제36회 한국정보처리학회 추계학술발표대회 논문집 제18권 제2호 (2011. 11)

스트리밍 XML 상에서 트윅 질의 처리를 위한 패턴 매칭 프루닝과 재사용성 감지 기법*

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

Pattern-matching Pruning and Reusability Detection for Twig Query Processing on Streaming XML Data

Sang-Hyun Park, Byung-Gul Ryu, Da-Oun Jung, SangKeun Lee
Division of Computer and Communication Engineering, Korea University

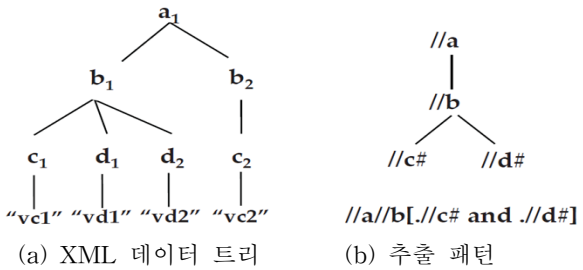
요 약

스트리밍 XML 데이터로부터 트윅 패턴 추출시 질의와 무관한 스트리밍 데이터를 프루닝함으로써 질의 처리 비용을 줄일 수 있어야 한다. 이때 작은 버퍼 사이즈를 유지하면서도 질의 매칭 과정을 최소화하는 것이 필요하다. 본 논문에서는 이를 위한 (1) 패턴 매칭 프루닝과 (2) 재사용성 감지 기법을 제안한다. 기존 기법과 비교하여 제안하는 기법은 스트리밍 데이터의 엘리먼트 이벤트, 버퍼상태 그리고 트윅 패턴을 고려하여 질의 매칭 과정을 최소화한다. 실험결과를 통해 제안기법이 기존 기법보다 우수한 성능을 나타냄을 보인다.

1. 서론

스트리밍 XML 데이터로부터 트윅 질의 패턴에 매칭되는 튜플을 추출하는 것은 맵핑 기반 XML 변환[1]과 같은 XML 기반 어플리케이션의 핵심 기술이다. 대표적인 연구로서 기존 StreamTX[2] 연구에서는 여러 추출 노드들을 가진 XML 패턴 질의를 고려하였다. StreamTX는 스트리밍 XML을 처리하기 위해 기존의 holistic 매칭 기법[3]을 기반으로 block-and-trigger 기법과 프루닝 기법을 제안하였으며 고정된 버퍼 사이즈를 유지하면서도 트윅 질의에 대한 튜플 추출시 성능이 향상됨을 보여주었다.

그러나, 입력되는 엘리먼트 이벤트에 대하여 StreamTX의 메인 알고리즘인 *GetNextStream*이 버퍼상에 입력된 스트리밍 엘리먼트가 매칭되는 튜플로서 확인되지 않은 경우에도 재귀적으로 반복해서 호출된다.



(a) XML 데이터 트리 (b) 추출 패턴

(그림 1) XML 데이터와 추출 패턴

그림 1(b)는 c와 d의 두 필드를 그림 1(a)의 XML 데이터로부터 추출하기 위한 패턴을 보여주고 있다. SE(a1)으로 표현되는 a1 엘리먼트의 시작 이벤트가 발생하고 a1만이 a 엘리먼트 큐에 저장되었을 때, StreamTX는 모든 쿼리 노드(a, b, c 그리고 d)를 해당 엘리먼트 큐에서 찾기 위해 알고리즘을 재귀적으로 호출한다. 이때 b, c 그리고 d 엘리먼트의 큐는 현재 비어있기 때문에 불필요한 매칭 과정이 발생하게 된다. 이러한 문제를 해결하기 위해 본 논문에서는 제안하는 기법의 기여도는 아래와 같다.

- 스트리밍 XML 데이터 상에서 트윅 질의 처리를 위한 (1) 패턴 매칭 프루닝과 (2) 재사용성 기법을 제안한다.
- 실험결과를 통해 기존 기법보다 트윅 질의에 매칭되는 튜플을 빠르게 추출할 수 있음을 보인다.

2. 제안기법

본 절에서는 제안하는 두 기법에 대하여 기술한다. 지면 관계상 전체 세부 알고리즘 및 이에 대한 설명을 생략하고 실제 수행 예제를 통해 각 기법을 설명한다.

2.1 패턴 매칭 프루닝

*패턴 매칭 프루닝*은 버퍼상의 엘리먼트 혹은 입력되는 엘리먼트를 프루닝함으로써 질의 매칭 과정을 최소화하기 위한 기법이다. 버퍼상의 스트림 엘리먼트의 사이즈를 최소화하기 위해 엘리먼트간의 관계를 *양방향 포인트*를 사용한다. 양방향 포인트는 각 엘리먼트에 해당하는 버퍼간에 부모-자식, 자식-부모관계에 대한 포인트로 구성되며 이러한 포인트들은 입력되는 엘리먼트들과 불필요한 후손

* 이 논문은 2011년도 정부(교육과학기술부)의 재원으로 한국연구재단의 기초연구사업 지원을 받아 수행된 것임(2011-0010325).

엘리먼트들을 프루닝하기 위하여 사용된다.

예를 들어 XML 데이터 D1 : <a1><b1>vb1</b1><b2>vb2</b2></a1>과 질의 Q1://a[.//b#]//c# 에 대하여 질의 노드들에 대한 스택을 S_a, S_b 그리고 S_c로 각각 유지하여 입력되는 스트림 엘리먼트를 버퍼링한다. SE(a1)과 SE(b1)이 순차적으로 도착하며, a1과 b1은 S_a와 S_b에 각각 버퍼링 된다. S_b상의 엘리먼트 b1은 S_a상의 a1에 대한 부모-자식 포인트를 생성한다. 이때 질의에 따르면 노드 a는 b와 c의 공통 선조 노드여야 한다. 그러나 EE(a1)으로 표현되는 a1의 엘리먼트의 끝 이벤트가 발생할 때, S_a상의 a1은 질의의 c노드에 대한 포인터를 가지고 있지 못하며 오직 b노드들 (S_b상의 b1 그리고 b2)에 대한 포인터만을 가지고 있게 된다. 따라서, b1, b2 그리고 a1은 S_a와 S_b로부터 안전하게 프루닝될 수 있다.

또 다른 예제로서 XML 문서 D2 : <a1><b1>vb1</b1><c1>vc1</c1></a1>와 동일한 질의 Q1을 고려하자. 스트림상으로 SE(a1), SE(b1), EE(b1), SE(c1), EE(c1) 그리고 EE(a1)의 이벤트를 순차적으로 수신하게 된다. SE(c1)이 도착한 후에 c1은 S_c에 버퍼링되고, c1은 a1에 대한 자식-부모 포인트를 생성한다. 이때 a1은 이미 b1에 대한 포인터를 가지고 있기 때문에 질의 Q1에 대한 매칭 튜플인 [a1, b1:vb1, c1:vc1]이 결과로 반환된다. 그리고 b1과 c1은 S_a와 S_b로에서 제거된다. 이때 일단 결과로 반환된 질의상의 말단노드들은 안전하게 제거될 수 있다. 마지막으로 EE(a1)이 도착하였을때 a1은 S_a로부터 제거된다.

2.2 재사용성 감지

재사용성 감지 기법은 매칭 튜플로서 사용되어 버퍼상에서 이미 제거된 엘리먼트들이 추후 또 다른 매칭 튜플 추출시 여전히 재사용 되는지를 감지하기 위한 기법이다. 이 기법은 버퍼의 사이즈를 줄이고 질의 처리 시간을 향상시키는데 사용되어진다. 구체적으로 *조건부 플래그*를 사용하여 현재 입력된 스트림 엘리먼트가 이미 버퍼상에서 제거된 매칭 튜플들과 매칭되어 또 다른 매칭 튜플이 될 수 있는지를 체크한다. 그림 1(a)에서 SE(d1)이 도착하였을때 S_a, S_b, S_c 그리고 S_d는 이미 a1, b1, c1 그리고 d1을 각각 포함하고 있다. 따라서 이러한 엘리먼트들은 양방향 포인트를 사용하여 매칭 튜플로서 반환된다. 그리고 c1과 d1은 해당 스택으로부터 제거된다.

그러나 SE(d2)가 도착했을 때, d2가 매칭되는 튜플인지 판단할 수가 없다. 왜냐하며 c1이 이미 해당 스택으로부터 제거되었기 때문이다(실제로 d2는 매칭 튜플중의 하나이다). 이러한 현재의 스트림 시작 이벤트가 매칭 튜플에 포함되는지를 체크하기 위하여 *True/False/Check*와 같은 조건부 플래그를 사용한다. 그림 2는 그림 1의 XML 데이터에서 엘리먼트 이벤트에 따른 플래그 변화를 보여준다.

만약 버퍼상의 엘리먼트들이 매칭 튜플로서 사용되어지지 않았다면, *False* 플래그가 기본적으로 할당된다. 그림 2에서 SE(a1)이 도착할 때, a1은 *False*를 가진다. SE(b1)이 도착한 후에 b1은 a1으로부터 *False*를 상속받는다. 유사하게 c1과 d1은 b1으로부터 *False*를 상속받는다.

F : False, T : True, C : Check

	$SE(a_1)$	$SE(b_1)$	$SE(c_1)$	$EE(c_1)$	$SE(d_1)$	$EE(d_1)$	$SE(d_2)$
a_1	F	F	F	F	F → T	T	T
b_1	-	F	F	F	F → T	T	T
c_1	-	-	F	F	F → -	-	-
d_1	-	-	-	-	F → -	-	-
d_2	-	-	-	-	-	-	T
b_2	-	-	-	-	-	-	-
c_2	-	-	-	-	-	-	-
	$EE(d_2)$	$EE(b_1)$	$SE(b_2)$	$SE(c_2)$	$EE(c_2)$	$EE(b_2)$	$EE(a_1)$
a_1	T	T	T	T	T	T	-
b_1	T	-	-	-	-	-	-
c_1	-	-	-	-	-	-	-
d_1	-	-	-	-	-	-	-
d_2	-	-	-	-	-	-	-
b_2	-	-	C	C	C	-	-
c_2	-	-	-	C	C	-	-

(그림 2) 그림 1에서 스트림 이벤트에 대한 플래그 변화

버퍼상의 엘리먼트들 중 하나가 매칭 튜플로서 사용되면 해당 엘리먼트의 플래그는 *True*로 변한다. SE(d1)이 도착했을때 d1은 우선 *False* 를 상속 받은 뒤 매칭 튜플인 c1과 d1이 반환되고 스택으로부터 제거된다. 그리고 a1과 b1의 플래그는 *True*로 바뀐다(그림 2에서 F→T에 해당한다). SE(d2)가 도착할 때, d2는 b1으로부터 *True*를 상속받는다. d2가 *True* 플래그를 가짐으로 인해서 d2는 이미 반환된 매칭 튜플인 [a1, b1, c1:vc1]과도 매칭되는 것을 확신할 수 있게 된다. 따라서 기존에 반환된 튜플을 재활용하여 또다른 매칭 튜플인 [a1, b1, d2:vd2]가 반환될 수 있다.

만약 현재의 스트림 노드의 시작 이벤트가 질의의 브랜칭 노드이고 스트림 노드의 부모 엘리먼트의 플래그가 *True*인 경우에 현재 스트림 노드에는 *Check* 플래그가 할당된다. 이러한 *Check* 플래그가 할당될 경우는 이후에 입력되는 스트림 이벤트를 더 살펴보아야 함을 의미한다. 따라서 SE(b2)가 도착할 때 b2의 플래그는 *Check*로 할당된다. 엘리먼트 c2는 b2로부터 *Check*를 상속받는다. EE(b2)가 도착하게 되면 위의 예제에서 b2는 더 이상 질의 결과에 해당되는 노드인, d를 자손으로 가지고 있지 않음을 알 수 있게 되며 이때 b2와 c2는 해당 스택으로부터 안전하게 제거될 수 있다.

위의 예제에서와 같이 질의와 매칭되는 결과값은 질의에 존재하는 모든 노드의 SE가 스트림으로부터 도착하였을 때 반환되고 질의 노드의 EE가 도착하였을 때 이전까지의 스택상에 존재하는 엘리먼트들 중 질의와 무관한 엘리먼트들이 스택상의 플래그와 양방향 포인트를 통해 제거된다.

3. 성능평가

제안한 기법의 성능 평가를 위하여 PR-Pruner, StreamTX[2], TwigStackTX[2]의 세 가지 알고리즘을 Visual C++ 프로그램 언어로 구현하였으며 PR-Pruner는 본 논문에서 제안하는 기법이다. 그리고 스트림을 저장하기 위한 버퍼 사이즈의 경우 기존 StreamTX의 경우와 마

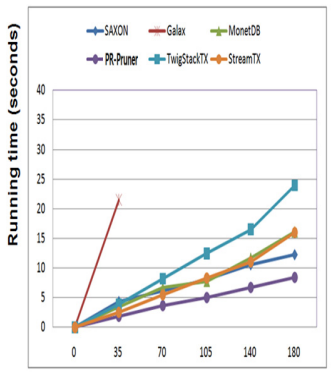
찬가지로 5MB로 고정하였다. 그리고 기존에 공개된 XQuery엔진들(SAXON[4], Galax[5], MonetDB[6])과 PR-Pruner의 성능을 비교하였다. 모든 실험은 2.66Ghz의 듀얼코어 인텔 i5 CPU와 2GB의 메모리를 가진 시스템과 윈도우7 운영체제에서 수행되었다.

실험을 위해 다양한 크기의 DBLP[7], Treebank[7] 그리고 XMARK[8] 데이터를 사용하였으며 각 데이터에 대하여 2개의 트윅 질의로서 성능평가를 수행하였다. 표 1에서 각 질의에 대해 기술하였다.

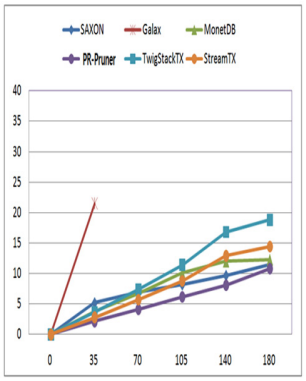
그림 3에서 각 XML 크기 변화에 따른 트윅 질의 성능 결과를 나타내고 있다. 제안 기법인 PR-Pruner는 데이터 크기가 커짐에 따라 기존의 기법보다 안정적인 성능을 보여주었다. Galax의 경우 모든 실험에서 주어진 질의의 결과를 출력하지 못하고 종료되었다.

<표 1> 실험에서 사용한 트윅 질의

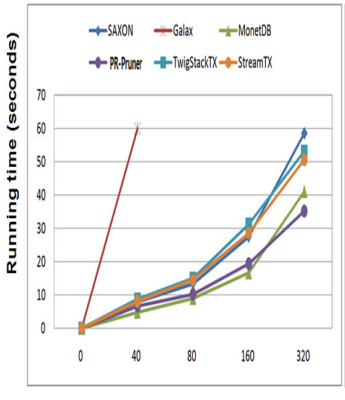
	트윅 질의
QD1	/dblp/inproceedings[title#]/author#
QD2	/dblp/inproceedings[title# and booktitle#] /author#
QT1	//S[.//VP[.//JJ][.//VBD]]//NP[.//WP]//DT#
QT2	//S//NP[.//IN][.//VBN]//JJ#
QM1	//item[./name]/mailbox/mail[./to]/text[./bold] /emph/bold$
QM2	//item[./payment][./quantity][./shipping][./mailbox /mail/text]/description/parlist//text$



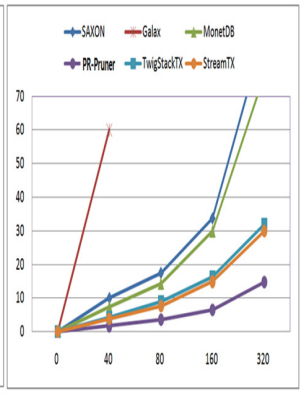
(a) 질의 QD1



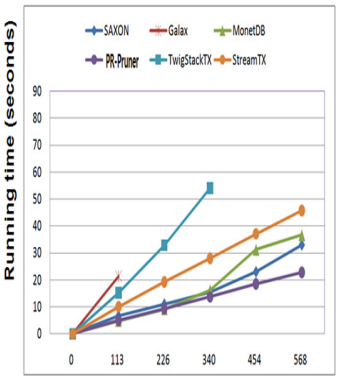
(b) 질의 QD2



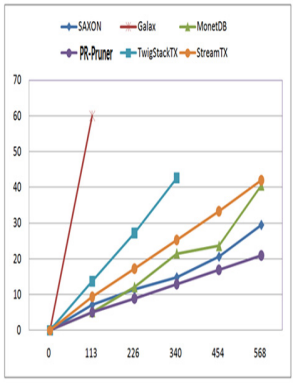
(c) 질의 QT1



(d) 질의 QT2

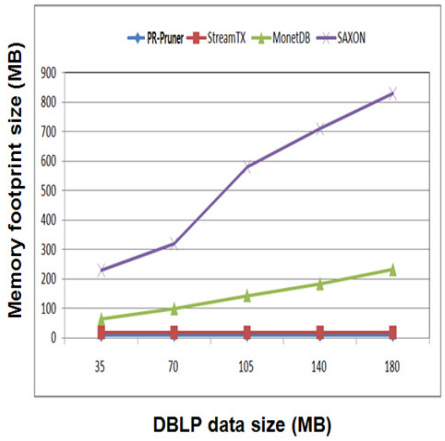


(e) 질의 QM1



(f) 질의 QM2

(그림 3) XML 크기 변화에 따른 트윅 질의 성능 비교



(그림 4) 메모리 사용량 측정 (질의 : QD1)

그림 4는 DBLP 데이터 크기에 따라 질의 QD1을 처리하는데 점유되는 메모리 사용량을 보여주고 있다. SAXON과 MonetDB의 경우 주어진 질의를 처리함에 있어 주어진 XML 데이터 크기 이상으로 메모리를 점유하기 때문에 스트리밍 환경에서 사용하기에는 부적합함을 알 수 있다. 다른 데이터 및 질의에 대해서도 그림 4와 유사한 결과를 보여주었다.

<표 2> StreamTX와 TwigStackTX에 대한 PR-Pruner의 평균 수행시간 비율

XML 데이터	StreamTX / PR-Pruner	TwigStackTX / PR-Pruner
DBLP	1.6	2.4
Treebank	2	1.7
XMARK	2.5	2

표 2는 그림 3의 실험결과를 토대로 StreamTX와 TwigStackTX에 대한 PR-Pruner의 상대적인 수행시간 비율을 요약한 것으로서 PR-Pruer가 기존 기법에 비해 약 2배정도 빠른 수행시간을 보여줌을 알 수 있다. 이러한 결과가 나타난 이유는 기존 기법의 경우 질의 노드들에

매칭되는 입력 스트림의 엘리먼트에 대해 반복적으로 재귀적인 알고리즘을 수행하기 때문에 PR-Pruner와 비교할 때 불필요한 질의 매칭 과정이 발생하기 때문이다.

4. 관련연구

최근 스트리밍 XML 데이터 처리에 관한 다양한 연구가 수행되었다[2,9,10]. XSQ 시스템의 경우 주어진 질의에 대하여 여러 매칭 튜플이 아니 단일 매칭 튜플을 추출하는 연구를 수행하였다[9]. TurboXPath 는 수행시간 및 디스크 입출력에서 XSQ보다 효과적인 성능을 가짐을 보여주었다[10]. StreamTX[2]는 스트리밍 XML 데이터로부터 여러 매칭 튜플들을 추출하기 위한 가장 최근의 연구로서 작은 버퍼 사이즈를 유지하면서도 주어진 트윅 질의의 패턴들을 추출할 수 있음을 보여주었다. 그러나 스트리밍 엘리먼트를 처리하는 재귀적인 알고리즘으로 인해 여전히 불필요한 질의 처리 시간이 발생하였다. 또한 PR-Pruner의 선행 연구로서 버퍼로 사용되는 스택내의 노드 프루닝 기법을 제안한 바 있다[11]. PR-Pruner는 기존 제안 기법을 기반으로 노드 프루닝이 아닌 질의에 대한 패턴 매칭 프루닝 기법으로 확장되었다. 그리고 기존 선행 연구의 경우 버퍼내에서 질의에 추후 재사용 되어야 하는 노드를 계속 유지해야 하는 문제가 있었으나 본 논문에서는 재사용 감지 기법을 새롭게 제안함으로써 조건부 플래그를 통해 버퍼내에서 이미 제거된 노드와 버퍼내에 입력된 노드와의 매칭 여부를 판별할 수 있다.

5. 결론 및 향후연구

본 논문에서는 스트리밍 XML 데이터로부터 트윅 질의에 매칭되는 여러 튜플을 추출하기 위한 PR-Pruner 기법을 새롭게 제안하였다. 입력되는 스트리밍 데이터와 질의 노드간의 매칭 과정을 최소화하기 위하여 첫 번째로 *양방향 포인트*를 사용하여 스트리밍 엘리먼트를 프루닝하는 *패턴 매칭 프루닝* 기법을 제안하였으며 두 번째로 *조건부 플래그*를 사용하여 버퍼 크기를 줄이면서 질의 수행 시간을 향상시키기 위한 *재사용성 감지* 기법을 제안하였다. 실험결과를 통해 PR-Pruner가 기존 기법에 비해 수행시간에 있어 효율적임을 검증하였다. 향후 스트리밍 XML 데이터에 대한 키워드 검색을 수행하기 위한 연구를 수행할 예정이다.

참고문헌

[1] Haifeng Jiang, Howard Ho, Lucian Popa, and Wook-Shin Han., "Mapping-driven XML transformation," In Proceedings of the 16th international conference on World Wide Web, pp. 1063-1072, 2007.

[2] Wook-Shin Han, Haifeng Jiang, Howard Ho, and Quanzhong Li., "StreamTX: extracting tuples from streaming XML data," In Proceeding of VLDB Endowment, 1(1), pp. 289-300, 2008.

[3] Nicolas Bruno, Nick Koudas, and Divesh Srivastava., "Holistic twig joins: optimal XML pattern matching," In Proceedings of the 2002 ACM SIGMOD international conference on Management of data, pp.310-321, 2002.

[4] The SAXON XSLT and XQuery Processor, http://saxon.sourceforge.net/.

[5] Galax, http://www.galaxquery.org/.

[6] MonetDB, http://www.monetdb-xquery.org/.

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

[8] XMark : an XML Benchmark Project, http://www.xml-benchmark.org/

[9] Feng Peng and Sudarshan S. Chawathe., "XSQ: A streaming XPath engine," ACM Trans. Database Syst. 30(2):577-623, 2005.

[10] Vanja Josifovski, Marcus Fontoura, and Attila Barta., "Querying XML streams," The VLDB Journal, 14(2):197-210, 2005.

[11] Byung-Gul Ryu, Sang-Hyun Park, Jong-Woo Ha, and SangKeun Lee, "Fast Twig Query Processing for Streaming XML Data," In Proceedings of the 34th KIPS Fall Conference, vol. 17, no. 1, pp. 65-68 2010.