다중 윈도우 조인을 위한 튜플의 도착 순서에 기반한 효과적인 부하 감소 기법

(Effective Load Shedding for Multi-Way windowed Joins Based on the Arrival Order of Tuples on Data Streams)

권 태 형 † 이 기 용 †† 손 진 현 ††† 김 명 호 ††††
(Tae-Hyung Kwon) (Ki Yong Lee) (Jin Hyun Son) (Myoung Ho Kim)

요 약 최근 다중 데이터 스트림에 대한 연속 질의 처리에 관한 연구가 활발하게 진행되고 있다. 데이터 스트림에서 튜플들의 도착 속도가 폭증하여 시스템의 메모리 용량을 초과하는 경우, 일부 튜플을 버림으로써 시스템이 과부하 상태가 되지 않도록 하는 기법을 부하 감소(load shedding)라 한다. 본 논문에서는 다중 데이터 스트림에 대한 다중 윈도우 조인을 위한 효과적인 부하 감소 기법을 제안한다. 기존의 부하 감소 기법들은 버릴 튜플을 선택하기 위해 튜플들의 조인 키 값을 이용하여 각 튜플이 생성할 조인 결과 개수(생산성)를 예측하고, 생산성이 최소가 되는 튜플을 버린다. 그러나 이러한 방법들은 조인 키 값이 다시 나타나지 않거나, 조인 키 값의 분포가 일정하게 유지되지 않는 경우 튜플들의 생산성을 올바르게 예측하기 어렵다. 본 논문은 이러한 경우를 위해 튜플들의 조인 키 값 대신, 튜플의 데이터 스트림에 대한 도착 순서를 사용하여 튜플들의 생산성을 예측하는 방법을 사용한다. 제안하는 방법은 조인 키 값으로 튜플들의 생산성을 예측하기 어려운 상황에서 튜플의 도착 순서를 통해 각 튜플의 생산성을 효과적으로 예측할 수 있도록 해준다. 다양한 실험과 분석을 통해 제안하는 새로운 부하 감소 기법이 기존 기법에 비해 더욱 효과적이고 효율적으로 부하를 감소시킬 수 있음을 보인다.

키워드 : 부하감소, 다중 스트림 윈도우 조인, 해시 조인

Abstract Recently, there has been a growing interest in the processing of continuous queries over multiple data streams. When the arrival rates of tuples exceed the memory capacity of the system, a load shedding technique is used to avoid the system becoming overloaded by dropping some subset of input tuples. In this paper, we propose an effective load shedding algorithm for multi-way windowed joins over multiple data streams. Most previous load shedding algorithms estimate the productivity of each tuple, i.e., the number of join output tuples produced by the tuple, based on its "join attribute value" and drop tuples with the lowest productivity. However, the productivity of a tuple cannot be accurately estimated from its join attribute value when the join attribute values are unique and do not repeat, or the distribution of the join attribute values changes over time. For these cases, we estimate the productivity of a tuple based on its "arrival order" on data streams, rather than its join attribute value. The proposed method can effectively estimate the productivity of a tuple even when the productivity of a tuple cannot be accurately estimated from its join attribute value. Through extensive experiments and analysis, we show that our proposed method outperforms the previous methods in terms of effectiveness and efficiency.

Key words : Load Shedding, Mulit-way Stream Window Join, Hash Join

· 이 논문은 2009년도 정부(교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(No. 2009-0083055)

† 학생회원 : 한국과학기술원 전산학과
thkwon@dbserver.kaist.ac.kr
†† 정 회 원 : 한국과학기술원 전산학과 연구교수
kiyong.lee@gmail.com
††† 종신회원 : 한양대학교 전자컴퓨터공학부 교수
jhson@hanyang.ac.kr
†††† 종신회원 : 한국과학기술원 전산학과 교수
mhkim@dbserver.kaist.ac.kr

논문접수 : 2009년 7월 6일
심사완료 : 2009년 10월 19일

정보과학회논문지: 데이타베이스 제37권 제1호(2010.2)

1. 서 론

최근, 다중 데이터 스트림에 대한 지속적인 질의 처리에 대한 관심이 높아지고 있다. 이들 다중 데이터 스트림 응용들에서, 튜플들은 각 데이터 스트림으로부터 지속적으로, 혹은 종종 폭발적으로 시스템에 도착하며, 시스템은 지속적으로 질의 결과를 산출해야 한다. 이러한 응용의 예는 네트워크 트래픽 모니터링[1,2], 센서 네트워크를 사용하는 환경 모니터링[3,4], 그리고, 이동 물체 궤적탐지[5] 등 다양하다. 이들 응용의 두 가지 난제 중 첫째는 생산되는 데이터 스트림의 분포나 크기를 실시간으로 예측하는 것이 어렵다는 것이고, 둘째는 대부분의 응용들이 실시간 혹은 거의 실시간의 응답시간을 요구한다는 것이다[1,6-9]. 따라서, 이러한 유형을 처리하는 시스템에서는 입력되는 튜플의 양이 급격하게 상승할 경우에 대비하는 것이 특히 중요한 문제이다. 입력 데이터가 폭주하여 시스템의 메모리 용량을 초과하여 입력 튜플 전체를 저장할 수 없을 때, 우리는 정확한 질의 결과를 생산할 수 없다. 이러한 상황을 처리하기 위해 다양한 부하감소(load shedding) 기술들이 제시되었다[7-9]. 부하감소는 결과의 정확성보다도 실시간 응답이 중요한 응용들에서 사용되는 기법으로 메모리 부하 감소를 위해 입력되는 튜플의 일부를 제거한다. 다시 말해, 이 문제의 핵심은 실시간 응답을 보장하면서도, 정확한 결과 값에 가장 근사하게 조인결과를 생산할 수 있는가에 달려 있다.

다중 윈도우 조인 질의는 데이터 스트림 응용들에서 널리 이용되는 것으로 모든 데이터 스트림에서 공통적으로 나타나는 객체들이나 사건들을 식별하기 위해 사용된다. 데이터 스트림은 본질적으로 무제한의 범위를 가지고 있기 때문에 시간의 제한을 가지는 윈도우(혹은 튜플의 개수)로 한정시켜야 질의를 수행할 수 있다(예를 들어 최근 1시간 동안 도착한 튜플에 대해 질의를 수행하라). 다음의 다중 윈도우 조인 질의들은 이들의 전형적인 예들로 볼 수 있다.

예제 1. [여러 개의 웹 사이트에서 언급되는 공통 뉴스 기사를 찾아라] 모니터링 뉴스 사이트를 한국일보, 경향신문, 데일리뉴스, 및 대전일보라고 하자. 각 뉴스 기사가 네 개의 속성(저자, 타이틀, 키워드, 일시)을 가진다고 가정할 때, 지난 1시간 동안 모든 뉴스 사이트에서 공통적으로 다루어지는 뉴스 기사를 찾기 위해, 우리는 '키워드'를 가지고 조인 질의를 수행할 수 있다.

예제 2. [네트워크 트래픽의 사용량을 모니터 하라] 네 개의 라우터(R_1, R_2, R_3, R_4)를 연결하는 한 개의 네트워크 패스가 있다고 하자. 각 라우터는 자신을 지나는 네트워크 패킷의 로그를 실시간으로 모니터링 시스템에 보낸다. 이들 로그는 패킷아이디로 식별될 수 있다. 우리는 지난 30분 동안 이 패스의 네트워크 사용량을 측정하기 '패킷아이디'를 이용하여 조인 질의를 수행할 수 있다.

본 논문에서 우리는 다중 윈도우 조인 질의를 위한 효과적인 부하감소 알고리즘을 제안한다. 스트림의 윈도우에 할당된 메모리가 가득 찼을 때, 이 스트림에 새로운 튜플이 도착된다면(윈도우 시간 경과로 인한 자연 도태 튜플이 없는 경우), 우리는 신규 튜플을 저장하기 위해 새로운 메모리가 필요하다. 하지만, 해당 윈도우가 더 이상 메모리를 할당받을 수 없다면, 더 이상 신규 튜플을 저장할 수 없다. 이 상황에서 우리는 윈도우에 있는 것들 중 어느 하나를 지워야 한다. 그러한 결정을 하기 위해, 부하감소 알고리즘들은 각 튜플의 조인 생산성(예를 들어 해당 튜플을 조인했을 때 산출되는 결과물의 개수) 예측을 통해 가장 생산성이 떨어질 것으로 판단되는 튜플을 제거함으로써 여분의 메모리 공간을 확보한다. 각 튜플의 생산성을 예측하기 위해, 기존 알고리즘들은 대표적으로 다음 두 가지 범주의 기준을 사용한다. (1) 출현빈도 기준 [9]: 튜플 t의 생산성은 자신의 윈도우를 제외한 나머지 윈도우들에서 튜플 t와 같은 조인키 값을 가지는 튜플들의 곱으로 표현된다. (2) 결과 기준 [7]: 튜플 t의 생산성은 t와 같은 조인키 값을 가진 튜플이 과거 출력했던 조인결과로 표현한다. 다시 말해, 이러한 알고리즘들은 튜플의 조인키 값의 과거 분포를 이용해서 미래 튜플의 생산성을 예측한다. 하지만, 많은 실세계 응용들은 미래 튜플의 생산성 예측을 조인키 값만으로 계산하기 어렵다는 것을 말해준다. 조인키 값들은 유일한 경우가 많고(예를 들어, 예제 2의 경우 조인키 값은 패킷아이디로 유일함), 조인키 값에 반복이 있더라도 일정시간(윈도우 시간)내에 반복이 일어나지 않는 경우가 많다(예제 1에서 뉴스는 계속해서 새롭게 생성됨). 만약, 특정 윈도우에서 조인키 값의 반복이 드물게 발생한다면, 그것을 이용해서 미래 튜플의 조인 생산성을 예측하는 것은 의미가 없다. 왜냐하면, 아직 나타나지 않은 값에 대한 예측이 불가능하기 때문이다. 또한, 조인키 값이 반복이 있더라도 그들의 분포가 미래의 데이터 스트림에서 유사할 것이라고 단정할 수도 없다. 이러한 경우들에서 기존 방법들은 효과적으로 동작하지 않을 것이다.

우리가 제안하는 부하감소 알고리즘은 다음 몇 가지 관찰을 통해 튜플의 우선순위를 결정하는 새로운 기준을 제시한다. 많은 응용들에서, 여러 개의 데이터 스트림에 공통적으로 발견되는 객체나 사건들은 그들이 데이터 스트림에서 최초로 나타날 때, 어떤 반복적인 패턴을 보인다. 예제 1에서 주요 뉴스 신문사(한국일보)에서 처음 나타난 뉴스 기사는 조인결과에 포함될 확률(모든 데이터 스트림에 나타남)이 지역 신문사(대전일보)에 처

음 나타난 뉴스 기사에 비해 상대적으로 높다. 또한, 예제 2에서 네트워크 패스가 R_1부터 R_4까지 순차적일 때, R_2나 R_3에 처음 나타난 패킷은 이 패스를 지나갈 확률이 낮을 것이다. 반면, R_1이나 R_4에 처음 나타난 패킷은 이 패스를 지날 확률이 상대적으로 높다고 할 수 있다. 다시 말해, 이들 예제에서 입력 튜플의 생산성 예측은 그 튜플의 조인키 값을 고려하는 것보다 그들 튜플이 데이터 스트림에서 나타나는(혹은 윈도우에 도착하는) 순서를 고려하는 것이 훨씬 효과적일 것이다.

본 논문에서 우리는 튜플이 스트림의 윈도우에 최초로 도착하는 순서에 기준을 두는 새로운 부하감소 알고리즘을 제안한다. 조인 질의에서 데이터 스트림의 개수가 n이라고 하자. 새로운 튜플 t가 데이터 스트림으로부터 조인 윈도우에 도착했을 때, 우리는 n비트로 구성된 비트벡터(이하 존재패턴, existence pattern)를 연관 시킨다. 존재패턴의 i번째 비트는 튜플 t와 같은 조인키 값을 가지는 튜플이 현재 i번째 스트림의 조인 윈도우에 존재하는지를 표현한다. 즉, 각각의 데이터 스트림의 윈도우에 대해, 우리는 같은 존재패턴을 가지는 튜플들을 유지하는 동시에 각 존재패턴의 평균 생산성을 계산한다. 다음으로, 부하감소 상황에서 가장 낮은 생산성을 가지는 존재패턴에 속한 튜플을 제거한다. 다시 말해, 우리는 튜플들의 우선순위를 결정하기 위해 그들의 조인키 값이 아닌, 도착순서를 사용한다.

결과적으로, 제안된 부하감소 알고리즘은 다음의 효과를 창출할 수 있다. (1) 제안하는 알고리즘은 조인키 값의 반복이 거의 없고, 그들이 시간에 따라 다른 분포를 나타내는 응용환경에서, 튜플의 조인 생산성을 효과적으로 예측할 수 있다. (2) 제안하는 알고리즘은 키 값의 분포를 고려하는 기존의 알고리즘들 보다 빠르게 부하감소를 수행 할 수 있다. 본론에서 다루겠지만, 제안된 알고리즘은 튜플의 우선순위를 유지하기 위해, 조인키 값의 범주보다 크기가 작은 존재패턴에 대해 우선순위를 유지하며, 이를 통해 보다 빠르게 우선순위를 결정할 수 있다. 다중 윈도우 조인은 대용량 데이터 스트림을 대상으로 하기 때문에 빠른 연산은 매우 중요한 고려 대상이다.

본 논문의 구성은 다음과 같다. 2장에서는 부하감소의 대상이 되는 다중 윈도우 조인 모델과 관련연구, 그리고 동기부여를 위한 예제를 소개한다. 3장에서 우리는 제안하는 방법의 핵심 아이디어와 알고리즘을 제시하며, 4장에서 복잡도 측면에서 제안하는 알고리즘을 분석한다. 5장에서 다양한 실험을 통해 알고리즘의 성능을 보이고 6장에서 결론을 내리도록 한다.

2. 예비사항

2.1 시스템 모델

먼저, 우리는 다중 데이터 스트림에 대해 다중 윈도우 조인 질의를 처리하는 한 가지 모델을 제시한다. S_1, S_2, ..., S_n으로 표현되는 n개의 데이터 스트림이 있다고 하자. 본 논문에서 우리는 n개의 스트림에 대해 n-웨이 조인 질의 Q를 다음과 같이 표현한다. $Q=S_1 \bowtie S_2 \bowtie ... \bowtie S_n$. 또한, W_i를 $S_i(1 \le i \le n)$의 윈도우라고 한다. 각 W_i는 S_i에 나타나는 튜플들을 저장하고 있으며, 튜플의 개수 혹은 발생시간에 따라 제약조건(예를 들어 튜플 100개 혹은 1시간)을 가지고 있으며, 편의상 지정된 크기의 메모리 공간이 할당된다고 가정한다. 그림 1은 논문에서 사용되는 조인 처리 모델을 보여준다. S_i에서 새로운 튜플 t_i가 나타나면 t_i는 W_i에 삽입되며, 윈도우 제약조건에 의해 W_i에서 제거된다. 이때, t_i는 W_1, ..., W_{i-1}, W_{i+1}, ..., W_n과 조인되며, 그 결과(예를 들어 $W_1 \bowtie ... \bowtie W_{i-1} \bowtie t_i \bowtie W_{i+1} \bowtie ... \bowtie W_n$)는 출력 스트림으로 전달된다.

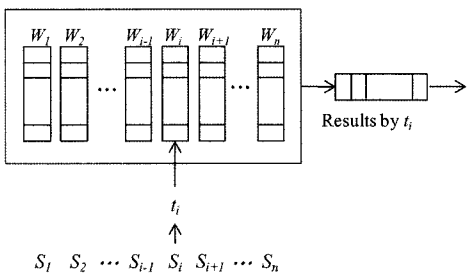


그림 1 다중 윈도우 조인모델

튜플 t_i가 스트림 S_i에서 나타났을 때, 만약 W_i가 가득 찼다면, 우리는 정확한 결과를 생성하기 위해 S_i에 대해 전체 윈도우를 유지할 수 없다. 이러한 경우에, 부하감소 알고리즘은 t_i와 W_i에 있는 튜플 중 어느 것을 버려야 할지 결정해야 한다. 이러한 결정을 위해, 부하감소 알고리즘들은 튜플의 우선순위를 각기 다른 방법으로 결정한다. 다음으로 결정된 우선순위에 의해 그 값이 가장 작은 튜플은 제거되며, 동일할 경우 보통 가장 오래된 것을 제거한다. 조인 질의의 정확성을 높이기 위해 이러한 결정은 조인 결과를 최대화하는 방향으로 진행된다.

2.2 관련 연구

전통적으로, 대부분의 부하감소 알고리즘은 조인키의 값에 근거하여 우선순위를 결정한다. [1]에서, 부하감소의 결정은 가장 출현빈도가 높거나, 이 출현빈도에 윈도우 내에서 튜플의 수명을 곱한 값이 상대적으로 큰 튜플들을 유지하는 방향으로 진행되었다. [10]에서 부하감소는 조인 결과를 가장 많이 생산한 튜플들에 메모리를 최적화하는 방향으로 수행되었다. 하지만, 초기에 부하감소 문제는 두 개의 스트림에 대한 조인문제에 최적화되어 있었다.

현재 여러 응용들에서 조인 문제는 다중 스트림 조인으로 확대되고 있으며, 보다 중요한 문제로 인식되고 있다. 이에 대한 연구로, [4]은 여러 다중 스트림 조인 알고리즘들을 분석했으며, 단위 시간에 조인비용을 최소화할 수 있도록 조인순서를 결정하는 휴리스틱 방법을 제시했고, [11]는 최초로 다중 조인 연산자(MJoin)를 제안했다. 이 방법은 대칭적인 해시조인을 사용하여 단위 시간당 질의의 결과를 최대화한다. 이런 관점에서 우리는 MJoin을 개선한 AMJoin[12]을 제안했다. AMJoin은 BiHT(Bit-vector Hash Table)을 이용하여 입력 튜플의 조인 성공 유무를 상수 시간에 판별함으로써 이들 알고리즘을 발전시켰다.

다중 조인 질의에서 부하감소 문제는 두 개의 스트림에 대한 조인보다 그 복잡도가 높다[7,9]. 이는 각각의 데이터 스트림이 서로 연관성을 가짐으로써, 어떤 하나의 스트림에 대한 부하감소 결정이 전체 조인결과에 미치는 영향이 크기 때문이다. [7]은 스트림 간에 존재하는 시간의 연관성에 대해 언급했다. 그들은 실세계의 우연한 사건들은 스트림 내에서는 독립적일 수 있지만, 스트림 간에는 시간적 연관성이 존재함을 설명하면서, 공통뉴스 검색이나, 이동물체 궤적탐지 등을 예로 들었다. 이 논문에서 조인 윈도우들은 여러 개의 조각으로 나누어지며, 이들 간의 우선순위를 이용하여 CPU 부하감소를 수행했다. 즉, 전체 윈도우를 조인하지 않고 연관성이 높은 조각들만 조인함으로써 조인시간을 단축했다. [9]은 메모리 부하감소를 위해 출현빈도를 기준으로 하는 부하감소 기법을 제안했다. 이 논문에서는 튜플 t의 생산성은 현재 조인 윈도우들에서 자신의 윈도우를 제외한 나머지 것들에서 t의 조인키 값과 동일한 튜플들의 곱으로 표현된다. 또한, 속도 향상을 위해 전체 스트림은 스캔하지 않고 한계 에러 범위 내에서 스케치 기법을 사용한다[6]. 하지만, 이들 모두는 조인키 값을 기준으로 부하감소를 수행하기 때문에, 키 값이 유일하거나 반복이 적은 경우 혹은 값의 분포가 지속적으로 변화하는 경우에는 적절한 해법이 되지 못한다.

2.3 예제

그림 2는 튜플의 도착 순서를 설명하기 위해 만들어진 예제이다. 우리는 이 예제를 만들기 위해 특정 시점에 네 개의 뉴스 사이트를 검색하여 각 사이트에서 다루어지고 있는 메인 뉴스 7개를 발생순서대로 수집하였다. 그림 2에서 각 뉴스들은 키워드로 대표되며, 각각이 고유한 생성시간을 가진다. 각 뉴스 사이트들이 다루는 주요 뉴스들이 일부 상이한 이유는 각 뉴스 매체들의 특징에서 찾아 볼 수 있다. 주요 신문인 한국일보와 경향신문은 해당 신문이 추구하는 가치에 따라 주요 기사를 식별하며, 지역 신문인 대전일보는 해당 지역에 이슈를 주요 기사로 다루게 된다. 이러한 차이점으로 인해 공통 관심사가 되는 뉴스 기사들은 도착(혹은 발생)에 있어 어떤 패턴을 가지게 된다.

각 스트림의 윈도우가 4개의 뉴스기사를 보관할 수 있는 메모리를 가지고 있으며, 2.1절에 설명된 조인 모델을 통해 조인키 '키워드'로 조인한다고 하자. 그림 2의 예제에서 조인키 값을 고려하는 기존의 부하감소 기법은 많은 제약을 가진다. 이것은 각 튜플이 각 스트림의 윈도우 내에서 반복되지 않기 때문에, 현재까지 수집된 조인키 값으로 미래 튜플의 조인 가능을 설명하기 어렵다는 것에서 발생한 결과이다. 이러한 경우 우리는 튜플의 조인키 값을 대신하여 실제 응용에서 발생되는 튜플의 도착 순서를 사용하여 튜플의 조인 가능성을 예측한다. 예제에서 보듯이, 타임 t17에서 S_1에 '재테크전략'을 키워드로 가진 튜플이 나타났을 경우, 기존의 부하감소 기법을 이용할 경우 현재까지 윈도우들에서 출현빈도가 낮은 '신종플루' 혹은 '개성공단'를 제거할 가능성이 높지만, 제안하는 기법에서는 '북핵위협'을 제거한다. 왜냐하면, 현재까지 조인결과(도착순서)를 볼 때, S_1에서 뉴스 기사가 최초로 다루어질 경우 높은 조인 가능성을

S_1(한국일보)

time	Join key
t1	화물연대
t5	신종플루
t9	*개성공단*
t13	북핵위협
t17	재테크전략
t21	월드컵예선
t25	지방선거

S_2(경향신문)

time	Join key
t2	국방대이전
t6	월드컵예선
t10	화물연대
t14	신종플루
t18	달러약세
t22	*개성공단*
t26	북핵위협

S_3(데일리뉴스)

time	Join key
t3	관세행정
t7	북핵위협
t11	화물연대
t15	여기자억류
t19	*개성공단*
t23	달러약세
t27	신종플루

S_4(대전일보)

time	Join key
t4	국방대이전
t8	지방선거
t12	화물연대
t16	교과부지원
t20	*개성공단*
t24	재테크전략
t28	신종플루

그림 2 주요 뉴스 검색에 대한 실행 예제

가지기 때문이다.

그림 2의 조인 예제는 메모리 제한에도 불구하고, 도착 순서를 기준으로 부하감소를 수행할 경우, 제한이 없을 때와 동일한 정확한 조인 결과를 얻을 수 있다. 이러한 응용의 예는 1장에서 설명한 네트워크 모니터링 외에 이동물체 궤적 탐지나, 오염물질을 탐지하는 센서 모니터링 등 다양한 것에서 찾아 볼 수 있다. 즉, 객체나 사건들이 여러 개의 데이터 스트림 소스에서 독립적으로 탐지되지만, 그들 간에 시간적인 연관성이 있는 경우, 우리는 그들의 도착(혹은 발생)순서의 반복적인 패턴을 이용하여 지금까지와는 다른 기준으로 부하감소의 정확도를 높일 수 있다. 다음 장에서는 이러한 도착순서 패턴을 이용할 수 있는 정확하고 간편한 부하감소 알고리즘을 제시한다.

3. 제안하는 방법

3.1 튜플의 생산성

부하감소 시에, 각 튜플의 우선순위는 그것의 생산성에 의해서 결정된다. 개별 튜플의 생산성은 그 튜플에 의해 생산된다고 기대되는 결과물의 개수로 표현될 수 있다. 이전 알고리즘들과 비교했을 때, 우리가 제안하는 방법은 튜플의 생산성을 조인키 값 자체가 아닌 그 값들이 데이터 스트림에 도착하는 순서에 기준을 두어 판단한다. 먼저, 우리는 튜플의 도착 순서를 표현하기 위해 존재패턴 *EP(Existence Pattern)*을 사용하며, 이를 다음과 같이 정의한다.

정의 1. *존재패턴*

조인키를 A_1, A_2, ..., A_m이라 하고, $t.A$를 튜플 t의 조인키 값이라 하자. 튜플 t_i를 데이터 스트림 S_i에서 도착한 것이라고 할 때, t_i의 존재패턴 $ep(t_i)$를 n(n은 스트림의 개수) 비트로 이루어진 존재패턴으로 표시한다. 거기서, j번째 값은 W_j의 튜플 t_j가 $t_i.A_1 = t_j.A_1$, $t_i.A_2 = t_j.A_2$, ..., $t_i.A_m = t_j.A_m$를 만족할 경우 1로 표현되고, 만족하지 못할 경우 0으로 표현된다.

예를 들어, 그림 3의 예제는 3-way 조인에서 현재 윈도우를 보여준다. 여기서, *BiHT*는 현재 윈도우들에서 튜플들의 존재유무를 보여주고 있다. 만약, W_1에 신규 튜플 e가 도착했을 경우 해당 튜플의 존재패턴은 ep(e) = '100'이며, 같은 방법으로 W_3에 신규 튜플 b가 도착한다면 ep(b) = '011'이다. 이 값은 BiHT의 bit-vector와 달리 해당 튜플의 고유 값으로 변경되지 않으며, 해당 튜플이 조인 윈도우에 어떤 순서에 의해 도착했는지를 표현한다. n개의 데이터 스트림에 대해, 각 튜플이 가질 수 있는 존재패턴은 $2^{(n-1)}$개의 가지 수 이며, W_i에 있는 튜플의 $ep(t)$의 i번째 비트의 값은 항상 1이다.

데이터 스트림에서 조인 윈도우로 새 튜플이 도착할 때마다, 그 튜플의 고유한 존재패턴 $ep(t)$를 지정한다. 이를 위해, 우리는 각 데이터 스트림 S_i마다 존재패턴테이블(Existence pattern Table, *EPT*)를 유지하며, 다음과 같이 정의한다.

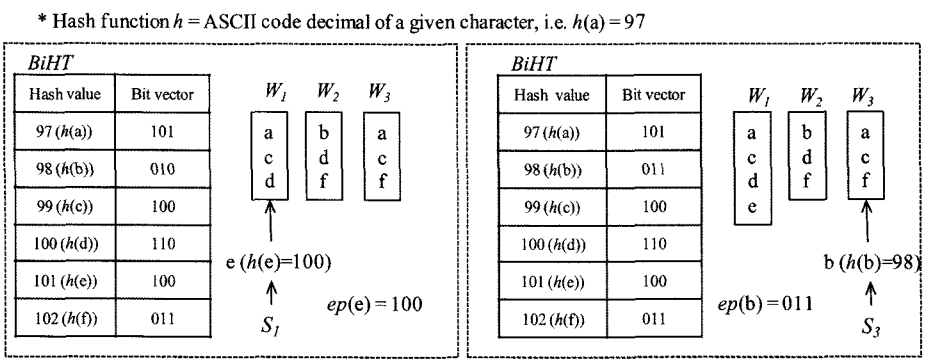
정의 2. *존재패턴테이블*

각 데이터 스트림 S_i에 대해, 존재패턴테이블 EPT_i는 $2^{(n-1)}$개의 로우(row)로 구성된다. EPT_i의 각 로우는 해당 존재패턴 ep, 해당 패턴을 가지는 튜플의 리스트 $l_i(ep)$ 그리고 해당 ep를 가지는 튜플이 현재까지 생성한 평균 조인결과(리스트에 포함된 튜플이 생성한 총 조인결과의 개수/리스트에 포함된 전체 튜플의 개수)의 $p_i(ep)$를 가진다.

여기서, $p_i(ep)$는 해당 존재패턴의 생산성을 말한다. 즉, 우리는 $p_i(ep)$를 이용하여 존재패턴들 간의 우선순위를 결정한다.

정의 3. 튜플의 생산성(*Productivity*)

튜플 t에 대해, 그것의 존재패턴을 $ep(t)$라 하면, t의 생산성은 $p_i(ep(t))$로 정의한다.



(a) W_1에 새로운 튜플 e가 도착했을 경우 (b) W_3에 새로운 튜플 b가 도착했을 경우

그림 3 3-way 조인에서 존재패턴의 예제

EPT_1

ep	$l_1(ep)$	$p_1(ep)$
100	$t_1,t_2,\ldots$	0.1
101	$t_3,t_4,\ldots$	0.6
110	$t_5,t_6,\ldots$	0.4
111	$t_7,t_8,\ldots$	1

EPT_2

ep	$l_2(ep)$	$p_2(ep)$
010	$t_{21},t_{22},\ldots$	0.4
011	$t_{23},t_{24},\ldots$	0.8
110	$t_{25},t_{26},\ldots$	0.1
111	$t_{27},t_{28},\ldots$	1

EPT_3

ep	$l_3(ep)$	$p_3(ep)$
001	$t_{41},t_{42},\ldots$	0.8
011	$t_{43},t_{44},\ldots$	0.4
101	$t_{45},t_{46},\ldots$	0.1
111	$t_{47},t_{48},\ldots$	1

그림 4 3-way 조인에서 도착패턴테이블

그림 4는 3-way 조인에서 각각의 스트림 S_i가 가지고 있는 존재패턴테이블에 대한 예제이다. EPT_1은 S_1에서 발생 가능한 모든 존재패턴(100, 101, 110, 111)을 가지고 있으며, 각각의 존재패턴으로 도착한 튜플의 리스트와 그들의 생산성을 실시간으로 수정한다. 그림 4에서 보듯이 EPT_1의 튜플 t_3는 ep로 '101'을 가지며 윈도우에 삽입될 당시 윈도우 W_3에 해당 튜플과 같은 조인키 값을 가지는 튜플이 존재했다는 것을 알 수 있습니다. 이러한 방식으로 존재패턴은 각 튜플의 도착순서를 표현한다. 우리가 제안하는 방법은 위에 정의 3의 $p_i(ep(t))$가 높은 튜플에 대해 높은 우선순위를 부여한다. 즉, 우리는 이것을 이용하여, 부하감소가 필요할 때, 가장 낮은 $p_i(ep(t))$를 가진 튜플 중 가장 오랫동안 윈도우에 있는 튜플을 제거한다. 이처럼 조인키의 실제 값을 사용하는 대신 그 값들의 도착순서 패턴을 기준으로 튜플의 생산성을 결정하는 방식은 다음과 같은 상황에서도, 도착순서에 패턴이 존재한다면 효과적으로 부하감소를 수행할 수 있다. (1) 조인키 값이 유일하거나 반복인 없는 경우. (2) 조인키 값의 분포가 지속되지 않는 경우. 다음 절에서는 이번 절 정의된 튜플의 생산성을 기준으로 부하감소를 처리하는 간편한 알고리즘을 설명한다.

3.2 부하감소 알고리즘

우리가 제안하는 부하감소 알고리즘은 해시조인(Hash Join)을 사용하는 AMJoin[12]의 프레임워크(Framework)를 사용한다. $BiHT$와 EPT_i ($1 \le i \le n$)는 조인 윈도우 $W_i(1 \le i \le n)$와 함께 메모리에서 유지된다. 설명의 편의를 위해 모든 조인 키들이 동일하다고 가정하며, 한 개의 해시함수 h를 사용한다고 가정한다. 데이터 스트림 S_i에서 튜플 t_i가 도착하면, 다음의 조인과정이 연속적으로 실행된다.

1. 해시: S_i에서 입력된 튜플 t_i의 조인키 A에 대해 해시 값을 구한다. 예를 들어, $v = h(t_i.A)$.
2. 입력:
   (1) W_i에 할당된 메모리가 가득 차고, 윈도우 제약에 의해 삭제될 튜플이 없다면, EPT_i에서 가장 작은 $p_i(ep)$를 가진 로우의 $l_i(ep)$에서, 가장 오래된 튜플을 삭제하고, t_i를 W_i에 삽입한다. 그렇지 않은 경우, t_i를 W_i에 바로 삽입한다.
   (2) 튜플 t_i를 해시 테이블 W_i에 삽입할 때, $BiHT$의 해시주소 v의 비트벡터 b를 가져와서 i번째 비트를 1로 수정한 다음(현재 i번째 비트가 1이라면, 수정 불필요), b를 t_i의 존재패턴 $ep(t_i)$로 지정한다. 이와 동시에, EPT_i의 $l_i(ep(t_i))$에 t_i의 링크 값을 저장하고 $p_i(ep)$를 수정한다.
3. 조인: t_i의 비트벡터 b가 모두 1인지 체크하고, 체크가 성공하면 $W_1 \bowtie \ldots \bowtie W_{i-1} \bowtie t_i \bowtie W_{i+1} \bowtie \ldots \bowtie W_n$를 수행하여, 조인결과를 스트림으로 출력한다. 그렇지 않은 경우, 조인 과정은 중단된다. 만약, t_i가 조인결과를 생성하였다면, 조인결과에 포함된 모든 튜플 t_j($(1 \le j \le n)$에 대하여, 생산성 $p_i(ep(t_j))$를 수정한다.

※본 알고리즘에서는 편의상 윈도우 제약조건에 따른 튜플의 제거(expire)는 구체적으로 언급하지 않는다. 또한, 튜플의 제거에 따른 $BiHT$의 수정은 필요하지만, 과거로부터 현재까지의 도착패턴의 누적값을 유지하는 EPT_i에서의 수정은 불필요하다.

제안하는 알고리즘은 입력(1)에서 윈도우 W_i의 메모리가 가득 찼을 경우, 정의 1, 2, 3에 의해서 W_i에서 가장 생산성이 낮은 존재패턴을 가진 튜플 중에서 가장 오래된 튜플을 제거한다. 그림 5는 3-웨이 조인에서 제안한 부하감소 알고리즘의 실행예제를 보여준다. 3개의 네트워크 라우터가 있다고 하자. 이중 라우터 R_1, R_2, R_3를 지나는 패스의 패킷 사용량을 모니터링 한다고 하자. 각 라우터는 튜플을 발생시키고, 각 튜플은 라우터를 통과하는 패킷의 정보를 가지고 있다. 각 패킷정보는 패킷아이디(여기서는 알파벳)로 유일하게 식별된다고 가정한다. 우리는 해당 패스를 지나는 패킷 사용량을 모니터링하기 위해 3-way 조인을 수행한다. 각 윈도우가 4개 이상의 패킷정보를 보관할 수 없다고 가정할 때, 각 윈도우에 다섯 번째 튜플이 도착할 때 부하감소가 수행된다. 그림 5의 예제에서는 두 개의 생산성이 높은 도착패턴이 존재한다. $R_1 \rightarrow R_2 \rightarrow R_3$와 $R_3 \rightarrow R_2 \rightarrow R_1$가 그것이

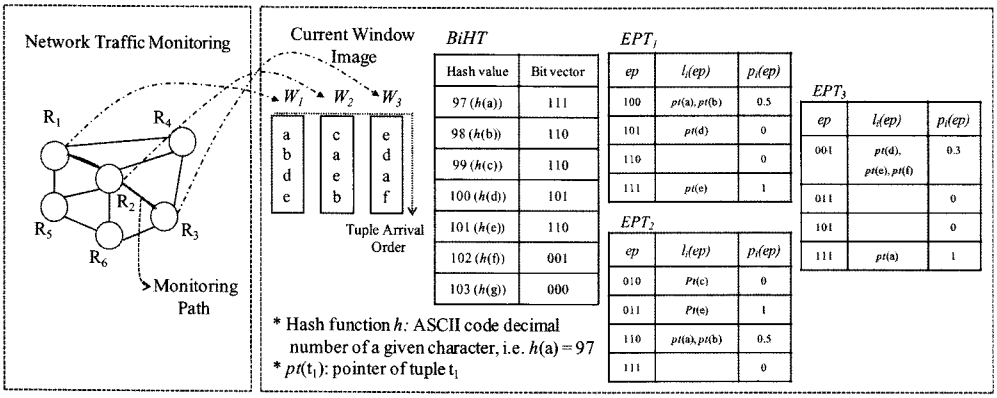


그림 5 3-way 조인에서 부하감소 실행예제

다. 즉, W_1에서는 존재패턴 '100'과 '111'을 가진 튜플이 가장 생산성이 높고, W_3에서는 존재패턴 '001'과 '111'을 가진 튜플이 가장 생산성이 높고, 같은 방식으로 W_2에서는 존재패턴 '110'과 '011'이 높다. 여기서, R_1이나 R_3로부터 새로운 튜플이 도착한다면 제안하는 알고리즘은 튜플 d를 제거하며, R_2로부터 새로운 튜플이 도착한다면 튜플 c를 제거한다. 하지만, 기존 알고리즘들은 이러한 상황에서 부하감소를 위한 적절한 기준을 제시하기 어렵다. 출현빈도 기반 방법은 W_1과 W_3에서 누적 출현빈도가 낮은 튜플 b와 f를 제거할 가능성이 높다. 이 경우 W_1에서 튜플 b는 도착패턴을 고려할 때, 조인될 가능성이 높은 튜플이며, W_3에 튜플 f는 튜플 d보다는 조인 가능성이 높은 튜플이다(그림 5에서 d는 R_2를 거치지 않을 가능성이 높음). 또한, 결과기반 방법도 현재까지 조인결과를 내지 않은 튜플은 우선순위가 동일하기 때문에 적절한 기준이 될 수 없다.

4. 분 석

이번 장에서는, 제안하는 알고리즘의 시간 및 공간 복잡도(Complexity)에 대해서 기술한다. 기본적으로, 우리가 제안하는 방법과 2장 관련연구에서 언급한 기존 방법들은 모두 해시기반 조인을 사용한다고 가정한다. 이 경우 부하감소를 위해 가장 많은 비용이 소요되는 부분은 우선순위를 결정하고 유지하는 부분이다.

먼저, 시간 복잡도 측면에서, 튜플당 소요되는 생산성 계산과 이들의 우선순위를 유지하는 비용은 대용량의 스트림을 대상으로 하기 때문에 그 파급효과가 크다. 조인키 값의 출현빈도를 기준으로 부하감소를 수행하는 방법의 경우에 신규 튜플이 조인 윈도우에 도착할 때마다 자신을 제외한 나머지 윈도우들에서 그 튜플의 짝을 찾아야 하는 비용이 소요된다. 이에 반하여, 결과 기준으로 부하감소를 수행하는 경우 기본적으로 조인결과를 생성했을 경우에만 우선순위를 결정하면 되기 때문에 오버헤드가 크지 않다. 또한, 우리가 제안하는 방법도 후자와 유사하다고 볼 수 있다. 하지만, 우선순위를 유지하는 부분에 있어서는 큰 차이를 보인다. 통상적으로 우선순위를 유지하기 위해 대부분의 부하감소 기법들은 우선순위 큐를 사용한다. 우선순위는 어떤 집단에서 최소(혹은 최대) 값을 추출하는데 유용하게 사용된다. 예를 들어, 이 데이터 구조는 O(log$N$)의 시간 복잡도를 요구한다. 여기서, N은 서브셋(subsets)의 개수를 의미한다. 이 관점에서 기존 알고리즘들은 해시 엔트리의 개수를 N으로 가지게 되며, N값이 커지게 되면, 복잡도가 증가한다. 이에 반해 우리가 제안하는 알고리즘은 EPT 엔트리에 대해 우선순위를 유지하기 때문에 발생 가능한 존재패턴 $2^{(n-1)}$(n은 스트림의 개수)를 N으로 가진다. 물론, 조인결과를 생산성에 반영하기 위해 모든 EPT를 수정하기 때문에 시간 복잡도는 O(n^2)으로 볼 수 있다. 통상적으로, 여러 응용들에서 조인에 참여하는 스트림의 개수가 많지 않기 때문에 우리가 제안하는 알고리즘의 복잡도는 기존 방법들에 비해 크지 않고, 조인키 값의 범위에 영향을 받지 않는 장점을 가진다.

다음으로 공간 복잡도 측면에서, 기존 알고리즘은 각 해시테이블에 대해 조인키 값의 생산성을 우선순위 큐로 유지하기 위한 추가적인 메모리(예를 들어 5장 실험에서 개별 해시 엔트리 마다 해시값 및 생산성 저장을 위해 8byte)가 필요하다. 또한, 조인 성능 향상을 위해 본 논문에서 사용하는 AMJoin을 사용한다면, BiHT를 유지하기 위한 메모리가 필요하다. BiHT를 이용할 경우 발생되는 추가적인 공간 오버헤드는 조인키 값의 범위에 영향을 받으며, 엔트리당 해시주소를 위한 4byte와 비트벡터를 위한 2byte로 총 6byte가 필요하다. 이에 반하여 제안하는 알고리즘은 BiHT를 필수로 하며 EPT를 유지하기 위한 메모리 공간과, 각 튜플의 존재패턴

기록을 위해 추가적인 메모리가 필요하다(예를 들어 5장의 실험에서 우리는 2bytes의 필드를 각 튜플의 레코드에 추가했다). 공간 복잡도의 경우는 튜플의 개수와 해시 테이블의 엔트리의 개수에 영향을 받기 때문에 실제 응용 환경에 따라 차이가 있을 수 있다.

5. 실 험

이장에서 우리는 제안한 알고리즘의 효과성(정확성과 속도)을 입증하기 위해, 존재하는 다양한 알고리즘들과 비교를 수행한다. 비교 대상이 되는 알고리즘들은 다음과 같다. (1) 출현빈도 기준 알고리즘: 입력 튜플에 대해 자신의 윈도우를 제외한 나머지 윈도우들에서 그것의 짝들의 곱을 계산하여 우선순위를 표현. (2) 결과 기준 알고리즘: 입력 튜플에 대해 그 튜플이 과거에 생성했던 결과 튜플의 개수를 누적하여 우선순위를 표현. (3) 무작위 기준 알고리즘: 입력 튜플을 제외한 나머지 튜플들에서 무작위로 부하감소를 수행함. 또한, 세 개의 알고리즘은 3장에서 설명한 AMJoin의 프레임워크를 적용하며, 부하감소의 우선순위를 결정하는 부분만 다르게 적용한다. 공통적인 프레임을 적용하는 구체적인 이유는 이장에 마지막에 기술하도록 하겠다.

먼저, 우리는 실험을 위해 데이터 생성모듈을 구현했다. 이 모듈은 입력 스트림의 개수, 각 스트림당 튜플의 개수, 조인키 값의 범위와 분포 등 다양한 입력 파라메터를 기준으로 가상 데이터를 생성한다. 각 튜플은 조인키 값과 발생시간, 그리고, 기타 필드를 가지고 있다. 우리는 4가지의 범주에 대하여 실험을 수행하였고, 그 결과는 다음과 같다.

유일한 조인키 값을 가진 스트림에 대한 성능차이

앞서 기술한 3가지 알고리즘과 우리가 제안한 알고리즘(EP)에 대해, 조인키 값이 유일한 경우(uniform distribution)에 대해 5-웨이 조인을 수행하였다. 그림 6의 실험에서는 각 스트림에서 유일한 조인키 값(1에서 20,000)으로 튜플 20,000개를 생성하고, 각 윈도우에서 이중 75%(15,000개)만을 저장할 수 있도록 메모리를 할당하였다. 각 스트림에서 무작위 순으로 튜플이 도착할 때를 'no pattern'으로 실험하고, 도착순서에 시간적인 연관성을 부여했을 때를 'pattern'으로 실험하였다. 여기서, 시간적인 연관성은 스트림이 S_1, S_2, S_3, S_4, S_5 순으로 도착하였을 경우, 그렇지 않은 경우보다 높은 조인 성공률을 가지도록 설정한다는 의미이다. 또, 그림 7의 실험은 이전 실험의 'pattern' 상황에서 각 윈도우에 할당되는 메모리 공간을 다르게 설정하여 출력되는 결과의 정확성을 비교하였다. 두 가지 실험의 결과에서 보듯이 유일한 조인키 값을 가지는 상황에서 출현빈도와 결과기준 알고리즘은 무작위로 부하감소를 수행하는 알고리즘과 유사한 정확도를 보이지만, 제안하는 알고리즘(EP)는 최고 30%의 높은 정확도를 달성할 수 있다. 다시 말해, 제안하는 알고리즘은 스트림의 도착순서가 존재하는 경우, 키 값이 유일한 경우에도 우수한 조인성능을 보임을 알 수 있다.

조인키 값의 분포가 정규분포이고, 스트림 간에 조인키 값의 시간 연관성이 있는 경우

두 번째 범주의 실험은 조인키 값의 분포가 정규분포를 따르는 경우에 대한 실험이다. 각 스트림에서 튜플은 조인키 값에 대해 동일한 분산과 평균을 가진다. 또한, 모든 스트림은 동일한 분산을 가지되, 그 평균을 각기 다르게 설정하였다. 다중 윈도우 조인에서 조인 선택치는 상당히 낮게 나타나기 때문에 이러한 설정은 실세계 응용을 고려할 때 적절하다[8,13]. 그림 8은 조인키 값에 대해 인위적인 시간 연관성을 주지 않은 'no pattern'과 도착시간에 대해 각기 다른 지연(delay)을 주어 인위적인 순서패턴을 부여한 'pattern'의 실험결과를 보여준다. 또한, 그림 9는 그림 8의 'pattern'상황에서 각 윈도우에 할당되는 메모리 공간의 차이에 따른 결과를 보여준다. 두 가지 실험에서 제안하는 알고리즘이 우수한 성능을

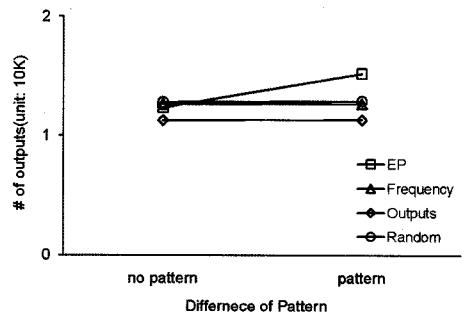


그림 6 유일한 조인키 값에서 패턴 존재 유/무에 따른 차이

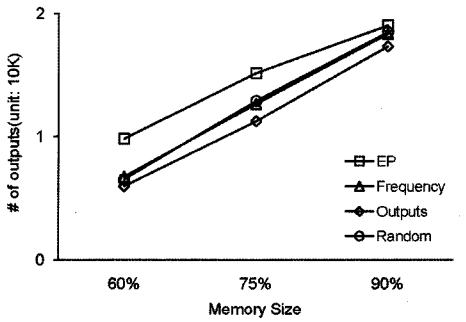


그림 7 유일한 조인키 값에서 패턴이 존재하는 경우 윈도우 크기 차이

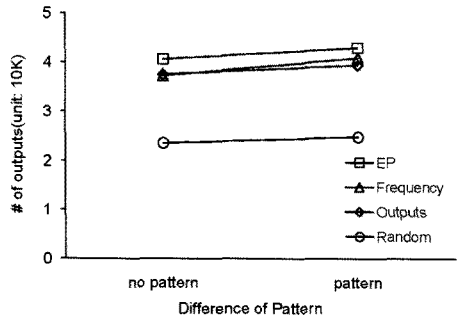


그림 8 정규분포에서 패턴 존재 유/무에 따른 차이

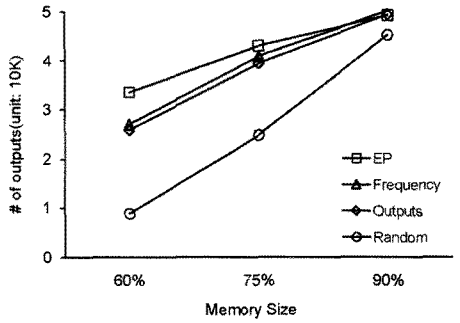


그림 9 정규분포에서 패턴이 존재하는 경우 윈도우 크기 차이

보이고 있음을 알 수 있다. 여기서, 출현빈도와 결과 기준 알고리즘이 낮은 성능을 보이는 이유는 다음 두 가지에 있다. 먼저, 스트림 간에 조인키 값의 분포가 상이할 경우 두 가지 알고리즘은 이를 반영하기 어렵다. 예를 들어, 스트림 S_1에서 조인될 가능성이 높은 튜플이 S_2에는 그 가능성이 낮아서 지워질 수 있다. 두 번째는 제안하는 알고리즘이 도착순서 패턴의 탐지를 통해 부하감소의 정확도를 높일 수 있음에 있다.

조인키 값의 분포가 정규분포이고, 도착순서 패턴이 반복되는 경우

그림 10의 실험은 앞선 실험과 유사하지만, 도착순서 패턴이 지속적으로 반복되게 설정하였다. 해당 실험에서는 스트림 간에 조인키 값의 분산에 차이를 두어, 스트림 순(예를 들어 S_1, S_2, S_3, S_4, S_5 순으로 앞쪽 스트림의 분산이 작게 설정)으로 같은 조인키 값을 가질 경우, 튜플이 앞쪽 스트림에서 최초로 나타날 경우 조인될 가능성이 높게 설정하였으며, 이러한 패턴을 반복되게 적용하였다. 그림 10에서 제안하는 알고리즘과 기존 알고리즘들은 정확도에 있어서 최대 20%까지의 차이를 보인다.

조인 시간의 차이

그림 11은 조인에 수행되는 시간을 비교하기 위해, 그림 10의 실험에서 Y축에 수행 시간을 표시한 그래프이다. 해당 실험에서 조인키 값의 범주는 20,000개 정도로 한정했기 때문에 제안하는 알고리즘과 결과 기준 알고리즘은 거의 차이가 없지만, 출현빈도 기반 알고리즘은 비교적 많은 시간이 소요된다. 출현빈도 기반 알고리즘의 경우 튜플이 입력될 때마다 그 출현빈도를 지속적으로 체크하여 우선순위를 조정하기 때문에 다른 알고리즘(제안하는 알고리즘과 결과 기반은 튜플이 조인결과를 생성할 때에만 우선순위를 조정함)에 비해 비교적 느리다. 이런 단점을 보완하기 위해 [9]에서는 어느 정도의 에러를 감수하고 스케치 기법[6]을 사용하여 출현빈도를 계산하였다.

여러 실험들에서 모든 알고리즘은 수행속도의 개선을 위해 AMJoin의 프레임 워크를 이용하였다. 제안하는 알고리즘을 제외한 나머지 알고리즘은 필수적으로 BiHT를 사용할 필요가 없다. 4장 분석에서 언급한 바와 같이 BiHT는 추가적인 공간을 요구하기 때문에, 우리는 각각의 알고리즘이 필요로 하는 메모리 공간을 세밀하게 분석할 필요가 있다. 먼저, 모든 실험에서 BiHT의 크기

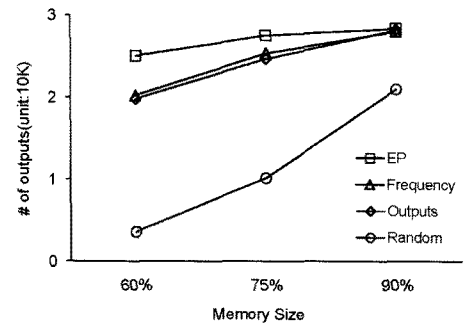


그림 10 정규분포에서 도착순서 패턴의 반복에 따른 윈도우 크기의 차이

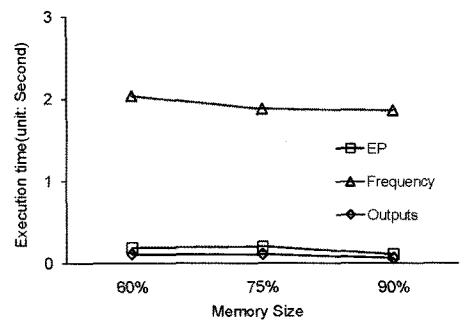


그림 11 정규분포에서 도착순서 패턴의 반복에 따른 조인 성능(시간) 차이

는 120Kbyte(6Bytes(엔트리당 해시주소 4bytes와 비트벡터 2bytes)*20,000(엔트리 개수))로 비교적 작은 공간을 차지하기 때문에 이를 사용하는데 대한 영향이 적다. 출현빈도와 결과 기반 알고리즘은 부하감소를 위해 8Mbytes (8bytes(해시주소 4bytes와 생산성 값 4bytes)*5(스트림개수)*200,000(엔트리 개수))를 필요로 하고, 제안하는 알고리즘은 존재패턴테이블을 위해 약 400Kbytes (4bytes(튜플의 리스트를 유지하기 위한 포인터)*100,000(전체 튜플의 개수))와 튜플의 존재패턴 기록을 위해 200KBytes(2Bytes(존재패턴)*100,000(전체 튜플의 개수))를 합해서 600Kbytes가 필요하다. 여기서, 제안하는 알고리즘이 작은 메모리를 요구하는 이유는 우선순위 큐가 상대적으로 작기 때문이다. 우리는 5개의 스트림을 조인하기 위해 각 존재패턴테이블을 위해 16개의 엔트리만을 요할 뿐이다. 결과적으로 기존 알고리즘들이 BiHT를 사용하지 않더라도 AMJoin 프레임워크를 이용하는 제안하는 알고리즘은 더 작은 메모리 공간을 사용한다.

6. 결 론

본 논문에서 우리는 다중 윈도우 조인에서 도착순서 패턴을 가진 경우에 효과적으로 부하감소를 수행할 수 있는 새로운 알고리즘을 제안했다. 제안하는 알고리즘은 실세계 응용들에서 자연발생적으로 생기는 튜플들의 도착순서의 패턴을 이용하여 부하감소를 수행한다. 이는 조인키 값의 분포에 근거하여 튜플의 조인 가능성을 판별하는 기존 알고리즘들이 가지는 한계점을 해결하는데 효과적이다. 즉, 조인키 값이 유일하거나 반복이 없는 경우 혹은 조인키 값에 중복이 있더라도, 스트림 간에 분포가 상이한 경우에도 제안하는 알고리즘은 상대적으로 우수한 성능을 발휘한다. 우리는 다양한 실험과 분석을 통해 제안하는 알고리즘의 효과성을 검증하였다.

참 고 문 헌

[1] A. Das, J. Gehrke and M. Riedewald. Approximate Join Processing over Data Streams. *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, San Diego, California, USA, pp.40-51, 2003.

[2] C. Cranor, T. Johnson, O. Spataschek and V. Shkapenyuk. Gigascope: A Stream Database for Network Applications, *Proceedings of the ACM SIGMOD International Conference On Management of Data*, San Diego, California, USA, pp.647-651, 2003.

[3] J. Gehrke and S. Madden. Query Processing in Sensor Networks. *IEEE Pervasive computing*, vol.3, no.1, pp.46-55, 2004.

[4] L. Golab and M. T. Ozsu. Processing Sliding Window Multi-Joins in Continuous Queries over Data Streams. *Proceedings of the 29th International Conference on Very Large Data Bases*, Berlin, Germany, vol.29, pp.500-511, 2003.

[5] M. A. Hammad, W. G. Aref and A. K. Elmagarmid. Stream Window Join: Tracking Moving Objects in Sensor-Network Databases. *Proceedings of 15th International Conference on Scientific and Statistical Database Management*, Cambridge, Massachusetts, USA, pp.75-84, 2003.

[6] A. Dobra, M. Garofalakis, J. Gehrke, and R. Rastogi. Processing complex aggregate queries over data streams. *Proceedings of the 2002 ACM SIGMOD international conference on Management of data, Madison*, Wisconsin, USA, pp.61-72, 2002.

[7] B. Gredik, K. Wu, P. S. Yu and L. Liu. A Load Shedding Framework and Optimizations for M-way Windowed Stream Joins. *IEEE 23rd International Conference on Data Engineering*, pp. 536-545, 2007.

[8] Y. Bai, H. Wang and C. Zaniolo. Load Shedding in Classifying Multi-Source Streaming Data: A Bayes Risk Approach. *Proceedings of the Seventh SIAM International Conference on Data Mining*, Minneapolis, Minnesota, USA, pp.425-430, 2007.

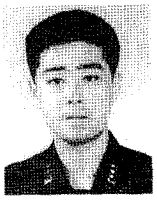
[9] Y. Law and C. Zaniolo. Load Shedding for Window Joins on Multiple Data Streams. *IEEE 23rd International Conference on Data Engineering*, pp.674-683, 2007.

[10] U. Srivastava and J. Widom. Memory-Limited Execution of Windowed Stream Joins. *Proceedings of the 30th VLDB Conference*, Toronto, Canada, pp.324-335, 2004.

[11] S. D. Viglas, J. F. Naughton and J. Burger. Maximizing the Output Rate of Multi-Way Join Queries over Streaming Information Sources. *Proceedings of the 29th International Conference on Very Large Data Bases, Berlin, Germany*, vol.29, pp.285-296, 2003.

[12] T. Kwon, H. Kim, M. Kim and J. Son, An Advanced Join Algorithm for Multiple Data Streams Using a Bit-vector Hash Table. *IEICE Transaction on Information and Systems*, vol.E92-D, no.7, pp.1429-1434, 2009.

[13] H. Yu, EP. Lim and J. Zhang. On In-network Synopsis Join Processing for Sensor Networks. *Proceedings of the 7th International Conference on Mobile Data Management*, Nara, Japan, pp.32-39, 2006.



권 태 형

1995년 3월 공군사관학교 전자계산학과 (학사). 2002년 2월 국방대학교 전산정보학과(석사). 2006년~현재 한국과학기술원 전산학과 박사과정. 관심분야는 데이터베이스 시스템, 센서 네트워크, 스트림 데이타 처리 등

이 기 용

1998년 2월 KAIST 전산학과 학사. 2000년 2월 KAIST 전산학과 석사. 2006년 2월 KAIST 전자전산학과 전산학전공 박사. 2006년 3월~2008년 2월 삼성전자 기술총괄 소프트웨어연구소 책임연구원 2008년 3월~현재 KAIST 전산학전공 연구조교수. 관심분야는 Data warehouse, OLAP, Embedded DB

손 진 현

1996년 서강대학교 전산학과 학사. 1998년 한국과학기술원 전산학과 석사. 2001년 한국과학기술원 전자전산학과 박사 2001년 9월~2002년 8월 한국과학기술원 전자전산학과 박사후 연구원. 2002년 9월~현재 한양대학교 컴퓨터공학과 조교수. 관심분야는 데이터베이스, 온디맨드 서비스, 유비쿼터스 컴퓨팅, 임베디드 소프트웨어, 분산 데이터 처리

김 명 호

1982년 서울대학교 컴퓨터공학과 학사. 1984년 서울대학교 컴퓨터공학과 석사. 1989년 미국 Michigan State University 전산학과 박사. 현재, 한국과학기술원 전산학전공 교수. 관심분야는 데이터 베이스 시스템, 분산시스템, XML, 센서 네트워크 등