스트리밍 XML 데이터의 빈발 구조 마이닝

황 정 희[†]

요 약

유비쿼터스 환경에서 상황정보 인식 분야를 연구하면서 가장 밑바탕에서 기초가 될 수 있는 것은 인터넷 기술과 XML(Extensible Markup Language)이다. 인터넷을 통한 통신에서 XML 데이터의 사용이 일반화되고 있으며 데이터의 형태는 연속적이다. 그리고 XML 스트림 데이터에 대한 질의를 처리하기 위한 방안들이 제시되고 있다. 이 논문에서는 스트림 데이터에 대한 질의처리를 효율적으로 수행하기 위한 기반연구로써 XML을 레이블의 순서화된 트리로 모델링하여 온라인 환경에서 빈발한 구조를 추출하는 마이닝 방법을 제안한다. 즉, 지속적으로 입력되는 XML 데이터의 구조를 트리로 모델링하고 각각의 트리를 하나의 트리 집합의 구조로 표현하여 현재 윈도우 시점에서 빈발한 구조를 정확하고 빠르게 추출하는 방법을 제시한다. 제시하는 방법은 XML의 질의 처리 및 색인 구성의 기초 자료로 활용될 수 있다.

키워드 : 스트림 데이터, XML 데이터, 빈발구조추출, XML 마이닝

Mining of Frequent Structures over Streaming XML Data

Jeong Hee Hwang[†]

ABSTRACT

The basic research of context aware in ubiquitous environment is an internet technique and XML. The XML data of continuous stream type are popular in network application through the internet. And also there are researches related to query processing for streaming XML data. As a basic research to efficiently query, we propose not only a labeled ordered tree model representing the XML but also a mining method to extract frequent structures from streaming XML data. That is, XML data to continuously be input are modeled by a stream tree which is called by XFP_tree and we exactly extract the frequent structures from the XFP_tree of current window to mine recent data. The proposed method can be applied to the basis of the query processing and index method for XML stream data.

Key Words : Stream Data, XML Data, Frequent Structure Extraction, XML Mining

1. 서 론

유비쿼터스는 문자 그대로 해석하면 모든 곳에 존재한다는 의미이다. 유비쿼터스 환경에서 상황정보 인식 분야를 연구하면서 가장 밑바탕에서 기초가 될 수 있는 것이 인터넷 기술과 XML(Extensible Markup Language)이다. 현재의 인터넷 서비스는 제한적으로 그것에 접근 할 수 있는 단말기와 모델만 있다면 언제 어디서든 원하는 서비스를 얻을 수 있는 충분한 인프라를 갖추고 있기 때문이다. 또한 서로 다른 기반의 서비스 간에도 통신이 가능하다는 장점이 있다[1, 2, 3].

XML은 HTML의 개념에서 한 단계 더 나아가 미래의 웹 서비스를 주도할 중요한 도구로서 유동적으로 정보의 포맷을 만들고 그것과 함께 해당 데이터를 공유할 수 있게 한다. 예를 들면 자동차의 생산자가 자동차에 대한 정보를 어떤 정해진 형식에 맞추어서 생성하면 그것이 어떤 “형식”에 부

합하므로 어느 곳에서나 그 정보를 유효한 값으로 받아 볼 수 있고 또 다른 자동차와도 쉽게 비교가 가능해 질 수 있다. 다시 말해 XML은 마크업 심볼을 임의로 정의할 수도 있으며, 단순히 이미지나 텍스트를 보여주는 것이 아니라 XML 자체가 데이터가 되어 다른 개체에 저장되거나 연산을 수행할 수 있다. XML은 위의 예와 같이 어떤 서비스든지 그 서비스를 기술하는 방법을 제공하는 유연성이 있으므로 유비쿼터스 환경의 상황정보 인식 부분을 구축하는데 중요한 매체로 사용될 수 있다. 그리고 현재 유비쿼터스 환경을 위한 기초 데이터로써 XML의 사용은 일반화되고 있고 데이터의 입력은 연속적인 형태(stream data)이다[4, 5, 6].

스트림 데이터(stream data)는 전자상거래 사이트의 웹로그, 네트워크의 트래픽을 감시하는 모니터링 시스템, 유비쿼터스 환경 등에서 사용하는 센서 네트워크를 통해 수집 가능한 데이터로써, 매우 빠른 속도로 끊임없이 지속적으로 발생하며 그 양이 방대하다는 특성을 갖는다. 스트림 데이터는 비즈니스 데이터와는 특성이 크게 달라서 스트림 데이터의 처리를 위해 기존의 데이터 저장, 분석 기술을 그대로

※ 이 논문은 2007학년도 남서울대학교 학술 연구비 지원에 의하여 연구되었음
[†] 정 희 원 : 남서울대학교 컴퓨터학과 전임강사
 논문접수 : 2007년 5월 18일, 심사완료 : 2007년 9월 24일

사용하기는 힘들다. 최근 들어 많은 연구자들이 스트림 데이터의 처리 기술 개발에 힘써 왔고, 다양한 시스템들이 개발되고 있다.

기존의 응용 프로그램에서는 데이터 원본들이 주로 디스크나 테잎에 데이터가 들어있다는 관점하에 이들에 대한 순차 접근이나 임의 접근이 가능하다고 보았다. 하지만 최근에 이르러 많은 응용분야에 있어서 이러한 관점이 더 이상 유효하지 않게 되었다. 따라서 기존의 데이터 원본에 대한 관점이 유효하지 않은 이러한 분야에서는 질의 처리나 데이터 마이닝에 관한 새로운 많은 이슈(issue)들이 발생하게 되었는데, 특히 데이터 마이닝과 관련된 이러한 이슈들에 대한 연구를 스트림 데이터 마이닝이라 부른다. 이러한 응용분야의 예로 네트워킹 분야를 들 수 있는데 이 분야는 라우터와 같은 굉장히 많은 네트워크 구성 요소들과 관련된 프로토콜 및 데이터 교환 트래픽들로 구성되며, 통행량의 조절, 장애처리, 용량 모니터 및 조절, 침입탐지, 주문형 회계와 청구서 발송 등의 작업을 요구한다[8, 9, 10].

이러한 응용 분야에서도 데이터가 계속해서 끊임없이 도착하는데, 이러한 데이터의 특징은 데이터 도착 속도가 가변적이며 많은 소스들이 데이터를 전송하고 또한 데이터의 사이즈가 아주 거대해서 무한한 데이터 스트림(infinite data stream)을 형성하기 때문에 디스크나 테잎 같은 곳에 저장을 할 수 없다는 것이다. 이러한 스트림 데이터를 처리하기 위하여 새로운 스트림 데이터 모델이 요구된다. 특히, 웹 데이터의 표준인 XML의 사용은 일반적인 기준이 되고 있다. 그러므로 XML 스트림 데이터에 대한 질의처리 및 이를 위한 색인기법에 대한 연구가 수행되고 있다[5, 7, 8]. 그러나 빠르고 효율적인 질의 처리 및 색인구성을 위해서는 XML 데이터의 구조 분석이 반드시 필요하다. 이 논문에서는 슬라이딩 윈도우 기법을 기반으로 스트리밍 XML 데이터로부터의 빈발하게 발생하는 구조를 효율적으로 추출하는 마이닝 방법을 제안한다. 개별적인 XML 데이터 트리로부터 빈발한 구조를 추출하지 않고 현재 윈도우 시점에서의 XML 트리들을 하나의 트리로서 표현하여 빈발구조를 추출하므로 윈도우 이동시점에서 삭제되는 트랜잭션의 XML 트리의 구조에 해당하는 노드들을 별도로 찾아서 빈발도를 갱신하는 과정을 거치지 않고 단지 윈도우 이동에 따른 트리노드의 빈발도를 일괄적인 이동에 의해 갱신을 수행하므로 정확하고 빠른 마이닝 결과를 얻을 수 있다.

이 논문의 구성은 다음과 같다. 2장에서는 데이터 스트림에 대한 관련 연구를 기술하고 3장에서는 입력되는 스트리밍 XML 데이터의 표현모델을 설명한다. 그리고 4장에서는 이 논문에서 제안하는 스트리밍 XML 데이터의 마이닝 방법을 기술한다. 5장에서는 제안모델에 대한 실험 결과를 보이고 6장에서 결론을 맺는다.

2. 관련연구

최근 인터넷의 급격한 발달과 유비쿼터스 컴퓨팅 환경 그

리고 센서 네트워크와 같은 많은 정보들의 교환이 이루어지는 환경에서 무한의 연속적인 데이터의 전송 처리가 요구된다. 스트림 데이터 마이닝과 관련하여 군집화 알고리즘, 분류 알고리즘, 연관 규칙 알고리즘 등이 최근에 발표되었다[5, 6, 7, 9, 11]. [9]는 최소 지지도를 만족하는 아이템 셋을 계산하되 결과 중에는 사용자가 허락하는 범위 내의 오차가 있는 결과도 포함될 수 있도록 하였다. 즉, 이 오차를 ϵ , 최소 지지도를 s , 전체 데이터를 N 이라고 했을 때 결과 값에서 나타나는 횟수가 sN 이상이 되는 것을 모두 찾아주고, 동시에 $(s-\epsilon)N$ 이상이 되는 데이터들 중에서 일부도 찾아준다. 이 연구에서는 버퍼를 많이 쓸수록 더 빠른 수행 속도를 보였다.

[7]에서는 NiagaraCQ에 대하여 언급한다. 많은 질의들이 비슷한 구조를 공유한다는 사실에 기반하여 연속 질의를 점층적으로 그룹핑 한다. 또한 새로운 데이터가 도착했을 때 실행되는 질의(arrival-based query)나 특정 실행 간격을 지니는 질의(interval-based query) 등의 두 가지 경우 모두를 지원하는다. 그러나 XML 문서 스트림에 대한 다중 연속 질의 최적화에 초점을 두었다. 또한 이와 관련하여 [10]은 XML 스트리밍 데이터로부터 빈발 트리를 분석하기 위해 가상 트리(virtual tree)에서 오른쪽으로 노드를 확장하는 방법(rightmost expansion)을 이용하는 온라인 알고리즘인 StreamT를 제안하였다. 그러나 정확한 결과를 보장하기 어렵고 확장 가능한 노드의 경우를 모두 고려하기 때문에 많은 메모리가 필요하다. 이 연구와 유사한 방법을 적용하는 STMer 알고리즘[11]이 최근에 제안되었다. 이 연구에서도 노드의 tail-expansion 방법을 적용하여 스트림으로 입력되는 노드들을 레벨과 레이블에 따라 공통의 prefix tree를 생성하여 노드들을 확장시키고 빈발한 구조패턴을 탐색한다. 그러나 같은 범위의 노드들이 입력될 때까지 버퍼에 노드들을 저장하고 확장 가능한 노드들을 고려하는 것은 StreamT와 유사하다.

[8]은 XML에 대한 빈발 질의 패턴을 발견하기 위해 Apriori 기반의 후보 패턴을 생성한다. 그러나 루트화된 서브트리만을 추출하므로 일반적인 스트림 데이터에 대한 마이닝 기법으로 보기 어렵다. [12]는 FP-tree를 변형한 DSTree [13]와 유사한 방법으로 트리를 구성하는 방법을 제안하였다. 그러나 [12]는 점진적 마이닝을 위해 제안한 트리 구조이며 DSTree는 스트림 데이터를 마이닝 하기 위해 트리를 구성하기 때문에 대상으로 하는 데이터가 다르므로 트리의 구성 및 항목에 대한 빈발도 유지 방법도 다르다.

데이터 스트림에 대한 연구는 국내에서도 꾸준히 진행되고 있다. [15]에서는 새로운 트랜잭션이 지속적으로 추가되는 환경에서 최신의 유용한 정보를 추출하기 위한 방법을 제안하고 있다. 이를 위해 새롭게 출현한 항목에 대한 지연 추가와 이전 데이터 집합에 출현한 항목들 중에서 중요하지 않은 항목에 대한 전지작업이 병행되는 방법을 제안하였다. 그리고 스트리밍 XML에 대해서 질의에 빠르게 매치되는 것을 찾기 위해 스트리밍 XML 데이터 필터링 기법을 오토마타를 이용하는 방법들도 제안되고 있다.

비순서화된 스트림에서 입력 튜플을 효율적으로 정렬하고

저장하기 위한 방법 및 슬라이딩 윈도우의 생성 시기를 결정하기 위한 평균-기반 추정 방식을 [16]에서 제안하였다. 또한 XML 스트림에 대한 동적인 질의 계획을 제안하는 [17]에서는 관계 데이터 모델을 사용하는 시스템의 적응력있는 질의 처리 모델을 적용하는 방법을 제안하였다.

이렇게 XML 스트림 데이터를 처리하기 위한 질의 계획 및 빈발 질의 패턴을 발견하기 위한 연구가 진행되고 있는 반면에 스트리밍 XML 데이터로부터 빈발한 패턴을 탐색하기 위한 마이닝 기법은 미진한 상태이다. 그러므로 스트리밍 XML 데이터에 대한 빈발 구조의 분석 즉, 유비쿼터스 환경에서 기초가 되는 XML 스트림 데이터에 대한 마이닝 기법의 연구가 반드시 필요하다. 이 논문에서 제안하는 트리구조를 이용한 마이닝 방법은 DSTree와 유사한 방법이지만 DSTree는 구조없는 스트림 데이터를 대상으로 마이닝하기 위한 트리 구조를 나타내고 있으며, 이 논문에서 제안하는 XFP_tree는 스트리밍 XML 데이터에 대한 마이닝을 위해 제안된 트리이다.

3. XML 스트림 데이터의 트리 모델

구조적 특징을 갖는 XML 데이터는 일반적으로 트리 모델로 표현한다. 즉, XML 데이터를 트리로 표현하여 노드들에 대한 레이블 및 순서를 부여한 트리(labeled ordered tree)의 구조로 나타낸다. 트리로 구성되는 XML 데이터 스트림은 연속적으로 입력되므로 무한한 노드들의 시퀀스이다.

XML 데이터를 트리로 모델링하여 마이닝을 수행할 때 일반적으로 각 XML 데이터에 대한 트리에서 빈발한 패스(path) 정보를 발견한다. 이 논문에서도 연속적으로 입력되는 XML 스트림 데이터를 트리구조의 집합을 표현하는 하나의 트리를 구성하여 빈발한 패턴을 발견한다. 이 논문에서 제안하는 스트림 데이터의 트리를 XFP_tree(XML Frequent Pattern tree)라 하고, XFP_tree는 루트를 가진 서브트리들을 포함하는 하나의 트리집합이다. 즉, XFP_tree는 지속적으로 입력되는 시퀀스 트리들을 하나의 트리으로 표현하는 $XFP_tree = \{ST_1, ST_2, \dots, ST_n\}$ 의 의미를 포함한다. 여기서 ST_i 는 입력되는 XML 트리를 의미하며, n 은 가장 최근에 생성된 트리의 식별자를 의미한다.

스트림 데이터에 대한 트리 XFP_tree를 구성하는 방법은 다음과 같다.

첫째, 처음으로 입력되는 루트를 포함하는 XML 데이터에 대한 기초 트리 XFP_tree를 구성한다.

둘째, 새롭게 입력되는 트리의 루트와, 같은 루트를 포함하는 서브트리가 존재하면 기존의 서브트리에 결합하지 않고 개별적인 서브트리를 구성한다. 이것은 XML은 루트부터 자신의 노드까지의 패스(path)가 구조를 구분하는 중요한 의미를 갖기 때문이다. 그리고 트리를 구성할 때 형제간의 노드 사이에는 순서를 고려하지 않는다. 이것은 빈발 구조의 발견에서 형제노드의 순서는 중요하지 않기 때문이다.

(그림 1)은 연속적으로 입력되는 XML 스트림 데이터

(그림 1) XML 스트림 데이터 트리 표현

(그림 2) 스트림 데이터 집합을 표현하는 트리(XFP_tree)

ST_1, ST_2, ST_3 을 트리로 표현한 것이다.

(그림 2)는 (그림 1)에 있는 스트림 데이터를 하나의 트리(XFP_tree)로 구성한 예를 보여준다. ST_1, ST_3 은 루트노드 A가 같기 때문에 같은 루트를 갖는 서브트리로 구성되며, ST_2 는 루트 노드 B를 갖는 서브트리로 구성된다.

XML 데이터에 대한 빈발 트리를 [8]에서는 루트를 가진 서브트리(rooted tree)가 전체 트리에 대한 포함 정도으로써 트리의 빈발도를 나타낸다. 그러나 이 논문에서는 루트를 가진 서브트리의 개념이 아닌 부모노드와 자식노드간의 관계를 나타내는 에지(edge)를 트리의 빈발도를 측정하는 기본 개념으로 사용한다. 즉, XML 데이터에서 에지는 데이터를 구성하는 기초단위으로써, XML의 구조 정보를 나타내는 중요한 요소이며 루트부터 특정 노드까지의 path 정보는 에지를 확장한 XML의 구조를 나타내는 중요한 의미를 지닌다.

XML 스트림 데이터에 대해 마이닝을 수행하기 위한 기본 개념을 다음과 같이 정의한다.

정의 1. XML 스트림 데이터를 표현하는 트리는 $XFP_tree = \langle E_1, E_2, \dots, E_n \rangle$ 의 에지들로 구성되며, 스트림 데이터 집합인 XFP_tree의 에지는 부모노드와 자식노드의 순서관계 $E_i = \langle P_i, C_i \rangle$ 를 유지한다.

스트림 데이터를 표현하는 트리 XFP_tree는 루트노드가 같으면서 부모노드와 자식노드 관계가 같으면 에지는 공통이므로 하나의 에지로 표현한다. 즉, (그림 2)에서 ST_1, ST_3 의 $\langle A, B \rangle, \langle A, C \rangle$ 는 공통의 에지이므로 하나의 에지로 표현된다. 그러나 ST_1, ST_2 의 에지 $\langle B, D \rangle$ 는 루트 노드가 다르므로 해당 루트의 자식으로 각각 표현된다.

정의 2. XML 트리에서 루트 노드 n_1 부터 특정 노드 n_i 까지의 연속된 에지들의 구조정보를 XML의 SPath라고 하고, $SPath = \langle n_1, n_2, \dots, n_i \rangle$ 는 노드들의 레벨에 따른 순서를 고려한다.

부모와 자식관계의 노드를 형성하는 두 개의 노드간의 관계를 나타내는 에지를 확장한 개념으로써 에지들 간의 순서적 개념을 고려함으로써 XML의 구조에 대한 레벨정보를 고려하여 빈발 구조를 발견하는 중요한 기준이 된다.

정의 3. XML 트리를 구성하는 에지 $E_i = \langle P_i, C_i \rangle$ 가 XML 스트림 데이터에서 발생하는 빈도 $XSup(E_i)$ 는 스트림 트리 XFP_tree 에 포함되어 있는 트리의 발생 빈도의 합이다. 이에 대한 식은 다음과 같이 나타낸다.

$$XSup(E_i) = |E_i| / |XFP_tree|$$

여기서 $|XFP_tree|$ 는 XML 스트림 데이터 트리 XFP_tree 에 포함되어 있는 트리의 수를 의미하며, $|E_i|$ 는 트리에 나타나는 에지의 발생빈도를 의미한다. 에지는 각 XML 데이터를 나타내는 트리에서 유일한 노드의 관계로 구성되며, 주어진 사용자 지지도($\min_sup = 0 < \theta < 1$)를 만족하면 $XSup(E_i) \geq \min_sup$ 이므로 에지 E_i 는 빈발하다는 것을 의미한다. (그림 2)에서 에지 $\langle A, B \rangle$ 의 빈발도는 $2/3(0.66)$ 이다.

스트림 데이터를 다루기 위한 슬라이딩 윈도우는 시간을 기반으로 윈도우의 크기를 결정하는 시간 기반 슬라이딩 윈도우와 튜플을 기반으로 윈도우 크기를 결정하는 튜플 기반 슬라이딩 윈도우가 있다[14]. 시간 단위로 윈도우 크기를 정의하는 경우에는 트랜잭션의 발생 시간을 고려하여 현재 범위에 포함되는 유효 정보 여부가 결정되며, 튜플 개수 단위로 윈도우 크기가 정의되는 경우에는 트랜잭션 발생 순서를 고려하여 현재 윈도우 범위에 포함되는 유효 정보 여부가 결정된다. 구조정보를 가지는 스트리밍 XML 데이터는 일반적인 스트림 데이터에서 튜플과 같이 단순한 속성(attribute)과 값(value)의 쌍으로 이루어지지 않으며, 단순하게 시간간격을 기준으로 나뉘질 수 없다. 그러므로 XML 스트림 데이터는 문서를 트랜잭션으로 고려하는 구조정보의 기준에 의한 트랜잭션 수로 나눈다. 이 논문에서도 윈도우 크기를 바탕으로 현재 윈도우 범위에 속하는 모든 XML 구조에 대한 탐색을 수행하기 위하여 트랜잭션의 개수로서 윈도우 크기 정의하고 고정된 윈도우 크기를 사용자가 지정할 수 있다. 그리고 이 방법은 시간단위로 정의되는 경우에도 응용하여 적용할 수 있다.

4. 스트리밍 XML 데이터 마이닝

연속적으로 입력되는 스트림 데이터로부터 유용한 정보를 추출하기 위해서는 데이터를 효율적으로 저장하고 관리하는

방법이 필요하다. 기존의 Apriori 알고리즘이 트랜잭션에 속한 아이템들의 트리 조인 과정이 오래 걸리고, 많은 수의 후보 집합 생성이라는 단점을 보완한 FP-growth 방식은 후보 생성과정을 수행하지 않고, 트리의 prefix를 공유하고 있는 아이템을 찾아서 트리의 경로를 찾아간다. 그리고 같은 아이템은 포인터를 사용하여 가리키게 하는 방식으로 수행되므로 기존의 방식보다 데이터 마이닝의 성능을 향상시킨다. prefix tree 구조를 확장한 형태인 FP-tree를 이용하므로 트랜잭션 데이터베이스를 압축된 형태로 표현한다. 이 논문에서는 슬라이딩 윈도우 기반의 스트리밍 XML 데이터에서 빈발 구조를 추출하기 위하여 FP-tree를 변형한 XFP_tree 를 이용한다. XFP_tree 는 [13]의 DSTree를 기반으로 XML의 구조를 고려한 트리구조로써 빈발한 XML의 구조를 빠르게 추출하는 특징이 있다.

기존 연구에서는 트리를 구성하기 위해 트랜잭션들에 대한 항목들을 일련의 순서를 구성하고 이를 기준으로 트리 구조를 구성한다. 그러나 XML 데이터는 구조적인 순서가 중요한 요소이기 때문에 구조 정보를 보존하면서 트리를 구성해야 한다.

입력되는 XML 데이터에 의해 구성되는 트리 XFP_tree 는 XML의 구조를 구성하는 엘리먼트들을 트리의 노드로 구성하여 항목들을 나타내고 각 항목에 대한 빈발도를 유지한다. 그리고 같은 경로를 갖는 노드들은 경로를 공유하는 구조로 나타낸다. (그림 3)은 XFP_tree 의 구조 예를 보이기 위해 간단한 구조의 XML 스트림 데이터로 구성된 트리의 예이다.

(그림 4)와 (그림 5)는 순서대로 입력된 (그림 3)의 T_1, T_2, T_3, T_4 로 구성된 XFP_tree 의 예를 보여준다. XFP_tree 에서 트리 T_1 과 T_2 는 루트 A와 B를 가진 서브트리이다. 그리고 XFP_tree 를 구성할 때 각 서브트리들의 루트가 다르면 분리된 구조의 서브트리형태로 구성된다.

트리의 구성과 더불어 고려해야 할 것은 각 노드의 빈발도를 유지하는 것이다. XFP_tree 는 트리에 대한 노드를 구

(그림 3) 스트리밍 XML 데이터의 트리 예

T_1, T_2, T_3 (그림 4) XFP_tree (T_1, T_2, T_3 입력)

(그림 6) batch_1과 batch_2에 대한 XFP_tree

 T_1, T_2, T_3, T_4 (그림 5) XFP_tree (T_1, T_2, T_3, T_4 입력)

(그림 7) batch_2와 batch_3에 대한 XFP_tree

성할 때 루트부터 임의의 노드까지 같은 경로를 가진 노드들에 대한 빈발도를 유지한다. 이를 위해 입력되는 XML 트리를 하나의 트랜잭션으로 가정하고 트리를 구성하는 노드들을 트랜잭션의 항목으로 고려하여 각 항목에 대한 빈발도를 유지하며 윈도우를 이동시킬 때 삭제할 트랜잭션과 항목들의 빈발도를 관리하기 위해 각 노드 항목에 대해 루트부터 노드까지의 경로에 해당하는 패스경로를 하나의 구조항목으로 저장 및 관리한다. 예를 들어, 노드 D에 대한 경로 A-B-D의 경우에 ABD, 0, 1(구조항목, batch_{i-1} 빈발도, batch_i 빈발도)의 세 가지 값을 하나의 엔트리로 구성한다. 윈도우의 이동으로 오래된 batch를 삭제하고 삭제되는 batch에 포함되어 있는 항목의 변화는 XFP_tree 트리의 모든 노드를 순회하면서 노드항목에 대한 엔트리를 검색하여 빈발도를 갱신한다.

이 논문에서는 트랜잭션들로 이루어진 스트림이 고정된 크기로 일괄(batch)처리되어 입력된다고 가정한다. 즉, 하나의 batch에는 일정한 수의 트랜잭션으로 구성되어 있다. FP_tree의 빈발도 유지에 대한 예를 보이기 위해 각 batch에 포함되어 있는 트랜잭션을 batch₁(t₁, t₂, t₃), batch₂(t₄, t₅, t₆), batch₃(t₇, t₈, t₉)라 하고, 윈도우 사이즈를 2 batch라 가정한다. (그림 3)의 트리를 적용하면 스트림에서 처음에 입력되는 2개의 batch(window size=2) 즉, batch₁과 batch₂

에 대한 XFP_tree는 (그림 6)과 같다.

입력된 batch₁과 batch₂에 포함되어 있는 트랜잭션의 항목들에 대한 빈발도는 노드의 루트부터 해당 노드까지의 경로에 해당하는 구조항목을 주된 키로 하여 해쉬 테이블을 이용하여 유지한다. 그리고 각 batch에 포함되어 있는 각 트랜잭션에 따른 항목의 상세한 정보는 유지하지 않고 각 batch에 포함된 구조항목의 전체적인 빈도수를 유지한다. (그림 6)에서 A 노드의 빈발도를 나타내는 2:1은 batch₁에 대한 빈발도 2와 batch₂의 빈발도 1을 나타낸 것이다.

그리고 batch₃이 입력되면 윈도우 사이즈가 2이므로 입력된 순서에 의해 가장 입력된 순서가 빠른 batch₁은 제거된다. 그러므로 batch₁에 해당하는 트랜잭션 T₁, T₂, T₃에 포함되어 있는 항목들의 빈발도는 제거하고 batch₂에 대한 항목들의 빈발도가 이동되고 새롭게 입력된 batch₃에 포함되어 있는 항목들의 빈발도가 저장된다. 이것에 대한 결과를 (그림 7)에서 보여준다. 여기서 루트노드 A의 자손노드인 B노드를 루트로 하는 서브트리(그림 6)의 회색 노드로 표현된 서브트리)는 batch₂에서 빈도수가 0이었고 batch₃에서도 빈도수가 0이었기 때문에 (그림 6)의 XFP_tree로부터 완전히 제거된다.

(그림 7)에서와 같이 지속적으로 입력되는 트랜잭션들을 포함하는 batch에 대해 XFP_tree를 구성하는 과정은 다음과 같다.

Header table

Conditional tree of "F"

(그림 8) 노드 F의 조건부 트리

- 입력된 batch의 수가 윈도우 사이즈보다 작거나 같은 경우
if batch_1
batch_1에 의한 XFP_tree 초기 생성
else
XFP_tree에 XML 트리 구조를 추가하여 갱신
새로운 구조 항목은 트리에 추가 생성
- 입력된 batch의 수가 윈도우 사이즈보다 큰 경우
oldest batch를 XFP_tree로부터 삭제
XFP_tree에 XML 트리 구조를 추가하여 갱신
새로운 구조 항목은 트리에 추가 생성

스트림 데이터가 입력된 임의의 시간 t 에 대한 XFP_tree 상태를 (그림 7)이라 할 때, 최소 빈발 지지도 $\min_sup = 3$ 라 하면, 이를 만족하는 항목 및 경로는 {A:4, B:3, C:6, D:5, F:4, AC:3, AD:3, BC:3, BF:3, CF:4, BCF:3}이다. 예를 들어, (그림 7)에서 경로 ACE는 batch_2에서 빈발도 1, batch_3에서 빈발도 1를 나타내므로 batch_2와 batch_3를 포함하는 시간 t 에서의 빈발도 합은 2이다.

XFP_tree에서의 빈발도는 FP-tree와 같은 방법으로 주어진 지지도를 만족하는 트리의 노드들에 대한 부분 데이터 (projected database)를 생성하여 빈발도를 얻을 수 있다. 예를 들어, 노드 F에 대한 조건부 트리는 루트 노드 A로부터의 패스 A-C-F(1:0)에서 (A 1:0, C 1:0)과 루트 노드 B로부터의 패스 B-C-F(2:1)에서 (B 2:1, C 2:1)를 기초로 (그림 8)과 같이 구축된다(노드 F에 대한 패스 ACF의 빈발도는 1이므로 $\min_sup$ 만족하지 않아 트리구성에서 제외되지만 노드 C는 예지 CF:4(루트노드 A의 자손 예지 CF(1:0)과 루트노드 B의 자손 예지 CF(2:1)를 고려하여 조건부 트리를 구성한다).

XFP_tree는 현재 윈도우에 포함되는 트랜잭션들의 트리 구조에 대한 빈발도를 중심으로 주어진 지지도와 관계없이 구축된다. 그리고 주어진 지지도를 만족하는 경로 및 예지를 마이닝 할 때는 FP-growth와 같은 방법으로 XFP_tree 트리의 노드들에 대한 조건부 트리를 구성하며, 조건부 트리를 구성할 때는 최소 지지도를 만족하지 않는 노드는 고려하지 않는다. 그리고 이들 조건부 트리로부터 예지 중심의 빈발한 경로를 발견한다. 즉, (그림 8)로부터 노드 F에 대한 경로 및 예지의 빈발도는 BCF : 3, CF : 4임을 알 수 있다.

(그림 9) 트랜잭션 수의 변화에 따른 수행시간

(그림 10) 최소 지지도의 변화에 따른 수행시간

이와 같이 윈도우 이동에 따른 XFP_tree를 구성하고, 트리를 구성하는 노드를 중심으로 조건부 부분 트리를 생성하여 노드간의 예지 및 노드의 경로에 대한 빈발도를 유지하여 사용자의 마이닝 결과에 대한 요구에 빠른 응답을 줄 수 있다.

5. 실험결과

이 절에서는 스트리밍 XML 데이터의 마이닝 방법에 대한 효율성 및 성능을 측정하기 위하여 <http://dblp.uni-trier.de/xml> 사이트의 데이터 및 XML 데이터 뱅크[18]의 DTD를 이용하여 IBM's AlphaWords XML generator에 의해 실험 데이터를 생성하였다. 생성된 1000개의 XML 데이터를 스트림 데이터가 입력되는 형태로 순차적으로 입력하여 실험하였다. 생성된 XML 데이터의 평균적인 트리 깊이는 5.3이다. 실험에서 윈도우 사이즈 $w = 4$, 그리고 batch에 포함되는 총 트랜잭션의 수를 80개 이상으로 하여 트랜잭션의 수에 따른 수행속도를 실험하였다. (그림 9)는 현재 윈도우에 포함되어 있는 트랜잭션 수의 변화에 따른 실행 시간을, (그림 10)은 최소 지지도 $\min_sup$ 의 변화에 따른 실행 시간을 보여준다.

(그림 9)에서는 트랜잭션 수의 변화에 따라 처리 시간이 선형적으로 증가하는 것을 보여준다. 그리고 XML 트리의 지속적인 입력으로 인해 노드의 수가 증가하고 이에 대한 처리 시간도 급격한 변화 없이 점진적으로 증가하고 있음을 보인다. 이 실험으로 알 수 있는 것은 윈도우 초기 상태 즉, 트리가 입력되는 초기에는 트리를 구성하고, 노드를 구성하는 항목들에 대한 구성으로 인해 처리시간이 조금 지연되지만 트리의 초기 구성과 항목 관리를 위한 데이터 구성이 초기에 구축되면 지속적으로 입력되는 트리에 대한 항목 관리 및 빈발 구조를 발견하는 데 소요되는 시간은 점진적으로

수행되는 것을 알 수 있다. (그림 10)에서는 지지도가 커질 수록 수행시간이 급격히 줄어드는 것을 볼 수 있다.

두 번째 실험에서는 이 논문에서 제안한 XFP_tree로부터 추출되는 빈발구조의 정확성을 측정하였다. XFP_tree는 윈도우의 이동에 따른 가장 최근의 트랜잭션을 포함하는 batch의 XML 트리집합을 나타낸다. XFP_tree로부터 마이닝하여 얻은 결과의 빈발 구조와 각각의 서브 트리 즉, 입력된 개별 트리로부터 직접 추출한 빈발 구조의 내용을 비교하였다. XFP_tree로부터 추출된 (department, deptname, gradstudent, name, phone, email), (movie, title, year, Directed-by, Genres, cast, Actor) 등과 같은 경로의 구조들은 개별 트리에서 FP-tree를 적용하여 추출된 빈발 구조와 비교하였을 때 100% 같은 추출 결과를 보였다. 이것은 개별적인 트리로부터 추출되는 빈발 구조와 스트림으로 입력되는 트리의 집합을 나타내는 XFP_tree로부터 빈발구조를 발견하는 것이 일치하는 것을 알 수 있다.

한편, 이 논문에서 제안하는 XFP_tree 구성에서 고려해야 할 것은 XML 트리가 유사하지 않은 구조의 XML 데이터가 계속적으로 입력될 경우에는 XFP_tree의 사이즈가 상당히 커질 수 있다는 단점이 있다. 그러므로 이러한 점들을 고려하는 방법과 트리의 사이즈에 따른 메모리 요구량, 윈도우 사이즈에 따른 성능평가 등에 대한 연구가 지속적으로 필요하다.

6. 결 론

XML은 유비쿼터스 환경의 상황정보 인식 부분을 구축하는데 중요한 매체로 사용될 수 있기 때문에 현재 유비쿼터스 환경을 위한 기초 데이터로 사용되고 있는 것이 일반적이다. 그리고 이러한 XML 데이터의 입력은 연속적인 형태를 지닌다. 이 논문에서는 스트리밍 XML 데이터로부터 빈발구조를 추출하는 마이닝 방법을 제안한다. 이를 위해 XML 데이터를 트리로 표현하였고 개별적인 XML 트리들을 하나의 트리 집합으로 표현하는 XFP_tree를 제안하였다. 그리고 XFP_tree로부터 빈발구조의 예지 및 경로를 추출하는 마이닝 기법을 제안하였다. 스트리밍 XML 데이터의 지속적인 입력으로부터 최신의 XML 데이터에 대한 마이닝 결과를 추출하기 위해 슬라이딩 윈도우 기법을 기반으로 하였으며 XFP_tree의 각 노드는 현재 윈도우에 포함되어 있는 항목들의 빈발리스트를 유지하므로써 효율적인 마이닝 결과를 얻을 수 있었다. 그리고 실험을 통해 제안하는 방법에 대한 성능 및 마이닝 결과에 대한 정확도를 알아보았다. 향후 연구에서 기존 연구 방법과의 실험 비교를 통해 XFP_tree의 사이즈 변화에 대한 처리 속도 및 성능 효과를 분석할 것이다.

참 고 문 헌

- [1] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom, "Models and Issues in Data Stream Systems," Invited paper in Proc. of PODS, 2002
- [2] V. Ganti, J. Gehrke, R. Ramakrishnan, "DEMON: Mining and Monitoring Evolving Data," TKDE 1391), pp.50-63, 2001
- [3] R. Nayak, R. Witt, A. Tonev, "Data Mining and XML Documents," International Conference on Internet Computing, 2002.
- [4] M. Zaki, "Efficiently Mining Frequent Tree in a Forest," Proceedings of the ACM SIGKDD International Conference, 2002.
- [5] T.Asai, K.Abe, S. Kawasoe, H.Sakamoto, et al., "Online algorithms for mining semi-structured data stream," In.Proc. ICDM, 2002
- [6] S. Babu, J. Widom, "Continuous Queries over Data Stream," SIG MOD Record 30(3), pp.109-120, 2001
- [7] J. Chen, D. J. DeWitt, F. Tian, U. Wang, "A Scalable Continuous Query System for Internet Database," ACM SIGMOD, 2000
- [8] 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
- [9] G. S. Manku, R. Motwani, "Approximate Frequency Counts over Data Streams," VLDB 2002
- [10] D. Braga, A. Campi, S. Ceri, M. Klemettinen, and P. Lanzi, "A Tool for Extracting XML Association Rules from XML Documents," Proceedings of IEEE-ICTAI 2002, USA, November, 2002.
- [11] M.C. Hsieh, Y.H. Wu, A.L. Chen, "Discovering Frequent Tree Patterns over Data Stream," In Proc of SIAM International Conference on Data Mining, 2006
- [12] 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
- [13] C. K. S. Leung Q. I. Khan, "DSTree: A Tree Structure for the Mining of Frequent Sets from Data Streams," In proc. ICDM 2006
- [14] J. Li. D. Maier, "Semantics and Evaluation Techniques for Window Aggregates in Data Streams," In Proc. of ACM SIGMOD International Conference on the Management of Data, 2005
- [15] 장중혁, 이원석, "데이터 스트림에서 개방 데이터 마이닝 기반의 빈발 항목 탐색," 정보처리학회논문지D, 제10-D권 제3호, 2003.
- [16] 김현규, 김철기, 김명호, "비순서화된 스트림 처리를 위한 슬라이딩 윈도우 기법," 정보과학회, 제33권 제 6호, 2006.
- [17] 김영현, 강현철, "XML 스트림 데이터에 대한 적응력 있는 질의 처리 시스템," 정보과학회, 제33권 제 3호, 2006.
- [18] NIAGARA query engine. <http://www.cs.wisc.edu/niagara/data.html>.

황 정 희

e-mail : jhhwang@nsu.ac.kr

1991년 충북대학교 전산통계학과(이학사)

2001년 충북대학교 대학원 전자계산학과
(이학석사)

2005년 충북대학교 대학원 전자계산학과
(이학박사)

2001년~2005년 정우씨시스템(주) 연구소장

2006년~현 재 남서울대학교 전임강사

관심분야: XML 및 웹 데이터베이스, 스트림 데이터 관리,
데이터 마이닝, 유비쿼터스 컴퓨팅, 시공간
데이터베이스