Journal of Digital Contents Society Vol. 13 No. 1 Mar. 2012(pp. 21-29)

능동적 슬라이딩 윈도우 기반 빈발구조 탐색 기법

황정희*

요 약

최근 인터넷의 급격한 발전과 유비쿼터스 컴퓨팅 환경 그리고 센서 네트워크와 같은 많은 정보들의 교환이 이루어지는 환경에서 연속적으로 전송되는 데이터에 대한 처리가 요구되고 있다. 이와 관련하여 XML 스트림 데이터에 대한 빈발구조 추출 및 효율적인 질의처리를 위한 마이닝 방법들이 연구되고 있다. 본 논문에서는 연속적으로 빠르게 발생하는 스트림 데이터로부터 유용한 정보를 발견하기 위한 기반 연구로써 트리거를 이용한 슬라이딩 윈도우 기반의 XML 빈발구조 탐색 방법을 제안한다. 제안된 방법은 스트림 데이터에 대한 마이닝과 연속질의 처리 등을 위해 트리거를 이용하여 데이터의 흐름을 자동으로 제어할 수 있는 기반이 된다.

A Method of Frequent Structure Detection Based on Active Sliding Window

Jeong Hee Hwang

Abstract

In ubiquitous computing environment, rising large scale data exchange through sensor network with sharply growing the internet, the processing of the continuous stream data is required. Therefore there are some mining researches related to the extracting of frequent structures and the efficient query processing of XML stream data. In this paper, we propose a mining method to extract frequent structures of XML stream data in recent window based on the active window sliding using trigger rule. The proposed method is a basic research to control the stream data flow for data mining and continuous query by trigger rules.

Keywords : XML Stream, XML Mining, Frequent Structure, Trigger

1. 서론

스트림 데이터 마이닝은 데이터 스트림에 대한 실시간 분석 능력을 지원한다. 다시 말해, 어느 시점에서나 해당 시점까지의 모든 트랜잭션을 포함하는 현재 데이터 스트림에 대한 마이닝 결과를 얻을 수 있도록 지원한다. 이러한 스트림 데이터 마이닝은 최대 한번 탐색하고, 메모리 사용량은 무한히 증가되지 않으며 최신의 데이터에 대한 빠른 분석을 제공하는 것을 목적으로 한다[1-4].

XML은 어떤 서비스든지 그 서비스를 기술하는 방법을 제공하는 유연성이 있으므로 유비쿼터스 환경의 상황정보 인식 부분을 구축하는데 중요한 매체로 사용될 수 있다. 현재 유비쿼터스 환경을 위한 기초 데이터로써 XML 스트림 데이터(stream data)의 사용이 일반화되고 있다[5-7]. 그러므로 XML 데이터로부터 빈발 구조를 탐색하는 기존의 연구[8]을 기반으로 스트리밍 XML 데이터의 특성을 고려하는 빈발 구조의 탐색방법에 대한 연구가 필요하다.

따라서 본 논문에서는 시간에 따라 변화되는 최신 XML 스트림 데이터로부터 유용한 정보를 효율적으로 탐색하기 위한 방법을 제안한다. 이를 위해 기존 스트림 마이닝 기법을 확장하여 XML 특성을 고려하는 트리거에 의한 슬라이딩

※ 제일저자(First Author) : 황정희
접수일:2011년 12월 20일, 수정일:2012년 01월 17일
완료일:2012년 02월 29일
* 남서울대학교 컴퓨터학과
jhhwang@nsu.ac.kr
▣ 이 논문은 2011년도 남서울대학교 학술연구비 지원에 의해 연구되었음

윈도우 기반의 마이닝 기법을 이용한다. 스트리밍 XML 데이터에 대한 빈발 구조 탐색은 스트림 데이터에 대한 효율적인 질의 처리 및 색인 구성에 효율적으로 이용될 수 있다.

트리거는 능동 데이터베이스 관리 시스템에서 사용되는 개념으로 능동 데이터베이스는 계속해서 데이터베이스의 상태를 모니터하고 미리 정의된 사건이 발생했을 때 스스로 반응한다. 그리고 데이터베이스 사건에 의해 트리거되는 조건들을 모니터 하다가 만약 그 조건이 만족되면 연관된 행위를 수행한다. 즉, 능동규칙은 이벤트 중심으로 처리되는 경우에 매우 효과적으로 처리할 수 있다는 장점이 있으므로 스트림 데이터의 자동적인 흐름 제어에 효율적이다. 본 논문에서는 트리거 규칙을 이용한 능동적 슬라이딩 윈도우 기반으로 현재 윈도우 범위에 속하는 XML 데이터 집합을 트리 모델, XFP_tree(XML Frequent Pattern Tree)로 구성하고 이를 이용하여 빈발구조를 추출한다. XFP_tree는 각 XML 데이터로부터 일정 임계값 이상의 빈발 구조 집합을 표현하는 트리 구조이고, 윈도우 이동시점에서 삭제되는 트랜잭션의 XML 트리에 해당하는 노드들을 일괄적인 이동에 의해 빈발도를 갱신하므로 현재 윈도우의 최신 데이터에 집중하여 빈발 구조를 탐색할 수 있다.

본 논문의 구성은 다음과 같다. 2장에서는 관련 연구에 대해 기술하고 3장에서는 슬라이딩 윈도우에 적용되는 트리거 규칙 및 마이닝 수행에 필요한 기본 개념들을 정의한다. 그리고 4장에서는 스트림 데이터를 표현하는 XFP_tree의 구성 및 빈발 구조를 탐색하는 방법을 설명한다. 5장에서는 실험을 통해 제안된 방법을 평가한다. 마지막으로 6장에서 결론을 맺는다.

2. 관련연구

XML 스트림 데이터의 구조 발견을 위한 기법들이 [5,6.9]에서 제안되었다. 많은 질의들이 비슷한 구조를 공유한다는 사실을 기반으로 연속 질의를 점증적으로 그룹핑 하는 방법을 [5]에서 제안하였다. 이 방법은 새로운 데이터가 도착했을 때 실행되는 질의나 특정 실행 간격을 지니는 질의 등의 두 가지 경우 모두를 지원하여 XML 문서 스트림에 대한 다중 연속 질의 최적화에 초점을 두고 있다. [6]은 XML에 대한 빈발 질의 패턴을 발견하기 위해 Apriori 기반의 후보 패턴을 생성하여 이용한다. 또한 XML 스트리밍 데이터로부터 빈발 트리를 분석하기 위해 가상 트리에서 오른쪽으로 노드를 확장하는 방법을 이용하는 온라인 알고리즘, StreamT을 [7]에서 제안하였다. 그러나 정확한 결과를 보장하기 어렵고 확장 가능한 노드의 경우를 모두 고려하기 때문에 많은 메모리가 필요하다. STMer 알고리즘[9]은 StreamT과 유사한 연구로써, 단말노드를 확장하는 방법을 적용하여 스트림으로 입력되는 노드들을 레벨과 레이블에 따라 공통의 prefix tree를 생성하여 노드들을 확장시키고 빈발한 구조패턴을 탐색한다.

스트림 데이터는 연속적이며 크기가 무한하다. 그러므로 스트림 데이터 처리는 기존 데이터베이스 시스템에서의 데이터 처리와는 다르게 축약 및 요약하여 처리되고 있다. [2]에서는 센서 노드에서 전송되는 스트림 데이터의 크기를 줄이기 위하여 데이터를 집계하는 알고리즘을 제안하였다. 스트림 데이터 정보를 트리 형태로 구축하고, 집계 연산을 통하여 요약함으로써 에러를 줄이고 전송되는 메시지 수를 줄인다. [1]은 최소 지지도를 만족하는 아이템 셋을 계산하고, 결과 중에는 사용자가 허락하는 범위 내의 오차가 있는 결과도 포함될 수 있도록 하였다. 그러나 이러한 연구들[1-3]은 구조적 특성이 있는 XML 스트림 데이터에 직접 적용하기는 어렵다.

본 논문에서는 XML 문서에 대한 빈발 구조를 탐색하기 위하여 XFP_tree를 이용한다. [10]은 일괄적으로 처리하는 데이터를 대상으로 점진적으로 마이닝을 하기 위해 트리를 이용하였다. 그리고 구조 없는 데이터에 적용하기 위해 [11]에서는 DSTree를 제안하였다. 본 논문의 XFP_tree는 DSTree와 구조는 유사하지만 구조정보를 포함하는 XML 데이터의 특성을 반영한 트리이다. 그리고 [7,9]에서 현재 시점의 XML 스트림 데이터에 대해 빈발구조를 추출하는 방법을 제시하였으나 본 논문에서 제안하는 트리거에 의한 능동적 슬라이딩 윈도우 기반으로 데이터 변화를 유지하는 방법과는 다르다.

본 논문에서는 능동 데이터베이스의 트리거를 기반으로 하는 슬라이딩 윈도우 기법으로 XML 문서의 빈발 구조를 탐색하는 방법을 제안한다.

트리거를 이용하면 연속적으로 입력되는 스트림 데이터의 자동적인 관리가 가능하며, 이를 기반으로 XML 구조의 특성을 고려한 빈발 구조의 탐색이 가능하다.

3. 기본정의와 트리거 규칙

3.1 기본 정의

본 논문에서는 각 XML 문서를 하나의 트랜잭션으로 고려하며, 윈도우에 포함된 일정한 배치의 수를 윈도우 크기로 고려한다. 마이닝을 위한 기본 정의는 다음과 같다.

정의 1 (배치). 주어진 시간 t 시점에서 일정한 시간 간격 d사이 즉, [t-d+1, t]에 입력되는 트랜잭션의 집합을 배치라 한다. 하나의 배치에 포함된 일정한 트랜잭션의 수를 배치사이즈라 하고, $|B_i|$으로 표현한다. 여기서 B_i는 순차적으로 부여되는 일련의 배치를 의미한다.

정의 2 (윈도우 사이즈). 윈도우 w에는 일정한 수의 배치들이 포함되며, 포함되어 있는 배치의 수를 윈도우 사이즈라 하고, |w|로 표현한다.

정의 3 (빈발구조). 배치 사이즈 $|B_i|$에서 루트부터 임의의 노드까지의 패스정보를 나타내는 구조항목 λ를 포함하고 있는 트랜잭션의 수가 주어진 지지도 Θ를 만족하면 λ는 빈발 구조라 한다. 이를 식으로 표현하면 다음과 같다.

$$\frac{|T_{\ni\lambda}|}{|B_i|} \geq \Theta \quad (0<\Theta\leq 1)$$

여기서 $|T_{\ni\lambda}|$는 λ구조항목을 포함하고 있는 트랜잭션의 수이다.

정의 4 (빈발도). XML 스트림 데이터에 대한 구조의 빈발도 $XSup(E_i)$는 XML 트리를 구성하는 XFP_tree에 포함되어 있는 에지 $E_i=<P_i, C_i>$의 발생 빈도이다. 이에 대한 식은 다음과 같이 나타낸다.

$$XSup(E_i) = \frac{|E_i|}{|XFP\_tree|}$$

여기서 |XFP_tree|는 XML 스트림 데이터 트리 XFP_tree에 포함되어 있는 트랜잭션 수를 의미하며, $|E_i|$는 에지의 발생빈도를 의미한다. 에지는 각 XML 데이터를 나타내는 트리에서 유일한 노드의 관계로 구성되며, 주어진 사용자 지지도 $(0<min_sup\leq 1)$를 만족하면 $XSup(E_i)\geq min\_sup$ 이므로 에지 E_i는 빈발하다는 것을 의미한다.

정의 5 (경계구조). 배치 사이즈 $|B_i|$에 대해 λ 구조항목을 포함하고 있는 트랜잭션의 수가 주어진 지지도 Θ를 만족하지 못하는 항목들 중에서 트랜잭션이 추가되면 빈발 지지도를 만족 할 수 있는 빈발 가능 항목을 경계구조라 한다. 이를 식으로 표현하면 다음과 같다.

$$\frac{|T_{\ni\lambda}|+1}{|B_i|} \geq \Theta \ (0<\Theta\leq 1)$$

경계구조는 연속적인 배치의 입력으로 빈발 구조에 포함될 수 있는 가능성이 있는 항목으로, 마이닝 결과에 대한 오차 범위를 줄이기 위해 사용된다.

3.2 트리거 규칙

본 논문에서는 현재 윈도우 범위에 속하는 트랜잭션을 대상으로 빈발구조를 탐색하기 위해 트리거를 이용하여 윈도우를 슬라이딩하는 방법을 제안한다. 즉, 트리거를 이용하여 연속적으로 삽입되는 스트림에 대한 윈도우를 유지한다. 이 과정에서 최초로 스트림 데이터가 삽입되는 워킹 테이블과, 워킹 테이블에서 마이닝을 하기 위한 현재 시점 기준의 데이터를 윈도우 테이블에 삽입한다. 그리고 마이닝 테이블은 윈도우 테이블에서 빈발구조를 탐색하기 위한 데이터를 유지한다.

입력되는 스트림 데이터에 대해 일정한 튜플 수를 기준으로 탐색대상이 되는 데이터를 윈도우 테이블에 자동으로 삽입하는 트리거 규칙 1(Trigger Rule 1)을 정의한다. 워킹 테이블에 데이터 삽입이 발생하면 트리거에 의해 윈도우 테이블로 일정 수의 트랜잭션 묶음인 배치가 삽입된다. 워킹 테이블에 삽입된 트랜잭션에 대한 배치 시퀀스 번호를 순차적으로 부여하여 윈도우 테이블에 삽입한다. 워킹 테이블은 삽입되는 전체 데이터를 유지하는 백업 데이터의 역할도 하고, 필요시 오래된 데이터들은 삭제한다.

```
<Trigger Rule 1>
CREATE OR REPLACE TRIGGER
trigger_wnd_insert
```

```
AFTER INSERT ON wrk_table
FOR EACH ROW
DECLARE
w_SEQ2 NUMBER;
w_fSEQ NUMBER;
w_TRN NUMBER: = 5;
BEGIN
 SELECT (MAX(tn_bseq) + 1) INTO w_SEQ2
 FROM  wnd_table
 IF (w_SEQ2 IS NULL) THEN
   w_SEQ2: = 1;
 SELECT MIN(tn_seq) INTO w_fSEQ
 FROM wrk_table
 GROUP BY tn_flg
 HAVING tn_flg = 0;
 // batch에 속한 트랜잭션 수만큼 이동
 FOR i IN 1..w_TRN LOOP
  INSERT INTO wnd_table
  (tn_id, tn_root, tn_seq, tn_bseq, tn_flg,tn_xml)
  SELECT tn_id, tn_root, tn_seq, w_SEQ2,
          tn_flg, tn_xml
  FROM wrk_table
  WHERE SEQ = w_fSEQ;
//이동한 트랜잭션 flg set
  UPDATE wrk_table
  SET (tn_bseq, tn_flg) = (w_SEQ2, 1)
  WHERE  tn_seq = w_fSEQ;
  w_fSEQ = w_fSEQ + 1;
END LOOP
END
```

트리거 규칙2(Trigger Rule 2)는 윈도우 테이블에 입력되는 배치에 대해 윈도우 사이즈에 적합한 배치만을 유지하도록 하며 가장 오래된 배치는 DELETE 연산으로 삭제한다. 여기서 tn_seq는 트랜잭션의 시퀀스이고, tn_bseq는 배치의 시퀀스 번호를 의미한다. 트랜잭션 시퀀스는 데이터의 삽입에 대한 일련의 번호이고, 배치 시퀀스는 트랜잭션 사이즈 단위의 배치에 대한 일련번호이다.

```
<Trigger Rule 2>
CREATE OR REPLACE TRIGGER
trigger_wnd_delete
AFTER INSERT ON wnd_table
FOR EACH ROW
DECLARE
w_SIZE NUMBER: = 3;
w_minb NUMBE;
w_maxb NUMBE;
BEGIN
SELECT MAX(tn_bseq), MIN(tn_bseq)
        INTO w_maxb, w_minb
FROM wnd_table;
//delete check
IF w_maxb - w_minb >= w_SIZE THEN
   DELETE FROM wnd_table
   WHERE SEQ2 = w_SEQ;
END IF
END
```

트리거 규칙3(Trigger Rule 3)은 윈도우 테이블에 있는 배치 중에서 현재 시점으로 XML 데이터에 대한 빈발 구조를 추출해야 할 마이닝 대상의 배치에 포함될 트랜잭션들을 배치 단위로 삽입하는 트리거 정의이다. 마이닝 테이블은 각 배치에 포함된 트랜잭션으로부터 빈발구조를 발견하기 위한 저장 테이블이다. 그리고 마이닝 테이블에 저장되어 있는 배치로부터 빈발 구조와 경계구조를 추출한다.

```
<Trigger Rule 3>
CREATE OR REPLACE TRIGGER
trigger_min_insert
AFTER INSERT ON wnd_table
FOR EACH ROW
DECLARE
w_fSEQ NUMBER;
BEGIN
//min_table clear
TRUNCATE TABLE min_table;
//min_table 이동 대상 batch 시퀀스
SELECT MIN(tn_bseq) INTO w_fSEQ
FROM wnd_table
GROUP BY tn_flg
HAVING tn_flg = 0;
//min_table 이동
INSERT INTO min_table
  (tn_id, tn_root, tn_bseq, tn_xml)
SELECT tn_id, tn_root, tn_bseq, tn_xml
```

```
FROM wnd_table
WHERE tn_bseq = w_fSEQ;
//min로 이동한 flg = 1 setting
  UPDATE wnd_table
  SET tn_flg = 1
  WHERE  tn_bseq = w_fSEQ;
END
```

본 논문에서는 트리거를 이용하여 빈발구조와 경계구조를 관리한다. 트리거 규칙4(Trigger Rule 4)는 경계구조를 저장하는 border_table를 유지하기 위한 정의이다. 현재 빈발구조는 아니지만 계속되는 트랜잭션의 입력으로 빈발구조가 될 가능성이 있는 항목들을 별도로 관리한다. 그러므로 트리거를 통해 같은 구조항목이 발견되면 해당 항목의 빈발도를 나타내는 tn_cnt 값을 1씩 증가시키고, 오래된 배치의 삭제 및 트랜잭션 수에 대한 경계구조의 임계치를 나타내는 w_TEMP:= w_TSD * w_TRN을 만족하지 않으면 삭제하는 과정을 트리거에 의해 자동으로 관리한다.

```
<Trigger Rule 4>
CREATE OR REPLACE TRIGGER
trigger_brd_insert
AFTER INSERT ON brd_table
FOR EACH ROW
DECLARE
w_TSD NUMBER:= 0.4;
w_TRN NUMBER:=9;
w_TEMP NUMBER:
BEGIN
w_TEMP:= w_TSD * w_TRN;
//항목 count 증가
UPDATE brd_table
SET tn_cnt = tn_cnt + 1
WHERE item_id = :new.item_id;
//항목 유지 & delete
DELETE
FROM brd_table
WHERE tn_cnt < w_TEMP;
END
```

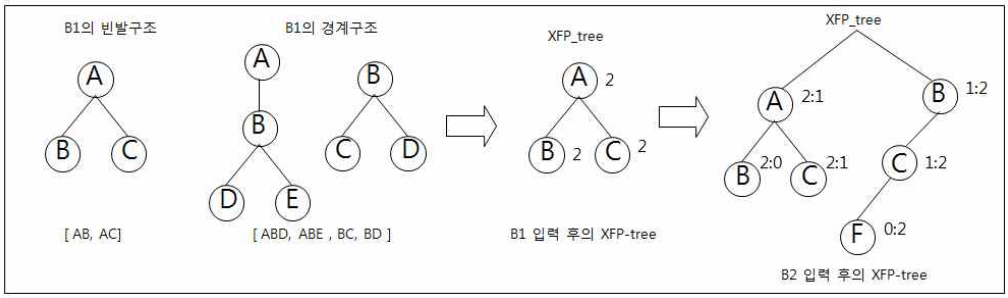
4. 빈발구조 탐색

이 장에서는 트랜잭션 입력에 따른 XFP_tree의 구성 및 XFP_tree로부터 빈발구조를 탐색하는 과정에 대해 설명한다. 트리로 구성되는 XML 데이터 스트림은 연속적으로 입력되므로 무한한 노드들의 시퀀스이다. 스트림 데이터의 빈발구조를 표현하는 트리, XFP_tree는 루트를 가진 서브트리의 집합으로 나타낸다. 루트노드가 같고 부모노드와 자식노드 관계가 같으면 공통의 에지로 표현하며 에지기반의 구조항목에 대해 빈발도가 누적된다. XML의 빈발구조는 루트부터 자신의 노드까지의 패스(XFPath)는 구조를 구별하는 중요한 의미를 갖기 때문에 서브트리의 루트노드가 같을 경우에 결합한다.

하나의 배치에는 일정한 트랜잭션 사이즈로 구성되어 있다. XFP_tree의 빈발도 유지에 대한 예를 보이기 위해 각 배치에 포함되어 있는 트랜잭션을 B_1(t1, t2, t3), B_2(t4, t5, t6), B_3(t7, t8, t9)라 하고, 윈도우 사이즈를 2라 가정한다.

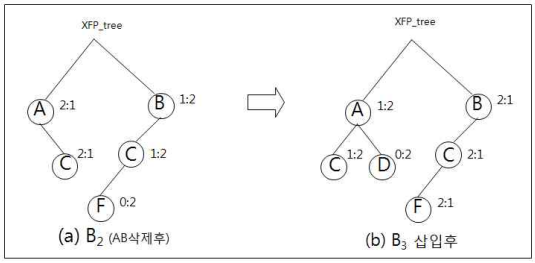
(그림 1)은 순서대로 입력된 T_1, T_2, T_3으로 구성된 B_1에서 min_sup를 0.5로 하여 빈발 구조와 경계구조(border), XFP_tree, 그리고 삽입된 B_2의 빈발구조를 포함한 XFP_tree 결과를 보여준다. 즉, XFP_tree는 빈발 구조 집합을 나타낸 트리이다. XFP_tree를 구성할 때 각 서브트리들의 루트가 다르면 분리된 구조의 서브트리로 구성된다. 경계구조는 현재 배치에서는 min_sup를 만족하지 못하여 빈발구조에 속하지 않는 구조이나 지속적으로 입력되는 스트림 데이터에서 빈발 가능성이 있는 잠재적인 빈발구조이다. (그림 1)에서 B_2의 빈발구조를 삽입한 XFP_tree의 XFPath B-C-F는 현재 XFP_tree에 포함되어 있지 않은 새로운 구조이다. 새로운 빈발구조를 삽입할 때는 이 구조가 경계구조에 포함되어 있었던 항목인지의 여부에 따라 빈발도를 추정하여 빈발도를 삽입한다. 빈발도를 추정하는 방법은 빈발구조가 경계구조에 존재하는 경우의 추정 빈발도는 ($|B_{i-1}|$* min_sub) -1이고, 경계구조에 존재하지 않을 때의 추정 빈발도는 ($|B_{i-1}|$* min_sub)-2에 의한다. 또한 XFP_tree를 유지함에 있어 계속적으로 입력되는 배치로 인해 더 이상 빈발도를 만족하지 않는 구조가 있는지 점검하여 삭제하는 과정이 필요하다. 삭제될 구조의 최소 빈발도는 현재까지 입력된 배치에 포함된

트랜잭션의 수에 대한 지지도를 고려하여 $\sum_{i=1}^{n}$ |B_i|* min_sub를 만족하는지 여부를 판단한다. (그림 2)의 (a)는 빈발도를 만족하지 않는 에지 A-B를 삭제한 결과이다. 또한 윈도우 사이즈를 2라 가정할 때 B_3이 입력되면 윈도우 사이즈를 고려하여 가장 오래전에 입력된 B_1의 정보를 XFP_tree로부터 삭제하고 최근 데이터인 B_2와 B_3의 빈발 구조정보를 XFP_tree에 유지한다. B_1



(그림 1) B_1, B_2 입력에 대한 XFP-tree

의 삭제는 XFP_tree의 각 노드정보를 일괄적으로 이동하여 삭제하고 B_3의 빈발구조를 XFP_tree에 추가한다. (그림 2)의 (b)는 B_1의 정보를 삭제하고 B_3의 빈발구조를 포함한 트리를 보인다.



(그림 2) B_1의 삭제 후 B_3의 입력에 대한 XFP-tree

한편 XFP_tree에서 빈발도를 만족하지 못하는 구조의 삭제와 더불어 경계구조에 대한 관리도 추가적으로 이루어진다. B_1의 입력상태에서의 경계구조 집합은 {ABD, ABE, BC, BD}이었고 B_2의 입력으로 인한 추가적인 경계구조 항목은 {ACE, ACF, BA, BCG, BD, AD}이다. 스트림 데이터에서는 최근에 입력된 데이터를 중요하게 고려한다. 빈발구조 탐색은 현재 윈도우 범위에 속하는 스트림 XML 데이터들로 구성된 트랜잭션으로부터 빈발구조를 마이닝한다. 즉, 마이닝 요청시에 XFP_tree로부터 주어진 임계치 이상의 빈발구조를 추출한다.

스트림 데이터가 입력된 임의의 시간 t에 대해 빈발구조 집합을 나타내는 XFP_tree 상태를 (그림 2)의 (b)라 할 때, XFP_tree로부터 일정 임계치 이상의 빈발구조만을 추출하고자 하는 최소 지지도를 min_support=3이라 하면, 이를 만족하는 항목 및 이들의 빈발 지지도는 {A:3, B:3, C:6, D:3, F:3, AC:3, AD:3, BC:3, BF:3, CF:3, BCF:3}이다. 즉, 항목 C의 지지도는 에지 A-C에서의 1:2와 에지 B-C에서 2:1의 합을 나타낸다. 여기서 A-C의 1:2 의미는 B_2에서의 A-C빈발 지지도가 1이고 B_3에서는 2인 것을 나타낸다. 그러므로 현재 시점에서의 AC 빈발 지지도는 1과 2의 합인 3이 된다. 같은 방법으로 에지 B-C의 빈발 지지도는 3이다.

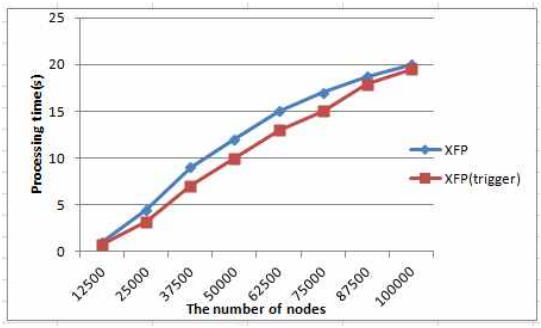
XFP_tree로부터 주어진 지지도를 만족하는 경로 및 에지를 마이닝 할 때는 FP-growth와 같은 방법으로 XFP_tree 트리의 노드들에 대한 조건부 트리를 구성하고, 이들 조건부 트리로부터 빈발한 경로 및 에지를 발견한다. 즉, 윈도우 이동에 따른 XFP_tree를 구성하고, 트리를 구성하는 노드를 중심으로 조건부 부분 트리를 생성하여 노드간의 에지 및 노드의 경로에 대한 빈발도를 유지한다.

5. 실험 및 분석

본 논문의 능동적인 슬라이딩 윈도우 지원은

빈발구조를 탐색하기 위한 전처리 단계에서 현재 윈도우 범위에 속하는 분석대상의 데이터를 트리거로 제어한다. 그리고 마이닝 과정에서 후보항목 개념의 빈발구조와 경계구조를 트리거를 이용하여 관리한다. 실험에서는 트리거 규칙 적용의 효율성에 초점을 두어 슬라이딩 윈도우와 마이닝 수행시간에 효율적으로 동작하는지에 대한 실험을 수행하였다. 실험 데이터는 http://dblp.uni-trier.de/xml 사이트와 XML 데이터 뱅크[13]의 DTD를 이용하여 IBM's AlpaWords XML generator에 의해 3000개의 XML 데이터를 생성하여 이용하였고, 오라클 9i 환경에서 실험하였다. 실험에서는 순차적으로 데이터를 입력하면서 스트림 데이터의 입력 상태를 유지하였다. XFP_tree의 실험에 필요한 조건으로 윈도우 사이즈 ω=4 그리고 각 배치에 포함되는 트랜잭션의 수는 100개로 하였다. 또한 트리거의 적용이 시스템에 미치는 효과를 실험하기 위해 트리거 규칙으로 슬라이딩 윈도우를 지원하는 XFP(trigger)와 트리거를 적용하지 않고 Java로 응용프로그램을 구현하여 실행한 XFP에 대한 비교실험을 수행하였다. 첫 번째 실험은 데이터의 입력시작으로부터 빈발구조 집합의 구조인 XFP 트리의 구성 및 유지에 소요되는 시간을 측정하였다. (그림 3)은 노드 수의 변화에 따라 트리가 구성되는 소요시간을 보여준다. 실험결과에서는 노드 수의 증가에 따라 전반적으로 XFP(trigger)가 XFP보다 상대적으로 소요시간이 적게 나타나는 것을 볼 수 있었다. 이것은 트리거를 이용한 윈도우 이동과 빈발구조 및 경계구조의 관리에 긍정적인 영향을 주고 있음을 알 수 있다. 그리고 노드 수가 증가할수록 트리의 구성 소요시간이 증가함을 알 수 있다. 즉, 입력이 많아지면서 XFP_tree의 사이즈가 커지므로 트리를 유지하는데 시간이 많이 소요되는 것으로 파악된다. 결과를 자세히 살펴보면 노드수가 80000을 초과하면서 XFP(trigger)와 XFP의 소요시간 차이가 점차로 줄어드는 것을 볼 수 있다. 이것은 노드 수가 증가된 일정 시점에서 트리거의 적용이 시스템에 부하를 주고 있음을 짐작할 수 있다. 이에 대한 상세한 시점의 분석을 위해 다음 실험을 수행하였다.

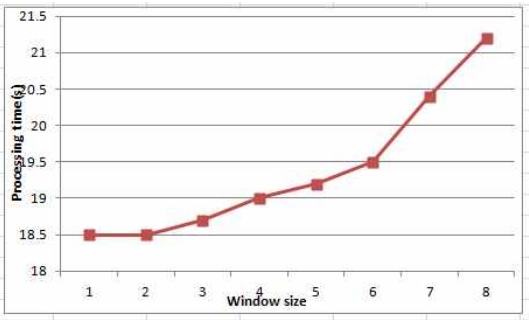
두 번째 실험에서는 트리거 규칙의 적용으로 시스템에 부하를 주는 시점을 살펴보기 위해 윈



(그림 3) 노드 수 증가에 따른 수행시간

도우 사이즈를 다르게 하면서 트리 구성 및 유지에 소요되는 시간을 측정하였다. (그림 4)는 윈도우 사이즈를 1~8로 하여 평균수행시간을 측정한 결과이다. 수행시간은 트랜잭션 데이터의 삽입 시작부터 모든 데이터의 입력이 끝나고 빈발트리 구성이 완성될 때까지의 시간에 대한 평균치를 측정하였다. 이 결과에서 알 수 있는 것은 윈도우 사이즈가 6을 초과하면서 수행시간이 급격히 증가하는 것을 볼 수 있다. 이것은 윈도우 사이즈를 크게 할수록 윈도우에 포함되는 트랜잭션 수의 증가로 트리거를 적용하는 대상 데이터의 양이 많아지기 때문에 발생하는 결과로 풀이할 수 있다. 이러한 결과는 추가적인 실험을 통해 빈발 지지도의 임계값에 변화를 주면서 트리 구성시간에 대한 실험을 하였을 때와도 유사한 결과를 나타내었다. 즉, 지지도가 커질수록 이를 만족하는 빈발구조의 노드 수가 감소하기 때문에 트리 구성 소요시간이 줄어들고, 지지도가 작을수록 고려되는 빈발구조가 많아지므로 트리 구성에 걸리는 시간이 증가하는 것을 알 수 있었다. 지지도 임계값은 마이닝 수행시간과 정확도의 결과에 많은 영향을 미치므로 데이터의 특성과 수행목적에 따라 임계값을 설정하는 것이 중요하다.

실험결과를 요약하면 트리거 규칙을 이용하여 슬라이딩 윈도우를 지원하고 빈발구조와 경계구조를 관리하는 것에 효율적이나 특정시점을 기준으로 입력되는 트랜잭션의 수가 많아지면 상대적으로 효율성이 감소하는 것을 알 수 있었다. 트리거 규칙은 데이터의 변화에 자동으로 반응하는 능동적인 기능을 갖는 장점이 있는 반면에 연관성 있는 데이터에 대한 트리거의 연쇄적인 수행



(그림 4) 윈도우 사이즈에 따른 평균 수행시간

으로 인해 시스템에 부하를 주는 경우가 발생할 수 있음을 고려해야 한다. 그러므로 트리거 동작에 대한 시스템의 영향력을 알아보는 실험은 중요하며, 시스템 부하를 최소화하여 성능을 향상시키는 방안을 연구하는 것이 추가적으로 필요하다.

6. 결론

본 논문에서는 트리거에 의한 능동적 슬라이딩 윈도우 기반의 XML 빈발구조 탐색 방법을 제안하였다. 제안된 방법은 연속적으로 입력되는 스트림 데이터에 대한 마이닝을 수행하기 위해 트리거를 이용하여 윈도우 슬라이딩을 지원하고, 마이닝 수행에서 필요한 빈발구조와 경계구조를 트리거 규칙으로 관리한다. 또한 빈발 구조 집합을 표현하는 XFP_tree을 구성하여 특정시점에서 빈발구조를 실시간으로 마이닝하는 방법을 제안한다. 이 연구는 스트림 데이터에 대한 마이닝 수행과 연속질의 처리 등을 위해 트리거를 이용하여 데이터의 흐름을 자동 제어할 수 있는 기반이 된다. 그러나 입력되는 데이터의 양이 상당히 많아지면 시스템에 부하를 가져올 수 있다. 그러므로 대용량의 스트림 데이터에 대한 트리거의 영향에 대한 상세한 분석 및 성능을 향상시킬 수 있는 방법에 대한 연구가 추가적으로 필요하다.

참 고 문 헌

[1] G. S. Manku, R. Motwani, "Approximate Frequency Counts over Data Streams", VLDB 2002.

[2] A. Deligiannakis, Y. Kotidis, and Roussopoulos“, Hierarchical In-Network Data Aggregation with Quality Guarantees", LNCS(EDBT 2004), 2004.

[3] G. Chen, X. Wu, and X. Zhu, "Mining Sequential Patterns Across Data Streams", Univ. of Vermont Computer Science Technical Report(CS-05-04), 2005.

[4] H. F. Li, S. Y. Lee, “Mining Frequent Itemsets over Data Streams using Efficient Window Sliding Techniques”, International Journal of Expert Systems with Applications, 2009.

[5] J. Chen, D. J. DeWitt, F. Tian, U. Wang, "A Scalable Continuous Query System for Internet Database", ACM SIGMOD, 2000.

[6] L.H. Yang, M.L. Lee, W. Hsu, “Finding Hot Query Patterns over An XQuery Stream", VLDB Journal Special Issue on Data Stream Processing, 2004.

[7] T. Asai, K. Abe, S. Kawasoe, H.Sakamoto, et al., "Online Algorithms for Mining Semi-Structured Data Stream", In.Proc. ICDM, 2002.

[8] J. H. Hwang, M. S. Gu, “Finding Frequent Structures in XML Stream Data”, Computational Science and Its Applications, ICCSA, 2009.

[9] M. C. Hsieh, Y. H. Wu, A. L. Chen, "Discovering Frequent Tree Patterns over Data Stream", In Proc of SIAM, 2006.

[10] C. K. S. Leung Q. I. Khan, T. Hoque, "CanTree:A Tree Structure for Efficient Incremental Mining of Frequent Pattern Sets", In proc. ICDM 2005.

[11] C. K. S. Leung Q. I. Khan, "DSTree:A Tree Structure for the Mining of Frequent Sets from Data Streams", In proc. ICDM 2006.

[12] J. Han, J. Pei, Y. Yin, "Mining Frequent Patterns without Candidate Generation", In Proc. of ACM SIGMOD International Conference on the Management of Data, 2000.

[13] NIAGARA query engine. http://www.cs.wisc.edu/niagara/data.html.

황 정 희

2001년 : 충북대학교 전자계산학과 (이학석사)
2005년 : 충북대학교 전자계산학과 (이학박사)
2001년 ~ 2006년 : 정우씨스템(주) 연구소장
2006년 ~ 현재 :남서울대학교 컴퓨터학과 전임강사
관심분야 : XML 및 웹 데이터베이스, 유비쿼터스 컴퓨팅, 데이터 마이닝