효율적 데이터 스트림 분석을 위한 발생빈도 예측 기법을 이용한 과부하 처리

장 중 혁†

요 약

근래 들어 유비쿼터스 컴퓨팅 및 센서 네트워크 환경 등과 같은 다양한 응용 분야에서 데이터 스트림 형태의 정보를 발생시키고 있으며, 이들 정보를 효율적으로 처리하기 위한 다양한 방법들이 활발히 제안되어 왔다. 대부분의 이들 방법들은 주로 처리 과정에서의 공간 사용량 및 데이터당 처리 시간을 줄이는데 초점을 맞추고 있다. 하지만 이들 방법들에서 데이터 발생량이 급격히 증가되는 경우 일부 데이터는 실시간으로 처리되지 못하며 해당 방법의 성능 저하를 초래한다. 따라서, 데이터 스트림 처리의 효율성을 높이기 위해서는 효율적인 과부하 처리 기법을 필요로 한다. 이를 위해서 본 논문에서는 발생빈도 예측법을 이용한 과부하 처리 기법을 제안한다. 즉, 해당 기법에서는 처리 대상 데이터의 현재 시점까지의 발생빈도를 고려하여 해당 데이터의 향후 발생 상황을 예측하며, 이를 통해서 해당 데이터 스트림에서 과부하가 발생했을 때 효율적으로 대처할 수 있도록 지원한다. 또한, 제안되는 방법에서는 데이터 스트림의 변화를 고려하여 튜플 선별을 위한 임계값을 적응적으로 조절함으로써 불필요한 과부하 처리 수행을 최소화한다.

키워드 : 데이터 스트림, 과부하 처리, 중요 튜플, 발생빈도 예측

Load Shedding via Predicting the Frequency of Tuple for Efficient Analsis over Data Streams

Chang, Joong hyuk†

ABSTRACT

In recent, data streams are generated in various application fields such as a ubiquitous computing and a sensor network, and various algorithms are actively proposed for processing data streams efficiently. They mainly focus on the restriction of their memory usage and minimization of their processing time per data element. However, in the algorithms, if data elements of a data stream are generated in a rapid rate for a time unit, some of the data elements cannot be processed in real time. Therefore, an efficient load shedding technique is required to process data streams efficiently. For this purpose, a load shedding technique over a data stream is proposed in this paper, which is based on the predicting technique of the frequency of data element considering its current frequency. In the proposed technique, considering the change of the data stream, its threshold for tuple alive is controlled adaptively. It can help to prevent unnecessary load shedding.

Key Words : Data Stream, Load Shedding, Significant Tuple, Prediction of Frequency

1. 서 론

근래 들어 각종 센싱 기기 및 스마트 센싱 네트워킹 기술 등이 활발히 제안되면서 유비쿼터스 컴퓨팅 환경에서 발생되는 정보의 양은 매우 방대하며, 이전의 단순 컴퓨팅 기기에서와 같이 한정적인 형태가 아니라 방대한 양의 정보들을 매우 빠른 속도로 지속적으로 발생시키고 있다[1]. 또한 상업정보 분석[2, 3] 및 웹 정보 혹은 통화 기록 로그 분석[4] 등의 분야에서도 다양한 정보들이 이러한 형태로 발생되고 있다. 일반적으로 이러한 형태의 데이터 집합을 데이터 스트림이라 지칭하며, 이러한 정보 발생 형태의 변화로 인해 한정된 데이터 집합을 대상으로 하는 기존의 데이터 분석 기술을 단순히 유비쿼터스 컴퓨팅 환경에서의 센서 데이터 처리[5, 6] 등에 적용하는데는 많은 한계를 가진다. 따라서, 근래들의 데이터베이스 연구 그룹의 관심은 데이터 스트림 형태의 정보를 효율적으로 처리하기 다양한 기법 들에 대한 연구로 옮겨가고 있으며[7], 데이터 스트림에 내재된 정보를 탐색하기 위한 다양한 방법들[8-10] 및 데이터 스트림에 대

※ 이 논문은 2005년 정부(교육인적자원부)의 재원으로 한국학술진흥재단의 지원을 받아 수행된 연구임. (KRF-2005-214-D00338)
† 정 회 원 : University of Illinois at Urbana-Champaign. Post-Doc. 연구원
연세대학교 소프트웨어응용연구소 전문연구원
논문접수 : 2006년 7월 19일, 심사완료 : 2006년 9월 29일

한 연속질의 처리를 위한 다양한 방법들[7, 11, 12]이 활발히 제안되고 있다.

이전의 연구들에서 제안된 데이터 스트림 처리 방법들에서는 처리 과정에서의 메모리 사용량을 줄이고 해당 데이터 스트림을 구성하는 단위 정보의 처리 시간을 최소화하는 것이 중요한 고려사항이다. 하지만, 각 데이터 스트림을 구성하는 단위 정보의 처리 시간을 최소화 하더라도 각 방법의 단위 시간당 처리 능력을 무한정 향상 시킬 수는 없다. 즉, 일정 단위 시간에 처리할 수 있는 단위 정보의 양은 한계를 가진다. 따라서, 데이터 스트림을 구성하는 단위 정보가 이들 방법들의 처리 능력보다 빠른 속도록 발생되는 경우에 이들 방법들은 일부 단위 정보를 실시간으로 처리할 수 없는 약점(drawback)을 갖는다. 분석 대상 범위가 제한되는 윈도우 접근법에서도 하나의 윈도우 범위에서 발생되는 단위 정보의 양이 급격히 증가되는 경우 이러한 약점은 여전히 존재한다.

이러한 약점을 극복하기 위한 방법으로 고려될 수 있는 가장 단순한 방법은 처리 대상 데이터 집합에서 전체 데이터 집합을 처리 대상으로 간주하는 것이 아니라 일부 데이터만을 선택하여 처리하는 샘플링 방법을 적용하는 것이다. 일반적인 샘플링 방법에서 처리 대상 데이터 집합은 통계적인 정보를 바탕으로 선택된다. 샘플링 방법에서는 실제 처리 대상 데이터 집합의 선택 범위에 따라 결과 집합의 정확도가 달라진다. 더욱이 통계적인 정보를 바탕으로 하는 샘플링 방법에서는 해당 통계정보의 정확성도 처리 결과의 정확성에 크게 영향을 미칠 수 있다. 하지만, 데이터 스트림은 시간 변화에 따라 지속적으로 변화된다[10, 13]. 따라서, 해당 데이터 스트림에 대한 통계 정보를 얻는 것이 불가능하며 샘플링 집합의 크기를 정의하는 것도 매우 어려운 작업이다. 또한, 데이터 객체의 중요성을 고려하지 않는 일반적인 샘플링 방법은 의미적으로 중요한 데이터 객체를 무시할 수 있다. 따라서, 데이터 객체가 지속적으로 발생되는 데이터 스트림에서 해당 데이터 객체들 중에서 의미적으로 중요성이 높은 데이터 객체를 실시간으로 선별하는 경우 해당 데이터 스트림에 대한 처리 능력을 향상 시킬 수 있다. 또한, 데이터 스트림의 변화를 고려하여 매개변수를 동적으로 조절함으로써 선별되는 데이터의 객체의 양을 처리 능력에 최대로 근접하도록 조절한다면 해당 데이터 스트림 처리 과정의 성능 향상을 기대할 수 있다.

본 논문에서는 튜플들로 구성되는 데이터 스트림에서 의미적으로 중요한 튜플들을 실시간으로 선별하여 해당 데이터 스트림을 위한 본처리(main-processing) 작업으로 전달하는 중요 튜플 선별 작업을 제안한다. 데이터 스트림에서 발생되는 튜플들의 중요성은 각 응용 분야의 특성에 따라 각기 다른 기준으로 측정될 수 있으며 본 논문에서는 튜플들의 발생빈도를 기준으로 중요성을 판단한다. 즉, 논문에서 제안하는 방법은 데이터 스트림에서 빈번히 발생하는 중요 튜플을 선별한다. 또한, 제안된 방법에서는 선별되는 튜플의 수가 본처리 작업의 처리 능력에 근접하도록 지원하기 위해서 매개변수를 데이터 스트림의 변화 및 본처리 작업에서의 처리 능력에 따라 조절하는 동적 조절 과정을 제시한다.

본 논문의 구성은 다음과 같다. 먼저, 제 2장에서는 본 논문에서 다뤄지는 문제 상황에 대한 정리 및 기본적인 고려사항 등을 기술한다. 제 3장에서는 튜플 데이터 스트림에서 발생빈도를 고려하여 중요 튜플을 선별하는 일련의 과정을 자세히 설명한다. 특히, 중요 튜플 선별 기준이 되는 임계값의 동적 조절 과정에 대해서 상세히 기술한다. 제 4장에서는 실험을 통해서 본 논문에서 제안된 방법의 성능을 검증하며, 끝으로 제 5장에서 논문의 결론을 맺는다.

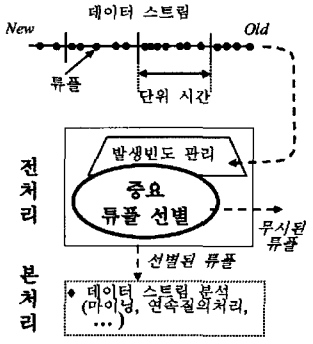
2. 기본 개념

유비쿼터스 컴퓨팅 환경 및 웹 정보 분석 등을 비롯한 다양한 응용 분야에서 발생되는 데이터 스트림에 대한 분석 작업에 있어서 전처리 과정에서 중요한 데이터 객체를 선별하여 이들을 본처리 과정로 전달하는 것은 해당 데이터 스트림에 대한 전반적인 처리 과정에 대한 성능 향상을 도모할 수 있다. 해당 데이터 스트림에 대한 분석 작업을 해당 데이터 스트림에 대한 본처리라 한다며 과부하 처리를 위한 이러한 선별 작업은 해당 데이터 스트림의 전처리 작업으로 간주할 수 있다. 이때, 일정 시간 단위에서 선별되는 데이터 객체의 수는 해당 본처리 과정의 처리 능력(즉, 일정 시간 동안 본처리 과정에서 처리될 수 있는 데이터 객체의 수)에 따라 결정된다.

본 논문에서 다뤄지는 튜플 데이터 스트림에서의 중요 튜플 선별 작업은 (그림 1)에서와 같이 해당 데이터 스트림에 대한 본처리 과정에 앞서 수행된다. 처리 대상이 되는 튜플 데이터 스트림은 지속적으로 발생되는 튜플들의 무한 집합으로서 다음과 같이 정의된다.

i. 하나의 속성 A_i의 도메인 $Dom(A_i)$은 해당 속성에 할당되는 값(atomic value)들의 집합을 의미하며, 해당 도메인에 포함되는 값들의 수를 해당 도메인의 크기라 한다.

ii. t 시점에서 발생된 하나의 튜플은 다수의 값들 $a_1, \cdots, a_n$ $(a_i \in Dom(A_i),\ 1 \le i \le n)$으로 구성되며, 이를 $u_t=(t, a_1, a_2, \cdots, a_n)$로 나타낸다.



(그림 1) 데이터 스트림에서 중요 튜플 선별

iii. 하나의 데이터 스트림 $D_k[A_1, \cdots, A_n]$는 해당 시점까지 발생된 모든 튜플들의 집합으로 정의된다. 즉, $u_t=(t, a_1, a_2, \cdots, a_n)$ $(l \le k,\ 1 \le t \le l)$일 때, $D_k[A_1, \cdots, A_n]=<u_1, u_2, \cdots, u_l>$로 정의된다. 하나의 데이터 스트림에 대한 속성 정보들을 구체적으로 명시할 필요가 없는 경우에는 해당 데이터 스트림을 간단히 D_k로 나타내며, 데이터 스트림 $D_k[A_1, \cdots, A_n]$에 포함되는 튜플의 총 수를 $|D|_k$로 나타낸다.

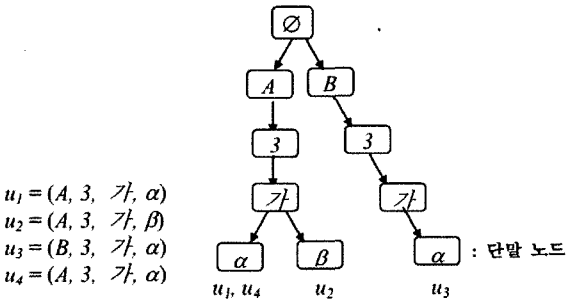
일반적으로 하나의 튜플은 키속성을 포함하는 다수의 속성으로 구성된다. 본 논문에서 다루는 문제에서는 튜플의 발생빈도를 계산하는데 있어서 튜플을 구성하는 모든 속성을 고려하는 것은 아니다. 전체 속성들 중에서 사용자에 의해 선택된 필터링 기준으로 고려돼야 할 일부 속성만을 고려하여 발생빈도를 분석한다. 각 튜플의 키속성은 다른 튜플의 해당 속성에서 동일한 값이 존재하지 않는 유일한 값을 갖는다. 따라서, 튜플들의 발생빈도를 고려하는데 있어서 키속성을 포함하는 경우에는 모든 튜플이 서로 배타적인 유일한 튜플이므로 각각의 발생빈도가 1로 계산되므로 상대적인 중요성을 구분할 수 없다. 이러한 상황을 피하기 위해서 발생빈도 분석을 위한 속성 선택에 있어서 키속성은 제외된다. 본 논문의 이후 부분에서는 튜플의 속성이라 함은 하나의 튜플을 구성하는 전체 속성들 중에서 발생빈도 분석을 위해 선택된 일부 속성들을 지칭한다.

데이터 스트림 D_k에 포함되는 하나의 튜플 u_t에 대해서 해당 튜플의 지지도(support)는 해당 데이터 스트림에 포함되는 튜플의 총수 $|D|_k$ 대비 해당 튜플과 동일한 튜플의 수의 비율을 의미한다. 지속적으로 발생되는 방대한 양의 튜플들로 구성된 데이터 스트림에 대해서 중요 튜플 선별 작업은 해당 데이터 스트림에 포함되는 튜플들 중에서 중요 튜플을 선별하여 해당 데이터 스트림의 본처리 과정으로 전달하는 작업이다. 이때, 선별되지 못한 다른 튜플들은 버려진다. 본 논문에서는 일정 임계값 이상의 지지도를 갖는 튜플을 ***중요 튜플***이라 정의하며, 이때 기준이 되는 임계값을 ***선별 지지도*** $\Lambda \in (0,1)$라 지칭한다. 더불어, 하나의 데이터 스트림에서 일정 시간동안 해당 데이터 스트림의 본처리 과정에서 처리되는 튜플의 수를 해당 ***본처리 과정의 처리 능력***이라 지칭하고 μ로써 나타낸다. 이때, 일정 크기의 시간을 단위 시간이라 지칭하며 사용자에 의해 정의되고, 예를 들어 1msec 또는 1분 등과 같이 정의될 수 있다.

한편, 발생되는 튜플의 수는 시간변화에 따라 수시로 변화될 수 있다. 따라서, 해당 데이터 스트림에 중요 튜플 선별 과정에서 선별 지지도 Λ가 일정하게 고정된다면 선별되는 튜플의 수가 본처리 과정에서 실제 처리 가능한 튜플의 수보다 작을 수 있다. 즉, 해당 데이터 스트림에서 Λ보다 낮은 지지도를 갖는 튜플들이 주로 발생되는 경우 선별되는 튜플의 수가 매우 작다. 반면, Λ보다 큰 지지도를 갖는 튜플들이 많이 발생되는 경우 선별되는 튜플의 수가 급격히 증가되어 이루 선별된 튜플들은 본처리 과정에서 처리될 수 없는 상황이 된다. 이러한 상황을 방지하기 위해서 선별 지지도 Λ는 데이터 스트림에서 튜플의 발생 정보 및 이들의 지지도를 고려하여 동적으로 조절될 수 있어야 한다. 이러한 과정이 선별 지지도의 동적 적응 과정이다.

3. 튜플 데이터 스트림에서 발생빈도 기반의 중요 튜플 선별

본 장에서는 ***FBS***(*Frequency-Based Selection*) 방법이라 불리는 데이터 스트림의 과부하 처리를 위한 중요 튜플 선별 방법을 제안한다. 제안되는 방법에서는 선별 지지도 Λ가 데이터 스트림의 변화에 따라 적응적으로 조절되며, 이를 통해서 선별되는 튜플의 수가 해당 데이터 스트림의 본처리 과정의 처리 능력에 맞도록 최적화 된다. 먼저, 3.1절에서는 데이터 스트림에서 발생되는 튜플들의 발생빈도 관리 구조에 대해서 기술한다. 3.2절에서는 선별 지지도를 적응적으로 조절하기 위한 지지도 분포 관리 구조에 대해서 설명하며, 끝으로 3.3절에서는 시간 변화에 따른 튜플들의 지지도 변화 예측 방법 과정을 포함하여 *FBS* 방법에 대해 상세히 기술한다.



(a) 데이터 집합 (b) 트리 구조

(그림 2) 출현빈도 수 관리 트리

3.1 튜플의 발생빈도 관리

튜플 데이터 스트림에서 발생되는 각 튜플들의 발생빈도를 관리하기 위해서 출현빈도 수 관리 구조를 제안한다. 출현빈도 수 관리 구조는 메모리 상에서 유지되며 서로 다른 튜플들 사이에 공통된 속성 정보들은 하나의 노드에서 표현하는 트리구조이다. 하나의 데이터 스트림 $D_k[A_1, \cdots, A_n]$ 및 이와 연관된 출현빈도 수 관리 트리에 대해서, 해당 트리의 속성 순서(attribute order) ρ는 해당 데이터 스트림을 정의하는 속성들이 해당 트리상에서 표현되는 순서를 정의한다. 즉, $\rho=A_1 \rightarrow A_2 \rightarrow \cdots \rightarrow A_n$일 때, 속성 $A_i(1 \le i \le n)$는 해당 트리의 i번째 레벨에 표현된다. 속성 순서 $\rho=A_1 \rightarrow A_2 \rightarrow \cdots \rightarrow A_n$를 갖는 데이터 스트림 $D_k[A_1, \cdots, A_n]$에서 발생되는 튜플들의 발생빈도를 관리하기 위한 출현빈도 수 관리 트리는 (그림 2)에서와 같이 해당 속성 순서에 따라 정의되는 전위 트리 구조이며 다음과 같이 정의된다.

i. 출현빈도 수 관리 트리의 레벨은 n이며, $i(1 \le i \le n)$번째 레벨은 주어진 속성 순서 ρ의 i번째 속성(즉, A_i)과 연관된다.

ii. 하나의 튜플 $u_t=(t, a_1, a_2, \cdots, a_n) \in D_k$에 대해서, 해당 튜플을 형성하는 값들을 ρ에 따라 정렬한 $a_1 \rightarrow a_2 \rightarrow \cdots \rightarrow a_n$을 ***정렬 튜플***이라 지칭한다. 하나의 튜플은 출현빈도 수 트리의 루트 노드로부터 단말 노드(leaf node)에 이르는 경로(path)들 중에서 해당 튜플의 정렬 튜플에서와 동일한 순서로 각 레벨에 값을 갖는 경로에 의해 표현된다.

iii. 내부 노드는 해당 노드와 연관된 값을 유지하기 위해서 정보목록 (*Val*)를 유지한다.

iv. 하나의 완전한 튜플을 표현하는 단말 노드는 정보목록 (*Val, tStamp, cCnt, aCnt, pCnt, pVel, pAcc*)을 유지하며 각 정보들의 의미는 <표 1>에서와 같다.

v. 서로 다른 두개의 튜플 $u_{t1}=(t1, a_1, \cdots, a_n)$, $u_{t2}=(t2, b_1, \cdots, b_n) \in D_k$에 대해서, 해당 튜플들의 정렬 튜플에서 공통되는 전위부(prefix)를 표현하기 위한 출현빈도 수 관리 트리 상의 경로(혹은 경로상의 노드)는 공유된다. 따라서 기존의 테이블 형태의 관리 구조에 비해 보다 적은 공간을 사용한다. 하지만, 개별 경로는 하나의 독립적인 튜플을 표현하므로 출현빈도 수 트리를 이용하여 데이터 스트림에서 발생하는 튜플의 출현빈도 수를 관리하는 경우에도 [5] 및 [14]에서 사용된 기존의 테이블 형태의 관리 방법에서와 마찬가지로 개별 튜플의 독립성을 보장할 수 있다.

3.2 지지도 분포 관리 구조

선별 지지도 Λ를 동적으로 조절하기 위해서 각 단위 시간에서 튜플들의 지지도 분포를 ***지지도 분포 배열***을 이용하여 관리한다. 지지도 분포 배열은 다수의 버켓(bucket)으로 구성되며, 버켓의 크기가 $r(0<r<1)$로 설정되었을 때 t번째 단위 시간에서의 지지도 분포 배열 B_t는 (m_i, N_i, a_i, β_i)로 구성되는 정보목록(entry)들의 집합으로 정의된다. 해당 지지도 분포 배열의 i번째 버켓인 $B_t[i]$는 $\{1-r\times i\}$보다 크거나 같고 $\{1-r\times(i-1)\}$보다 작은 지지도를 갖는 튜플들의 수를 관리한다. 이때, 해당 튜플들의 수는 $B_t[i]$의 네가지 정보들 중에서 N_i값에 의해 표현된다. 지지도 분포 배열의 각 버켓들에 의해 관리되는 튜플들의 지지도 범위는 상호간에 중복되지 않는다. 따라서, 지지도는 0이상 1이하의 값을 가지므로 지지도 분포 배열은 $\lceil 1/r \rceil$개의 버켓으로 구성된다. 즉, $B_t=\{(m_i, N_i, a_i, \beta_i) \mid 1 \le i \le \lceil 1/r \rceil\}$. i번째 버켓인 $B_t[i]$에서 관리되는 지지도의 하한값(lower bound)은 $B_t[i]$의 네가지 정보들 중에서 m_i에 의해 표현된다. 다시 말해서, 하나의 지지도 분포 배열 $B_t=\{(m_i, N_i, a_i, \beta_i) \mid 1 \le i \le \lceil 1/r \rceil\}$에 대해서 $m_i=1-r\times i$ $(1 \le i \le \lceil 1/r \rceil-1)$ 및 $m_{\lceil 1/r \rceil}=0$을 만족한다. 지지도 분포 배열의 한 버켓에서 관리되는 튜플들 중에서 하나의 단위 시간에서 발생되지 않거나 또는 발생빈도가 낮은 튜플이 존재한다면 해당 튜플의 지지도는 감소되며, 지지도가 해당 튜플의 속하는 버켓의 관리 범위보다 작아지는 경우에는 해당 튜플이 속하는 버켓이 변경되어야 한다. 즉, 보다 낮은 지지도를 갖는 튜플을 관리하는 버켓으로 이동되어 관리되어야 한다. 본 논문에서는 이러한 튜플을 *이동 튜플(shifting tuple)*이라 지칭한다. i번째 버켓인 $B_t[i]$에서 관리되는 N_i개의 튜플들 중에서 현재 단위 시간에서 이동 가능성이 있는 이동 튜플의 수는 $B_t[i]$의 네가지 정보들 중에서 a_i에 의해 표현된다. 이 값은 N_i보다 작거나 같은 값을 가지며 이전 단위 시간 (즉, $t-1$번째 단위 시간)에서 구해진다. 한편, 현재 단위 시간에서의 이동 가능성은 이전 단위 시간에서 구하는 예측 값으로서, 해당 값을 구하는데 있어서 현재 단위 시간에서 발생되는 튜플의 수는 이전 시점까지의 단위 시간당 평균 튜플 수와 동일한 것으로 간주한다. 더불어, $B_t[i]$의 네가지 정보들 중에서 β_i는 다음 단위 시간인 $t+1$번째 단위 시간에서의 이동 튜플 수 예측 값을 표현하며, 이 값은 현재 단위 시간이 종료되고 다음 단위 시간을 시작 되

〈표 1〉 단말 노드의 정보목록

기호	의미 / 역할
Val	해당 노드가 표현하는 값. Val∈dom(An)
TStamp	cCnt 값을 마지막으로 갱신한 튜플의 시간 정보(timestamp).
cCnt	해당 노드(즉, 하나의 튜플)의 현재 단위 시간(time slot)에서의 출현빈도 수. 단위 시간은 사용자에 의해 정의된 일정 크기의 시간을 의미. 예를 들어 1초, 1분 등과 같이 정의될 수 있음.
aCnt	해당 노드의 단위 시간당 평균 출현빈도 수. 해당 노드에 의해 표현되는 튜플의 현재 단위 시간을 제외한 범위에서의 단위 시간당 평균 출현빈도 수를 의미.
pCnt	현재 단위 시간을 제외한 가장 최근 단위 시간에서의 해당 노드의 출현빈도 수.
pVelp	해당 노드에 의해 표현되는 튜플의 현재 단위 시간을 제외한 가장 최근 단위 시간에서의 튜플 속도. ※ 하나의 단위 시간에서의 튜플의 속도 : 해당 단위 시간에서의 출현빈도 수와 직전 단위 시간에서의 출현빈도 수 차이.
pAcc	해당 노드에 의해 표현되는 튜플의 현재 단위 시간을 제외한 가장 최근 단위 시간에서의 튜플 가속도. ※ 하나의 단위 시간에서의 해당 ***튜플의 가속도*** : 해당 단위 시간에서의 해당 튜플의 튜플 속도와 직전 단위 시간에서의 해당 튜플의 튜플 속도 차이

는 시점에 a_i 값으로 부여(assign)된다.

3.3 발생빈도 기반 중요 튜플 선별 방법

튜플 데이터 스트림에서 중요 튜플 선별 방법인 *FBS* 방법은 네 단계로 구성된다. 즉, 발생빈도 갱신 단계, 튜플 선별 단계, 지지도 분포 배열 갱신 단계 및 임계값 갱신 단계로 구성된다. 속성 순서 ρ 및 연관된 출현빈도 수 관리 트리를 갖는 데이터 스트림 D_k에서 새로운 튜플 u_k=(k, a_1, a_2, …, a_n)가 t ($t \geq 2$)번째 단위 시간에서 발생되었을 때, 해당 튜플의 정렬 튜플이 *FBS* 방법에 의해 처리되며 임계값 갱신 단계를 제외한 모든 단계가 순차적으로 수행된다. 임계값 갱신 단계는 각 단위 시간이 종료됐을 때에만 수행된다. *FBS* 방법에서 필요한 매개변수 정보들은 <표 2>에서 나열한 바와 같으며, 선별 지지도 초기값 Λ_1은 사용자에 의해 설정된다.

〈표 2〉 *FBS* 방법에서 사용되는 기호들

기호	의미 / 역할
r	지지도 분포 배열의 버켓 크기
Λ_t	t번째 단위 시간에서의 선별 지지도
M_t	t번째 단위 시간까지의 단위 시간당 평균 튜플 수
μ	본처리 과정의 처리 능력
B_t	t번째 단위 시간에서의 지지도 분포 배열
$B_t[i]$	B_t의 i번째 버켓

[발생빈도 갱신 단계] 튜플 u_k에 의한 출현빈도 수 관리 갱신 작업을 수행하기 위해서 해당 튜플의 정렬 튜플에 연관된 경로가 탐색된다. 해당 정렬 튜플의 i번째 값에 대해서 출현빈도 수 트리의 연관된 노드가 존재 한다면 다음과 같은 갱신 작업이 수행된다.

i. 연관된 노드가 해당 트리의 내부 노드인 경우는 별도의 갱신 작업을 수행하지 않는다.

ii. 연관된 노드가 해당 트리의 단말 노드이면서 현재 단위 시간에서 처음으로 탐색된 경우에는 해당 노드에서 관리되는 정보목록 (*Val, tStamp, cCnt, aCnt, pCnt, pVel, pAcc*)의 *tStamp*, *cCnt* 및 *aCnt* 값이 다음과 같이 순차적으로 갱신된다.

$$aCnt = \frac{\{aCnt \times (v-1)\} + cCnt}{t-1}$$
$$\rightarrow tStamp = k$$
$$\rightarrow cCnt = 1$$

더불어, *pAcc* 값이 갱신된 마지막 단위 시간이 v번째 단위 시간일 때, *pAcc*, *pVel* 및 *pCnt* 값들도 순차적으로 다음과 같이 갱신된다.

$$pAcc = \frac{\{pAcc \times (v-1)\} + \{(cCnt - pCnt) - pVel\}}{t-1}$$
$$\rightarrow pVel = cCnt - pCnt$$
$$\rightarrow pCnt = cCnt$$

만약 *pCnt* 또는 *pVel*의 값이 0이라면(즉, *pCnt* 또는 *pVel*의 이전 값이 존재하지 않는다면), *pVel* 또는 *pAcc*값을 구할 수 없으며 이 경우에는 해당 값을 0으로 간주한다.

iv. 연관된 노드가 해당 트리의 단말 노드이면서 현재 단위 시간에서 두번 혹은 그 이상 탐색된 경우에는 해당 노드에서 관리되는 정보목록 (*Val, tStamp, cCnt, aCnt, pCnt, pVel, pAcc*)의 *tStamp* 및 *cCnt* 값이 다음과 같이 갱신된다.

$$tStamp = k, \quad cCnt = cCnt + 1$$

만약 해당 정렬 튜플의 i번째 값에 연관된 노드가 존재하지 않는다면 해당 노드가 다음과 같이 추가된다.

i. 연관된 노드가 해당 트리의 내부 노드인 경우는 *Val* 값이 다음과 같이 초기화 된다.

$$Val = \text{해당 정렬 튜플의 } i\text{번째 값}$$

ii. 새로 추가되는 노드가 단말 노드인 경우는 해당 노드에서 관리되는 정보목록 (*Val, tStamp, cCnt, aCnt, pCnt, pVel, pAcc*)의 각 값들이 다음과 같이 초기화 된다.

$$Val = \text{해당 정렬 튜플의 } i\text{번째 값},$$
$$tStamp = k,$$
$$cCnt = 1, \quad aCnt = 0$$
$$pCnt = 0, \quad pVel = 0, \quad pAcc = 0$$

[튜플 선별 단계] 출현빈도 수 관리 트리에서 정보목록 (*Val, tStamp, cCnt, aCnt, pCnt, pVel, pAcc*)를 갖는 단말 노드에서는 튜플의 선별 여부를 결정하기 위해서 해당 노드가 표현하는 튜플의 지지도를 다음과 같이 구한다.

$$\text{튜플 } u_k\text{의 갱신된 지지도} = \frac{\{aCnt \times (t-1)\} + cCnt}{|D|_k}$$

갱신된 지지도가 선별 지지도 Λ보다 크거나 같으면 해당 튜플은 중요 튜플로 간주되며, 선별되어 해당 데이터 스트림의 본처리 과정으로 전달된다. 반면에 갱신된 지지도가 Λ보다 작다면 해당 튜플은 본처리 과정으로 전달되지 않고 무시된다. 한편, 하나의 단위 시간에서 발생되는 튜플들 중에서 Λ보다 크거나 같은 지지도를 갖는 튜플의 수가 본처리 과정의 처리 능력 μ보다 많은 경우에는 시간적으로 먼저 발생되는 μ개의 튜플이 선별된 이후에 발생되는 튜플들은 비록 Λ보다 큰 지지도를 갖는 중요 튜플이라 할지라도 선별되지 않을 수 있다. 하지만 선별 지지도 값을 적응적으로 조절함으로써 이후 단계에서는 이러한 튜플들이 선별될 수 있도록 한다. 선별 지지도 값 갱신 과정은 임계값 갱신 단계에서 자세히 설명한다.

[지지도 분포 배열 갱신 단계] 이어서 $t+1$번째 단위 시간

에서의 해당 튜플의 발생빈도를 다음과 같이 예측할 수 있다. 해당 튜플의 *t+1*번째 단위 시간에서의 출현빈도 수 $cCnt'$는 현재 단위 시간에서의 *cCnt*, *pVel* 및 *pAcc* 정보를 이용하여 예측한다. 먼저 해당 튜플의 *t+1*번째 단위 시간에서의 튜플 속도 예측값 V'는 해당 튜플의 현재 단위 시간에서의 튜플 속도 및 *pAcc* 값을 이용하여 구할 수 있다. 표 1의 튜플의 속도에 대한 정의에 의해 해당 튜플의 *t*번째 단위 시간에서의 튜플 속도는 *cCnt* - *pCnt*로 구해지며, 해당 튜플의 *t+1*번째 단위 시간에서의 튜플 속도는 현재 단위 시간에서의 튜플 속도에 가속도만큼 증가될 것이므로 이를 고려하면 다음과 V'는 같이 구해진다.

$$V' = (cCnt - pCnt) + pAcc$$

다음으로 해당 튜플의 *t+1*번째 단위 시간에서의 출현빈도 수 예측값 $cCnt'$를 구한다. 표 1의 튜플 속도에 대한 정의에 의해 *t+1*번째 단위 시간에서의 튜플 속도 예측값 V'는 $V'=cCnt'-cCnt$로 표현될 수도 있다. 따라서, $cCnt'$는 *cCnt* 및 앞서 구한 V' 값을 이용하여 다음과 같이 구할 수 있다.

$$cCnt' = cCnt + V'$$

끝으로 해당 튜플의 *t+1*번째 단위 시간에서의 지지도 예측값 *PS*를 구한다. 지지도에 대한 일반적인 정의에 따라 *t+1*번째 단위 시간에서 해당 튜플의 총 출현빈도 및 발생 튜플의 총 수를 고려하여 구할 수 있다. 먼저 해당 튜플의 총 출현빈도는 *t-1*번째 단위 시간까지의 평균적인 발생빈도 *aCnt*, 현재 단위 시간 *t*에서의 발생빈도 *cCnt* 및 *t+1*번째 단위 시간에서의 발생빈도 예측값 $cCnt'$을 고려하여 $\{aCnt \times (t-1)\} + cCnt + cCnt'$와 같이 구할 수 있다. 한편, *t+1*번째 단위 시간에서의 발생 튜플의 수는 지금까지의 단위 시간당 평균 튜플 수와 동일한 것을 예측할 수 있으며, *t+1*번째 단위 시간까지 발생 튜플의 총 수는 $|D|_k + M_{t-1}$로 구할 수 있다. 여기서 M_{t-1}는 *t-1*번째 단위 시간까지의 단위 시간당 평균 튜플 수를 나타낸다. 따라서, 해당 튜플의 *t+1*번째 단위 시간에서의 지지도 예측값 *PS*는 다음과 같이 구할 수 있다.

$$PS' = \frac{\{aCnt \times (t-1)\} + cCnt + cCnt'}{|D|_k + M_{t-1}}$$

만약 하나의 튜플이 데이터 스트림의 첫번째 단위 시간에 발생했다면, 해당 튜플의 튜플 속도, 튜플 가속도 및 단위 시간당 평균 튜플 수가 정의되지 않는다. 따라서, 해당 튜플의 두번째 단위 시간에서의 출현빈도 수는 예측할 수 없다. 즉, 해당 튜플의 두번째 단위 시간에서의 지지도를 구할 수 없다. 이러한 상황을 고려하여 본 논문에서는 첫번째 단위 시간에서 발생한 튜플에 대해서 해당 튜플의 두번째 단위 시간에서의 지지도는 첫번째 단위 시간에서의 지지도와 동일한 것으로 간주한다. 이러한 가정으로 데이터 스트림의 초기 단계에는 해당 튜플의 지지도 예측값과 실제값의 차이가 클 수 있으나 데이터 스트림이 지속적으로 확장됨에 따라 예측값과 실제값의 차이는 급격히 감소된다.

해당 튜플의 발생 빈도 예측 결과, 지지도 변화로 인해서 해당 튜플을 관리하는 버켓이 변경돼야 하는 경우 지지도 분포 배열에 대한 갱신 작업이 수행된다. 현재 단위 시간의 지지도 분포 배열 B_t에서 *p*번째 버켓에 속하는 튜플이 다음 단위 시간에서 *q*번째 버켓에 포함될 것으로 예측될 때, $B_t[q].N$의 값은 1만큼 증가되는 반면 $B_t[p].N$의 값은 1만큼 감소된다. 더불어, 만약 $p<q$인 경우 (즉, *p*번째 버켓에서 관리되는 튜플의 지지도 하한값이 *q*번째 버켓에서 관리되는 튜플의 지지도 하한값보다 큰 경우), $B_t[p].a$값이 1만큼 감소된다. 하지만, 해당 값이 0이라면 더 이상 감소되지 않는다. 다음 단위 시간에서의 지지도 예측값이 변경되더라도 해당 튜플이 관리돼야 할 버켓이 변경되지 않는 경우 지지도 분포 배열에 대한 갱신 작업을 수행되지 않는다. 또한, 해당 튜플의 지지도를 관리하는 연관된 버켓이 현재 지지도 분포 배열에 존재하지 않는다면 해당 버켓이 새로 추가된다. 이 경우에는 해당 튜플이 이전에 속하던 버켓이 없으므로 새로 추가되는 버켓의 *N* 값만 1로 초기화 된다.

한편, 해당 튜플이 다음 단위 시간에서 출현하지 않는 경우의 지지도 예측값 PS'을 *aCnt*, *cCnt* 및 *t-1*번째 단위 시간까지를 기준으로 구해진 단위 시간당 평균 튜플 수 M_{t-1} 값으로부터 다음과 같이 구해진다.

$$PS' = \frac{\{aCnt \times (t-1)\} + cCnt}{|D|_k + M_{t-1}}$$

PS' 값을 이용하여 해당 튜플이 다음 단위 시간에서 전혀 발생되지 않는 경우에 속하게 되는 버켓이 변경되는지를 판단할 수 있다. 이러한 상황에서 해당 튜플이 속하는 버켓이 변경될 것으로 판단된다면 다음 단위 시간에서의 *q*번째 버켓의 이동 튜플 수(즉, $B_t[q].\beta$)가 1만큼 증가된다. 현재 단위 시간에서 발생한 모든 튜플에 대해서 지지도 분포 배열에 대한 갱신 작업이 완료된 후 *t+1*번째 단위 시간을 위한 지지도 분포 배열 B_{t+1}을 얻게 된다.

앞서 기술한 바와 같이 첫번째 단위 시간에서는 하나의 튜플을 위한 정보목록 (*Val, tStamp, cCnt, aCnt, pCnt, pVel, pAcc*) 중에서 *aCnt*, *pCnt*, *pVel* 및 *pAcc* 값이 0이며, 단위 시간당 평균 튜플 수가 정의되지 않는다. 따라서, 각 튜플에 대해서 두번째 단위 시간에서 해당 튜플이 출현하지 않는 경우의 지지도 예측값 PS'를 구하는 것이 불가능하다. 하지만, 두번째 단위 시간에서 첫번째 단위에서와 동일한 수의 튜플들이 발생하는 것으로 가정하는 경우 각 튜플의 지지도가 절반으로 감소된다. 이에 근거하여, 첫번째 단위 시간에서 발생한 각 튜플들은 두번째 단위 시간에서 이동 튜플로 간주한다. 따라서, 첫번째 단위 시간에서는 지지도 분포 배열의 각 버켓들에 있어서 β 값이 *N* 값과 동일한 것으로 간주한다.

[임계값 갱신 단계] *t*번째 단위 시간이 종료되었을 때 *t+1*

번째 단위 시간에서의 지지도 분포 배열 B_{t+1}= {(m_i, N_i, a_i, β_i) | $1 \le i \le \lceil 1/r \rceil$}이 구축된다. 해당 시점에서 B_{t+1}의 각 버켓에서 유지되는 정보들 중에서 a 값은 각 버켓에서 관리되는 튜플들 중에서 t번째 단위 시간에서 실제로는 발생하지 않은 이동 튜플의 수를 나타낸다. 해당 튜플들은 t번째 단위 시간에서 전혀 발생되지 않는 경우에 소속 버켓이 변경될 것으로 예측된 이동 튜플이며, 해당 단위 시간에서 실제 발생하지 않고 $|D|_k$의 증가로 인해 지지도가 감소되었다. 따라서, 해당 튜플이 속할 버켓이 변경되어야 한다. 결론적으로, B_{t+1}의 i번째 버켓에 대해서 해당 버켓의 N 값이 a 값만큼 감소되어야 하며, 반면에 $i+1$번째 버켓의 N 값은 i번째 버켓의 a 값 만큼 증가된다.

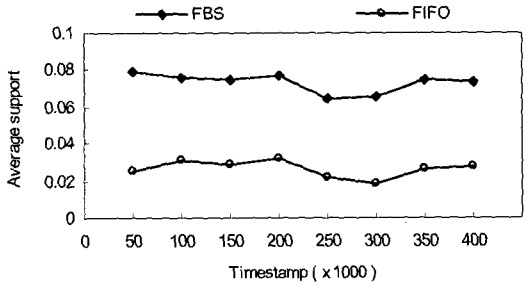
$$B_{t+1}[i].N = B_{t+1}[i].N - B_{t+1}[i].a, \quad B_{t+1}[i+1].N = B_{t+1}[i+1].N + B_{t+1}[i].a \quad (1 \le i < \lceil 1/r \rceil)$$

이어서 지지도 분포 배열 B_{t+1} 및 본처리 단계의 처리 능력 μ에 근거하여 $t+1$번째 단위 시간에서의 선별 지지도 Λ_{t+1} 값이 새로 구해진다. 과부하 처리를 위한 중요 튜플 선별 작업에 의해 선별되는 튜플의 수는 본처리 단계의 처리 능력 μ을 넘지 않아야 한다. 따라서, $\sum_{i=1}^{k} B_{t+1}[i].N \le \mu$ 및 $1 \le k \le \lceil 1/r \rceil$을 만족하는 k에 대해서 $B_{t+1}[k].m$ 값이 새로운 선별 지지도 Λ_{t+1}로 구해진다. 끝으로, 지지도 분포 배열의 각 버켓에서 β 값이 a 값으로 부여된다. 다시 말해서, $t+1$번째 단위 시간에서의 이동 튜플의 수가 B_{t+1}의 각 버켓의 a 값으로 표현되며, 해당 β는 $t+2$번째 단위 시간에서의 이동 튜플 수 예측값을 관리 하기 위해서 0으로 초기화된다. 이러한 과정을 통해서 새로운 선별 지지도 Λ_{t+1}가 구해지며, 지지도 분포 배열도 정제된다.

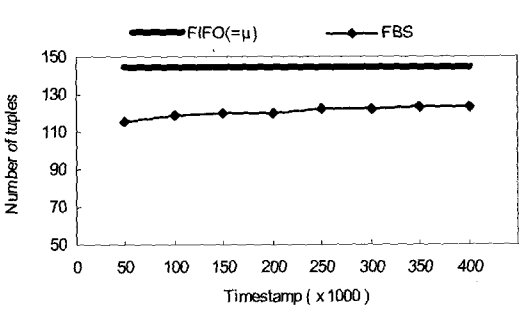
4. 성능 평가

FBS 방법의 성능을 평가하기 위해서 웹포털 사이트의 사용자 접근 로그로부터 생성된 *WebLog* 데이터 집합을 사용하였다. 하나의 웹 페이지에 대한 접속 기록을 하나의 튜플로 정의하였으며, 해당 데이터 집합은 10개의 속성을 갖는 5,056,000개의 튜플로 구성된다. 각 튜플들은 시간 순서에 따라 정렬돼 있으며 단위 시간당 튜플 생성률은 시간 변화에 따라 변한다. 한편, 10개의 속성들 중에서 출현빈도 수 관리를 위해서 4개의 속성을 활용하였으며, 이때 튜플의 지지도 평균값은 0.02917이다. 모든 실험에서 선별 지지도 초기값은 0으로 설정하였다.

먼저, *FBS* 방법에 의해 선별되는 튜플의 수 및 지지도 평균 값을 일반적인 선별 방법인 *FIFO*(first-in-first-out) 방법과 비교하였다. *FIFO* 방법은 각 단위 시간에서 먼저 발생되는 μ 개의 튜플을 선별하고 나머지 튜플은 고려하지 않는 방법이다. 본 실험에서 단위 시간의 크기는 100으로 설정되었다. 단위 시간당 튜플의 최소 발생 수는 1,216개 이며, 최대 발생 수는 1,664개이다. 이는 본 실험에 사용된 WebLog 데이터를 생성한 사이트에서는 단위 시간당 최소 1,216 번에서 최대 1,664 번의 사용자 접근이 이루어졌음을 의미한다. 한편, 해당 데이터 집합을 위한 본처리 과정의 처리 능력 μ는 145로 가정하였다. 즉, 단위 시간당 145개의 튜플을 처리할 수 있는 것으로 가정하였으며, 이는 단위 시간당 발생 튜플의 10% 정도만 처리될 수 있음을 의미한다. 본처리 과정의 처리능력은 가변적으로 설정될 수 있으나 이 경우에도 논문에서 제안된 방법의 성능에 미치는 영향이 미미하다. 지지도분포 배열의 버켓 크기는 0.001로 설정하였다. 해당 데이터 집합에서 발생되는 일련의 튜플들은 발생 시간을 기준으로 8개의 구간으로 구분하였다. (그림 3)은 두 가지 방법에 의해 선별되는 튜플의 평균 지지도를 각 구간별로 보여준다. 그림에서 보듯이 *FBS* 방법에 의해 선별되는 튜플들의 평균 지지도가 FIFO 방법에 의한 경우의 평균 지지도보다 크다. 한편, 그림에서 발생 시간 기준 250,000 및 300,000 지점에서의 평균 지지도가 다른 시점들에 비해 작은 것을 알 수 있다. 이는 해당 구간에서 높은 지지도를 갖는 튜플이 적게 발생되고 상대적으로 낮은 지지도를 갖는 튜플들이 주로 발생되었음을 의미한다. 즉, 본 실험에 사용된 데이터는 실제 웹 사이트의 웹접근 로그로부터 생성된 것으로서 두 시점에서는 해당 웹 사이트에서 기존에 자주 접근되던 웹페이지에 대한 접근은 적게 발생되고 이전까지 접근이 적었던 웹페이지에 대한 접근이 다수 발생한 것으로 판단할 수 있다. (그림 4)는 선별되는 튜플의 수를 보여준다. FIFO 방법에서는 모든 시점에서 튜플의 지지도와 무관하게 단위 시간당 처리 능력 만큼의 튜플을 선별하므로 항상 μ개의 튜



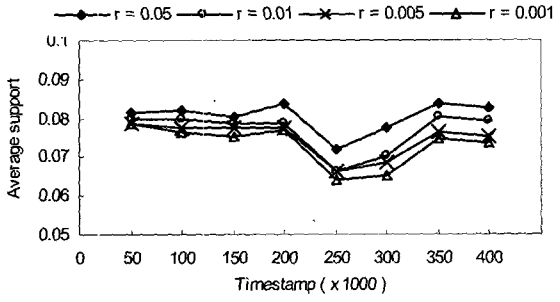
(그림 3) 선별된 튜플의 평균 지지도
(t_size=100, r=0.001)



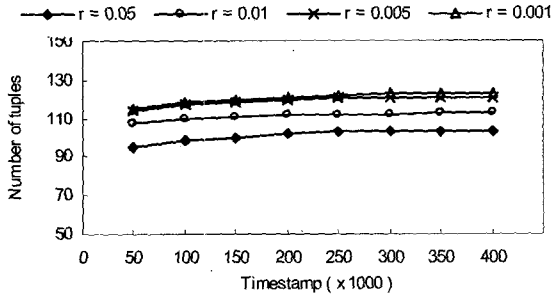
(그림 4) 선별된 튜플의 수
(t_size=100, r=0.001)

플이 선별됨을 알 수 있다. 하지만 *FBS* 방법에서는 튜플선별의 기준이 되는 선별 지지도는 이전 단위 시간까지의 튜플 발생 상황을 고려하여 설정된 값으로서 해당 단위 시간에서 지지도가 높은 튜플이 적게 발생되는 경우 실제로 선별되는 튜플의 수는 μ보다 작을 수 있다. 하지만, 튜플 발생이 증감함에 따라 선별 지지도가 보다 최적화된 값으로 설정되므로 선별되는 튜플의 수도 증가되어 μ에 가까워짐을 알 수 있다.

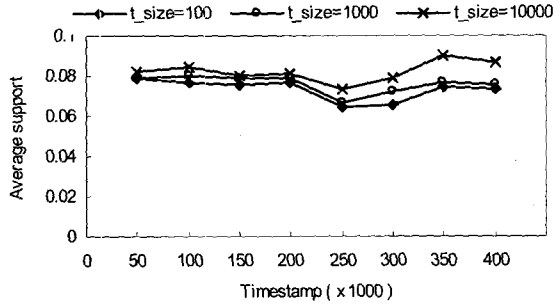
(그림 5)는 버켓의 크기 r을 다양하게 변화시켰을 때 선별되는 튜플의 평균 지지도를 보여준다. r 값이 클수록 지지도 분포 상황이 정밀하게 관리되지 못하므로 선별 지지도가 세밀하게 조절되지 못한다. 따라서, 본처리 과정의 처리 능



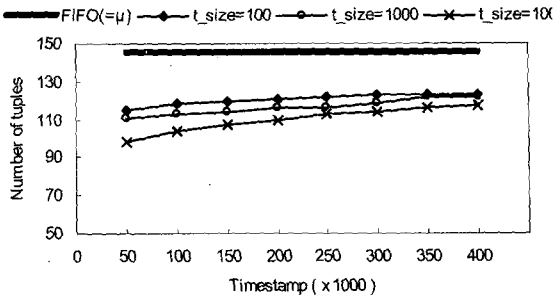
(그림 5) r 변화에 따른 선별된 튜플의 평균 지지도 변화 (t_size=100)



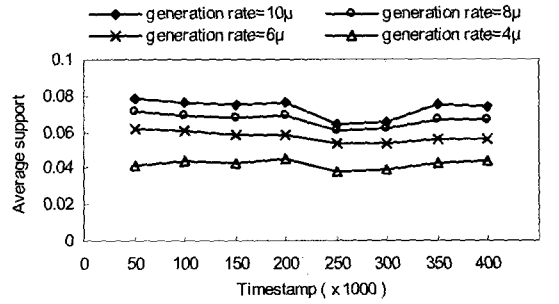
(그림 6) r 변화에 따른 선별된 튜플의 수 변화 (t_size=100)



(그림 7) 단위 시간 크기 변화에 따른 선별된 튜플의 지지도 변화 (r=0.001)



(그림 8) 단위 시간 크기 변화에 따른 선별된 튜플의 수 변화 (r=0.001)



(그림 9) 데이터 발생률 변화에 따른 선별된 튜플의 평균 지지도 (t_size=100, r=0.001)

력에 최적화될 수 있는 개수의 튜플을 선별하기 위한 임계값보다 큰 값으로 선별 지지도가 설정된다. 이로 인해서 선별되는 튜플의 평균 지지도는 높아진다. 반면, r 값이 증가될수록 선별 지지도가 상대적으로 높게 설정됨에 따라 (그림 6)에서 볼 수 있듯이 선별되는 튜플의 수는 감소된다.

(그림 7) 및 (그림 8)은 단위 시간 크기 t_size 변화에 따른 선별된 튜플의 평균 지지도 변화 및 선별된 튜플의 수를 보여준다. 각 단위 시간 크기에 대해서 본처리 과정의 처리 능력도 비례해서 조절하였다. 즉, 단위 시간 크기가 10배로 증가되는 경우 본처리 과정의 처리 능력도 10배로 증가시켰다. 선별 지지도 조절은 각 단위 시간이 종료될 때마다 수행한다. 따라서, 단위 시간 크기가 작을수록 선별 지지도 조절 작업이 보다 빈번히 수행된다. 선별 지지도 조절 작업이 보다 빈번히 수행될수록 선별 지지도가 본처리 단계의 처리 능력에 맞게 보다 낮은 값으로 설정된다. 따라서, 단위 시간 크기가 작을수록 선별된 튜플의 평균 지지도는 낮으나 선별된 튜플의 수는 증가된다.

(그림 9)는 데이터 발생률 변화에 따른 선별된 튜플의 평균 지지도를 보여준다. 본 실험에서 데이터 발생률이 10μ라는 것은 단위 시간당 발생되는 튜플의 수가 본처리 단계의 처리 능력 μ의 10배라는 것을 의미하며, 본 실험에서는 10μ, 8μ, 6μ 및 4μ 등과 같은 4개의 데이터 발생률이 실험되었다. 데이터 발생률이 낮아짐에 따라 선별된 튜플의 평균 지지도는 감소된다. 동일한 개수의 튜플을 선별하는데 있어서 데이터 발생률이 낮은 경우 보다 낮은 지지도를 갖는 튜플이 선별될 가능성이 크기 때문이다.

5. 결 론

유비쿼터스 컴퓨팅 환경을 비롯한 다양한 응용 분야에서 데이터 스트림 형태로 정보를 발생시키고 있다. 따라서, 최근 들어 데이터베이스 연구 그룹들의 주요 관심 대상이 이전과 같은 한정적인 데이터 집합에서 매우 빠른 속도록 발생되는 무한정의 데이터 집합인 데이터 스트림으로 옮겨왔으며, 다양한 분석 및 처리 방법들이 제안되고 있다. 그러나, 시간 변화에 따른 변화의 가능성이 큰 데이터 스트림의 특성을 고려할 때 데이터 스트림에서는 데이터 발생률의 변화도 매우 크다. 따라서, 시스템의 처리 능력보다 많은 부하가 발생될 수 있다. 데이터 스트림에 대한 분석 작업의 효율성을 높이기 위해서는 분석 대상 데이터 스트림에서 과부하가 발생한 경우 방대한 정보들 중에서 보다 중요한 정보를 선별하는 과부하 처리 과정이 중요하게 고려되어야 한다. 본 논문에서는 튜플 데이터 스트림에 대한 전처리 과정으로서 발생빈도를 기반으로 중요 튜플을 선별하여 효율적으로 과부하를 처리하는 적응형 선별 방법을 제안하였다. 논문에서 제안된 방법은 데이터 스트림 처리를 위한 본처리 과정의 종류에 무관하게 적용될 수 있으며, 특히 데이터 스트림에 대한 마이닝 작업 등과 같은 분석 작업에 매우 효과적으로 적용될 수 있다. 특히, 제안된 방법에서는 중요 튜플 선별의 기준이 되는 임계값을 데이터 집합에서 발생되는 튜플들의 발생빈도를 고려하여 동적으로 조절한다. 이를 위해서 하나의 튜플에 있어서 해당 튜플의 과거 발생빈도로부터 향후 발생빈도를 예측하는 출현빈도 수 예측 방법을 적용한다. 선별 지지도의 동적 조절 과정을 통해 선별되는 튜플의 수를 본처리 과정의 처리 능력에 최대한 근접하도록 지원할 수 있다.

참 고 문 헌

[1] J.M. Hellerstein, W. Hong and S.R. Madden, The Sensor Spectrum: Technology, Trends, and Requirements. *ACM SIGMOD Record*, Vol.32, No.4, pp.22-27, 2003.

[2] J. Chen, D. DeWitt, F. Tian, and Y. Wang. NiagaraCQ: A Scalable Continuous Query System for Internet Databases. In *Proceedings of the ACM International Conference on Management of Data*, pp.379-390, 2000.

[3] Y. Zhu, D. Shasha. StatStream: Statistical Monitoring of Thousands of Data Streams in Real Time. In *Proceedings of the 28th International Conference on Very Large Data Bases*, pp.358-369, 2002.

[4] C. Cortes, K. Fisher, D. Pregibon, A. Rogers, and F. Smith. Hancock: A Language for Extracting Signatures from Data Streams. In *Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp.9-17, 2000.

[5] D.J. Abadi, D. Carney, U. Cetinternel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S.B. Zdonik. Aurora: A New Model and Architecture for Data Stream Management. VLDB Journal, Vol.12, No.2, pp.120-139, 2003.

[6] R. Motwani, J. Widom, A. Arasu, B. Babcock, S. Babu, M. Datar, G. Manku, C. Olston, J. Rosenstein, and R. Varma. Query Processing, Approximation, and Resource Management in a Data Stream Management System. In *Proceedings of the 1st Biennial Conference on Innovative Data Systems Research*, pp.245-256, 2003.

[7] J. Kang, J.F. Naughton, and S. D. Viglas. Evaluating Window Joins over Unbounded Streams. In *Proceedings of the 19th International Conference on Data Engineering*, pp.341-352, 2003.

[8] M. Garofalakis, J. Gehrke, and R. Rastogi. Querying and Mining Data Streams: You Only Get One Look. In *the tutorial notes of the 28th International Conference on Very Large Data Bases*, 2002.

[9] M. Datar, A. Gionis, P. Indyk, and R. Motawi, Maintaining Stream Statistics over Sliding Windows, In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp.635-644, 2002.

[10] D. Lambert and J.C. Pinheiro, Mining a Stream of Transactions for Customer Patterns, In *Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp.305-310, 2001.

[11] R. Avnur and J. M. Hellerstein. Eddies: Continuously Adaptive Query Processing. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp.261-272, 2000.

[12] A. Das, J. Gehrke, and M. Riedewald. Approximate Join Processing over Data Streams. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp.40-51, 2003.

[13] B. Babcock, M. Datar, and R. Motwani. Load Shedding for Aggregation Queries over Data Streams. In *Proceedings of the 19th International Conference on Data Engineering*, pp.350-361, 2004.

[14] A. Arasu, S. Babu, and J. Widom. An Abstract Semantics and Concrete Language for Continuous Queries over Streams and Relations. *Stanford University Technical Report 2002-57*, 2002.

장 중 혁

e-mail : jhchang@uiuc.edu

1996년 연세대학교 컴퓨터과학과(학사)

1998년 연세대학교 대학원 컴퓨터과학과 (석사)

2005년 연세대학교 대학원 컴퓨터과학과 박사과정(박사)

2005년~현재 연세대학교 소프트웨어응용연구소 전문연구원

2006년 1월~현재 University of Illinois at Urbana-Champaign. Post-Doc. 연구원

관심분야 : 데이터 스트림, 데이터 마이닝, 정보보안, 생물정보학