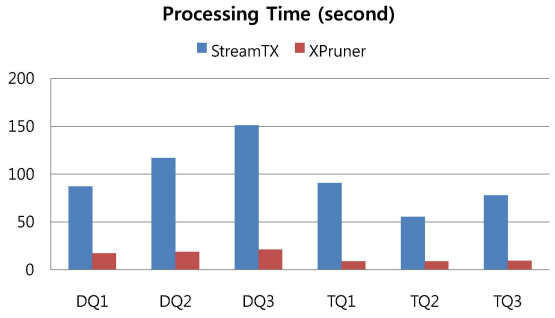
<표 1> 실험에서 사용한 XML 데이터

	Data size (MB)	Nodes (million)	Max. / Avg. depth
DBLP	130	3.3	6/2.9
TreeBank	84	2.4	36/7.8

<표 2> 실험에서 사용한 트윅 질의

	트윅 질의
DQ1	/dblp/inproceedings[title#]/author#
DQ2	/dblp/inproceedings[title# and booktitle#]/author#
DQ3	/dblp/inproceedings[title# and booktitle# and year#]/author#
TQ1	//S[.//VP[.//JJ][.//VBD]]//NP[.//WP]//DT#
TQ2	//S//NP[.//IN][.//VBN]//JJ#
TQ3	//S[.//NP[.//DT][.//NN]]//PP[.//TO]//NN#

최대 허용 메모리를 동일하게 주었지만 그림 6에서 보는 바와 같이 본 논문에서 제안한 XPruner가 StremTX보다 DBLP 데이터에 관해서 5-7배 빠른 질의 처리 성능을 보이며 TreeBank 데이터에 관해서는 약 6-10배 가량 빠른 질의 처리 성능을 보인다. 이를 통해 노드의 이벤트 발생시마다 반복적으로 질의 처리 알고리즘을 호출하는 StreamTX의 경우 불필요한 질의 처리 과정이 많이 포함되어 있음을 알 수 있으며 XPruner의 경우 이러한 불필요한 질의 처리 과정의 제거를 통해 그 성능이 향상되었음을 알 수 있다.



(그림 6) 트윅 질의 처리시간 측정결과

5. 결론 및 향후 연구

본 논문에서는 스트리밍 XML 데이터에 대해 트윅 질의를 빠르게 처리하기 위한 XPruner를 제안하였다. XPruner는 스트림을 저장하고 질의를 처리함에 있어 스택구조를 기반으로 각 스트림의 노드 이벤트에 따라 불필요한 노드들을 효과적으로 제거하는 특성을 가진다. 이로 인해 StreamTX와 유사한 메모리 점유율을 가지면서도 불필요한 질의 처리 과정을 줄임으로써 질의 처리 시간에 있어 효과적임을 성능평가를 통해 검증하였다.

향후 연구로는 스트리밍 XML 데이터의 갱신을 고려한 색인기법과 이에 기반한 질의의 갱신결과를 실시간으로 처리할 수 있는 스트림 처리 기법을 연구할 것이다.

참고문헌

[1] XML path language (XPath). http://www.w3.org/TR/xpath20.

[2] XML Query language (XQuery). http://www.w3.org/XML/Query.

[3] Wook-Shin Han, Haifeng Jiang, Howard Ho, and Quanzhong Li. “StreamTX: extracting tuples from streaming XML data,” VLDB Endowment, 1(1):289-300, 2008.

[4] N. Bruno, N. Koudas, and D. Srivastave. “Holistic twig joins: optimal xml pattern matching,” SIGMOD, pp. 310-321, 2002.

[5] S. Al-Khalifa, H. V. Jagadish, N. Koudas, J. M. Patel, D. Srivastava, and Y. Wu. “Structural joins: A primitive for efficient xml query pattern matching,” ICDE, pp. 141-152, 2002.

[6] J. McHugh and J. Widom. “Query optimization for xml,” VLDB, pp. 315-326, 1999.

[7] T. Milo and D. Suciu. “Index structures for path expressions,” ICDT, pp. 277-295, 1999.

[8] H. Jiang, H. Lu, and W. Wang. “Efficient processing of XML twig queries with or-predicates,” SIGMOD, pp. 59-70, 2004.

[9] H. Jiang, W. Wang, H. Lu, and J. X. Yu. “Holistic twig join on indexed xml documents,” VLDB, pp. 273-284, 2003.

[10] J. Lu, T. W. Ling, C.-Y. Chan, and T. Chen. “From region encoding to extended dewey: on efficient processing of xml twig pattern matching,” VLDB, pp. 193-204, 2005.

[11] F. Peng and S. S. Chawathe. “XSQ: A streaming XPath engine,” TODS, 30(2):577-623, 2005.

[12] V. Josifovski, M. Fontoura, and A. Barta. “Querying XML streams,” VLDB 14(2):192-210, 2005.

[13] D. Florescu, C. Hillery, D. Kossmann, P. Lucas, F. Riccardi, T. Westmann, M. J. Carey, A. Sundararajan, and G. Agrawal. “The BEA/XQRL streaming XQuery processor,” VLDB, pp. 997-1008, 2003.

[14] C. Koch, S. Scherzinger, N. Schweikardt, and B. Stegmaier. “Schema-based scheduling of event processors and buffer minimization for queries on structured data streams,” VLDB, pp. 228-239, 2004.

[15] University of washington XML repository. http://www.cs.washington.edu/research/xmldataset.