빈발항목 탐색 기법을 이용한 실시간 네트워크 트래픽 모니터링 방법

이재우, 이원석
연세대학교 컴퓨터과학과
e-mail : {jwlee,leewo}@database.yonsei.ac.kr

Real-time Network Traffic Monitoring using Frequent Itemset Mining

Jae Woo Lee, Won Suk Lee
Dept. of Computer Science, Yonsei University

요 약

네트워크 인프라가 급속히 발전하면서 네트워크 상에서 발생되는 트래픽을 관리하기 위해 마이닝 기법을 적용하려는 여러 연구가 활발히 진행되고 있다. 그러나 기존의 방법들은 DBMS 를 이용하여 개개의 플로우를 저장 후 분석하는 방식을 채택함으로써 엄청난 부하와 실시간 마이닝을 어렵게 하는 문제점이 있다. 본 논문에서는 제한된 크기의 메모리를 사용하여 실시간으로 발생하는 네트워크 플로우 데이터 중 빈발한 플로우를 추출하는 방법을 제안한다. 오직 빈발하게 발생하는 플오우만을 메모리에서 모니터링 트리를 사용하여 관리함으로써 메모리를 효율적으로 사용한다. 제안된 방법은 기존의 방법들과 비교할 때 적은 시스템 부하를 주면서 초고대역폭의 트래픽을 실시간으로 모니터링 할 수 있다.

1. 서론

네트워크 인프라의 급격한 발전은 네트워크에 연결된 컴퓨터 및 인터넷 접속자수를 기하급수적으로 증가 시켰으며, 이에 따라 네트워크를 통해서 전송되는 데이터의 양도 함께 증가하고 있다. 네트워크 트래픽이 급격히 증가함에 따라 각 기업 및 공공기관에서는 적은 비용으로 안정되고 지속적인 서비스를 제공하기 위해 네트워크 트래픽과 대역폭에 대한 관리의 필요성이 대두되었다. 또한 대용량 네트워크 트래픽 데이터를 대상으로 마이닝 기법을 적용하여 의미 있는 정보를 얻고자 하는 많은 연구[1,2]가 활발히 진행되고 있다. 침입탐지를 위한 네트워크 트래픽 데이터의 마이닝을 그 예로 들수 있다. 그러나 기존의 방법들은 실시간으로 발생하는 개개의 트래픽에 대한 정보를 먼저 데이터베이스 관리 시스템(DBMS)에 저장 한 뒤 사용자가 정보를 요청한 시점에 다시 읽어 처리하는 방식을 취하고 있다. 이러한 방법은 여러 측면에서 문제점을 가지고 있다. 첫째, 개개의 플로우 정보를 DBMS 에 저장함으로써 묵시적으로 엄청난 용량의 디스크 공간을 필요로 한다. 이러한 문제점을 해결하기 위해서 몇몇의 시스템들이 주기적으로 데이터베이스에 저장 된 플로우 정보들을 집계하여 필요한 디스크 용량을 줄이려고 시도했으나 이 방법 역시 데이터베이스에 삽입은 되었지만 아직 집계되지 못한 데이터들은 분석에 포함되지 않으며, 집계된 상세정도(*granularity*) 보다 더 세세한 단계까지는 분석 할 수 없다는 단점이 있다. 둘째, 일반적으로 DBMS 를 사용하여 방대한 양의 트래픽 데이터가 삽입/삭제하여야 하므로 이로 인한 시스템 부하가 상당하여 사용자의 요청 시 실시간으로 응답하기 어렵다. 따라서 기존에 제시된 시스템들은 시스템 부하와 응답의 실시간성 측면에 있어서 매우 비효율적이다. 특히 오늘날처럼 네트워크에 연결된 수 많은 호스트에서 엄청나게 많은 양의 트래픽 데이터가 지속적으로 발생하는 상황에서는 더더욱 그러하다. 네트워크 트래픽 데이터는 데이터가 무한히 빠르고 지속적으로 발생하는 데이터 스트림 환경과 유사하며 이를 분석하기 위해서는 데이터 스트림 처리기법이 적용된 방법이 사용되어야 한다.

본 논문에서는 스위치 단에서 sFlow[3] 형태로 제공되는 네트워크 플로우 데이터를 수집하기 위하여 sflowtool[4]을 사용하였다. sFlow 는 업계표준으로써 스위치 또는 라우터를 사용하는 네트워크에서 발생하는 트래픽 데이터를 추출하기 위한 샘플링 기반의 메커니즘으로써 플로우 레벨의 데이터 정보를 제공하며, 샘플링 방식을 사용하기에 초고속 네트워크에 적합한 방식이다. sFlow 데이터그램은 네트워크 플로우와 관련된 정보를 포함한 여러 필드들로 구성되어 있으나

<표 1> 네트워크 플로우 데이터의 예

Switch IP	Protocol	src.IP	src.Port	dst.IP	dst.Port	InputPort	OutputPort	Bytes
192.168.1.10	TCP	192.168.1.55	24452	yahoo.com	WWW	129	521	46
192.168.1.10	UDP	192.168.1.66	3611	213.1.2.1	12400	128	520	52
192.168.1.10	TCP	yahoo.com	WWW	192.168.1.55	3312	129	521	64
192.168.1.10	ICMP	192.168.1.66	0	213.1.2.1	0	128	520	32

본 논문에서는 오직 8 개의 필드만을 추출하여 사용하였다. 아래 **<표 1>**에는 위와 같은 방법으로 추출된 네트워크 플로우 데이터의 예가 나와있다.

본 논문에서는 기존의 네트워크 관리 솔루션들이 채택하고 있는 DBMS 를 이용한 방법이 가지는 문제점들을 해결 할 수 있는 메모리 기반의 실시간 빈발 항목 탐색기법에 대하여 논의한다. 즉, 네트워크 트래픽 데이터 발생 시 이를 DBMS 와 같은 별도의 저장 소에 저장하지 않고 제한된 메모리 공간을 사용하여 오직 한번의 접근을 통해 실시간으로 처리하는 방법을 제안한다. 제안된 기법은 지연된 삽입(*delayed insertion*) 연산과 강제 전지작업(*force-pruning*)을 통해 가까운 미래에 빈발할 가능성이 있는 플로우들만 메모리 상에서 모니터링하며 데이터 감쇄기법을 적용하여 최근의 빈발한 항목을 찾을 수 있도록 하는 것을 특징으로 한다.

본 논문의 구성은 다음과 같다. 2 장에서는 관련 연구에 대해 알아보고, 3 장에서는 제안된 방법을 이용하여 실시간으로 빈발한 네트워크 플로우를 추출하는 방법에 대해 기술한다. 4 장에서는 실험결과 및 성능에 대해 논의하며 5 장에서 결론을 기술한다.

2. 관련연구

기존의 빈발항목 탐색 기법의 대부분은 Apriori[5] 알고리즘에 기반한 것들이다. 이 방법은 빈발한 길이 *n* 항목집합과 빈발항목을 이용하여 길이 (*n+1*)인 후보 항목집합을 생성한 뒤 데이터베이스의 트랜잭션을 탐색하여 최소지지도 이상의 지지도를 갖는 항목집합을 찾는다. 이 방법은 빈발하지 않는 항목을 포함한 항목집합은 빈발할 수 없다는 *anti-monotone* 성질을 이용하여 탐색공간을 축소시키는 방식을 취하고 있다. 그러나 빈발항목을 추출하기 위하여 주어진 데이터 셋을 여러 번 반복하여 읽어야 하므로 기존의 방법과 마찬가지로 먼저 데이터를 저장하여야만 적용 가능한 방법이다.

Lossy counting[6] 알고리즘은 주어진 최대 허용오차 내에서 단 한번의 데이터 스캔을 통해 빈발항목을 추출한다. 그러나 이 알고리즘은 새로 발생한 트랜잭션의 빈도를 계산하기 위하여 별도의 버퍼영역을 사용하여 일괄처리(batch-processing)를 한다.

estDec[7]에서는 제한된 메모리 공간을 사용하여 빈발항목을 탐색할 뿐만 아니라 항목이 가장 최근에 발생한 트랜잭션이 얼마나 최근인가에 따라 서로 다른 데이터 감쇄율을 적용하는 데이터 감쇄(decay) 메커니즘을 제안하고 있다. estDec 에서는 빈발한 항목집합을 모니터링하기 위하여 전위트리를 사용한다. 이는 특정 항목집합이 현재의 트랜잭션에 발생했을 때 해당 항목집합을 나타내는 노드 뿐만 아니라 해당 항목집합의 부분집합에 해당하는 모든 노드들까지 갱신하기 위해서이다. 전위트리 형태의 트리구조는 특정 항목집합 뿐만 아니라 해당 항목집합의 부분집합에 해당하는 모든 항목집합의 빈도를 각각 유지하므로 부분집합에 해당하는 빈도를 구할 때에는 별도의 트리 탐색 없이 결과를 줄 수 있는 반면 트리의 삽입/갱신연산에 걸리는 시간이 길어진다는 단점이 있다. 왜냐하면 길이 n 에 해당하는 항목집합을 전위트리에 갱신해야 할 경우 (2^n-1) 개의 부분집합에 해당하는 모든 노드를 갱신해야 하므로 갱신연산의 복잡도는 $O(2^n)$ 이 된다. 따라서 방대한 양의 네트워크 플로우 데이터의 빈발한 삽입/갱신이 필요한 네트워크 트래픽 데이터의 마이닝에는 적합하지 않다.

3. 제안하는 빈발항목 탐색기법

3.1 모니터링 트리

실시간으로 발생하는 네트워크 플로우 데이터 중 빈발한 플로우를 추출하기 위하여 본 논문에서는 OLAP 에서 효과적으로 큐브를 생성 및 관리하기 위해 제안된 H-tree[7] 구조에서 Header table, side link 등과 같이 큐브 생성을 위한 부차적인 데이터 구조를 제거해 단순화된 트리구조를 사용하였으며 이를 모니터링 트리라고 명명하였다. 모니터링 트리의 구조는 H-tree 와 유사하나 트리를 갱신/삽입하는 방법은 H-tree 의 것과 매우 상이하다. 본 논문에서는 빠르게 발생하는 네트워크 데이터의 신속한 처리를 위하여 즉, 스트림 환경에 적합하게 만들기 위해 새로운 트리 관리기법을 도입하였다. 이에 대한 설명은 3.2 절에 나와있다. 본 논문에서 사용하는 모니터링 트리는 아래와 같은 특징들을 갖는다.

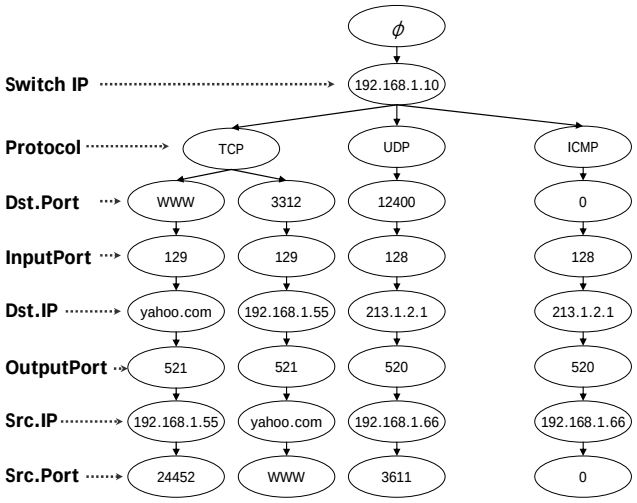
i) 모니터링 트리의 최대 깊이는 분석을 위해 사용되는 속성의 수와 동일한 8 로 고정되어 있다.
ii) 모니터링 트리의 루트에서 리프(*leaf*)노드에 이르는 하나의 고유한 경로는 하나의 플로우 데이터를 나타낸다.
iii) 모니터링 트리의 각 레벨은 플로우 정보의 각 속성과 1:1 로 대응된다.

그림 1 에는 **<표 1>**의 예제에 나와있는 4 개의 플로

우 데이터를 삽입한 모니터링 트리의 모습이 나와있다. 트리의 루트에서 리프노드에 이르는 각각의 경로는 표 1 의 각 행에 표시된 플로우 데이터와 대응된다. 모니터링 트리의 각 레벨과 속성을 맵핑 시 카디날리티(*Cardinality*)가 작은 속성을 트리의 상위 레벨에 위치시키는 것이 성능향상에 도움이 된다. 왜냐하면 카디날리티가 작은 속성이 상위 레벨이 위치 할수록 더 많은 노드들이 공유되기 때문에 전체 노드의 개수를 줄일 수 있기 때문이다.

모니터링 트리의 각 노드는 다음과 같은 정보를 포함한다.

i) item: 노드가 나타내는 항목의 속성값
ii) count: 해당 노드에 대응되는 항목집합의 발생 횟수
iii) tid: 해당 노드가 마지막으로 갱신된 트랜잭션의 식별자
iv) child[]: 자식노드를 가리키는 포인터의 배열



(그림 1) 모니터링 트리

3.2 모니터링 트리의 관리

모니터링 트리의 관리는 크게 2 단계로 이루어진다. 첫 번째 단계는 갱신단계(*Update phase*)로 모니터링 트리에서 현재 발생된 트랜잭션에 해당하는 항목집합을 갱신하는 단계이며 두 번째 단계는 현재 항목집합에 대응되는 노드가 모니터링 트리 내에 존재하지 않는 경우 새로운 자식노드를 생성하여 트리를 확장시키는 확장단계(*Expansion phase*)이다.

데이터 스트림에 새로운 트랜잭션이 발생하면 즉, 매번 새로운 네트워크 플로우 데이터가 발생하면 모니터링 트리에서 관리되는 해당 트랜잭션에 대응되는 모든 노드들은 갱신되어야 한다. 네트워크 트래픽으 집합에 대응되는 데이터 스트림 $D^t=\{T^1,T^2,...,T^t,...\}$에 새로운 트랜잭션 $T^t=\{I_1^t,I_2^t,...,I_8^t\}$ 이 발생하면 모니터링 트리의 루트에서부터 깊이우선탐색 기법(*Depth-first search*)을 사용하여 현재의 트랜잭션 T^t에 해당하는 항목들을 탐색해나간다. 탐색하고자 하는 항목이 모니터링 트리 상에 존재하지 않는 경우 탐색을 멈춘다.

오로지 빈발한 플로우 만을 모니터링 트리에서 관리하기 위해서 본 논문에서는 두 가지 임계값을 사용한다. 하나는 추가지연지지도(S_{ins})이며 다른 하나는 전지지지도(S_{prn})이다. 지연추가지지도 S_{ins} 와 전지지지도 S_{prn} 는 사용자 정의 임계값이며 최소지지도 S_{min} 보다 작은 값이어야 한다(i.e. $S_{prn}<S_{ins}<S_{min}$).

갱신단계를 마치게 되면 트리의 확장을 위해 다시 한번 트리의 루트에서부터 현재의 항목집합에 해당하는 노드들을 순차적으로 탐색해나간다. 탐색해야 할 레벨(v)에 해당하는 항목이 모니터링 트리 상에 존재하지 않는 경우 해당 항목은 다음과 같은 두 가지 조건을 만족할 경우 마지막으로 탐색된 노드 즉, (v-1)레벨 노드의 자식노드로 생성된다.

i) 해당 항목의 지지도는 지연추가지지도($S_{ins}<S_{min}$) 이상이다. (i.e. $count(I_v^t)/|D^t| \geq S_{ins}$)
ii) 해당 (v-1)레벨의 노드(N_{v-1})의 지지도가 S_{ins} 이상이다. (i.e. $count(N_{v-1})/|D^t| \geq S_{ins}$)

특정항목집합이 현재 트랜잭션에서 발생했을 때 이를 바로 모니터링 트리에 추가하지 않고 자신의 부분집합에 해당하는 항목집합들이 지연추가지지도 S_{ins} 이상이 될 때까지 지연시켰다가 삽입하는 이유는 가까운 미래에 빈발할 가능성이 있는 항목집합들만 모니터링 트리에서 관리하기 위해서이다. 그러나 이러한 방식으로 지연삽입을 한다고 하더라도 모니터링 트리에 존재하는 지지도가 어느 정도 낮은 항목집합들을 제거하지 않는다면 모니터링 트리 안에서 관리되는 항목집합의 수 는 계속 늘어나게 된다. 따라서 모니터링 트리 안에 관리되는 항목집합들 중 가까운 미래에 빈발항목이 될 가능성이 없는 항목집합들은 삭제를 해주어야 한다. 따라서 지지도가 일정수준 S_{prn} 미만인 항목들은 트리에서 제거를 한다. 이 지지도를 전지지지도라고 한다.

3.3 빈발항목 집합의 탐색

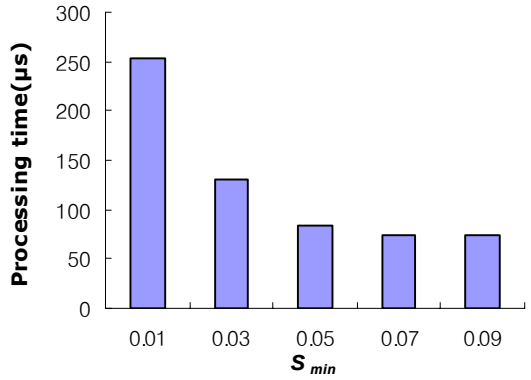
모니터링 트리 안에서 관리되는 항목집합들 중 빈발항목집합을 찾기 위해서는 트리안에 존재하는 모든 경로들을 전부 방문해야 한다. 트리에 존재하는 모든 경로들을 전부 방문하는 것은 상당한 처리시간을 요구하는 작업이나 이는 사용자의 요청이 있을 시에만 실행되는 작업이며 *anti-monotone* 성질을 이용하여 탐색공간을 줄일 수 있다. 즉, 깊이우선탐색 방법으로 트리를 탐색하는 중 해당 노드의 지지도가 $S_{\min}$ 미만일 경우 현재 노드의 자식노드들은 더 이상 방문할 필요가 없다. 이러한 방법을 통해 빈발항목집합 탐색 시 탐색공간을 줄일 수 있다.

4. 실험 및 성능평가

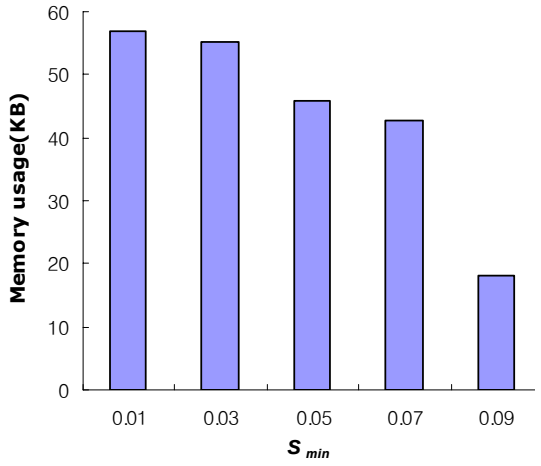
제안된 방법의 성능을 평가하기 위해서 sFlow 로그 데이터 생성이 가능한 두 대의 스위치에서 각 스위치당 6 개의 인터페이스 즉, 총 12 개의 인터페이스를 sFlow 데이터를 생성하도록 활성화 시킨 뒤 100,000 개의 네트워크 플로우 데이터를 수집하였다. 앞에서 설명한 바와 같이 각 플로우 데이터는 9 개의

속성을 가지고 있으며 이중 8 개는 트리의 각 레벨에 맵핑되고 1 개는 측정값으로 사용되었다. 모든 실험은 1GB 의 메모리를 가진 2.8GHz 팬티엄 컴퓨터와 리눅스 환경에서 실험되었으며 모든 프로그램은 C 언어로 구현되었다.

그림 2 는 최소 지지도 변화에 따른 튜플 당 처리시간의 변화를 보여주고 있다. 최소 지지도 값이 커질수록 빈발항목의 수가 급격하게 감소하게 되어 실행시간도 이에 비례하여 감소하게 된다.
그림 3 은 최소지지도 변화에 따른 메모리 사용량의 변화를 나타낸다. 최소 지지도 값이 커질수록 모니터링 트리에서 관리되는 노드 수가 감소하므로 메모리 사용량이 점점 줄어든다.



(그림 2) 최소지지도 변화에 따른 처리시간 변화



(그림 3) 최소지지도 변화에 따른 메모리 사용변화

5. 결론

대용량의 네트워크 데이터 특히, 기가비트 이상의 대역폭을 가지는 백본 네트워크의 트래픽 분석에 있어서는 얼마나 빠른 시간안에 적은 크기의 메모리를 사용하여 데이터를 처리하는 가가 큰 관건이다. 따라서 본 논문에서는 실시간으로 발생하는 네트워크 플로우 데이터 스트림에 대해서 빈발항목 집합을 탐색하는 기법을 제안하였다. 네트워크 관리자들은 실시간으로 트래픽을 모니터링 하기를 원하며 실제 네트워크 관리에 있어 중요한 데이터는 최근의 데이터임을 감안하여 처리시간과 제한된 메모리의 사용에 최적화된 방법을 제시하였다.

참고문헌

[1] Y. Hu and B. Panda, "A Data Mining Approach for Database Intrusion Detection," In Proceedings of the 2004 ACM Symposium on Applied Computing, pp. 711-716, 2004.
[2] J. Luo and S. Bridges, "Mining fuzzy association rules and fuzzy frequency episodes for intrusion detection," International Journal of Intelligent Systems, Vol. 15, No. 8, pp. 687-704, 2000.
[3] http://www.sflow.org
[4] http://www.inmon.com/technology/sflowTools.php
[5] R. Agrawal and R. Srikant, “Fast Algorithms for Mining Association Rules,” *Proceedings of the 20th International Conference on Very Large Data Bases*, pp. 487-499, 1994.
[6] G.S. Manku and R. Motwani, “Approximate Frequency Counts over Data Streams,” *Proceedings of the 28th International Conference on Very Large Data Bases*, pp. 346-357, 2002.
[7] J. H. Jang and W. S. Lee. Finding recent frequent itemsets adaptively over online data streams. pp. 487-492, In Proc. of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining
[8] J. Han, J. Pei, G. Dong, and K. Wang. Efficient computation of iceberg cubes with complex measures, pp. 1-12, In Proc. of the 2001 ACM SIGMOD Intl. Conf. on Management of data.