비순서화된 스트림 처리를 위한 적응적 버퍼 제어 기법

(Adaptive Buffer Control over Disordered Streams)

김 현 규 † 김 철 기 †† 이 충 호 ††† 김 명 호 ††††
(Hyeon Gyu Kim) (Cheol Gi Kim) (Chung Ho Lee) (Myoung Ho Kim)

요 약 비순서화된 스트림은 윈도우 기반의 질의를 처리할 때 부정확하거나 지연된 결과를 유발할 수 있다. 기존의 방식에서는 일반적으로 버퍼를 이용하여 비순서화된 스트림을 정렬하며, 버퍼의 크기를 추정하기 위해 네트워크 지연의 최대값에 기반한 방식을 이용한다. 그러나 이러한 방식은 버퍼의 크기를 불필요하게 큰 값으로 추정할 수 있으며, 지연된 질의 결과를 발생시킬 수 있다. 본 논문에서는 네트워크 지연의 변화에 따라 적응적으로 버퍼의 크기를 추정하기 위한 확률론적인 접근 방법을 제안한다. 제안하는 방법에서는 튜플의 생성이 포아송 분포를 따르며, 네트워크 지연은 정규 분포를 따른다고 가정한다. 그리고 이러한 가정을 바탕으로 추정식을 유도한다. 추정식은 튜플의 손실율을 입력인자로 요구하며, 이는 실시간에 튜플의 손실에 있어서 허용 가능한 백분율을 나타낸다. 사용자는 손실율을 질의문에서 정의함으로써, 응용의 요구에 따라 질의 결과의 정확성이나 처리속도 중 원하는 특성에 중점을 둘 수 있다. 본 논문의 실험 결과는 제안한 추정식이 기존의 네트워크 지연의 최대값에 기반한 추정식에 비해 적응성이 우수함을 보인다.

키워드 : 데이타 스트림, 비순서, 손실율, 적응성

Abstract Disordered streams may cause inaccurate or delayed results in window-based queries. Existing approaches usually leverage buffers to handle the streams. However, most of the approaches estimate the buffer size simply based on the maximum network delay in the streams, which tends to over-estimate the buffer size and result in high latency. In this paper, we propose a probabilistic approach to estimate the buffer size adaptively according to the fluctuated network delays. We first assume that intervals of tuple generations follow an exponential distribution and network delays have a normal distribution. Then, we derive an estimation function from the assumptions. The function takes a *drop ratio* as an input parameter, which denotes a percentage of tuple drops permissible during query execution. By describing the drop ratio in a query specification, users can control the quality of query results such as accuracy or latency according to application requirements. Our experimental results show that the proposed function has better adaptivity than the existing function based on the maximum network delay.

Key words : Data Streams, Disorder, Drop Ratios, Adaptivity

1. 서 론

최근 데이타베이스 분야에서 연속 스트림(Continuous Stream)을 지원하기 위한 많은 연구가 진행되었다[1-3]. 온라인 주식정보나 모니터링 정보와 같이 끊임없이 생성되어 전달되는 튜플을 실시간에 처리하는데 있어 릴레이션 개념을 적용하여 문제를 해결하고자 하였으며, 이러한 시도는 곧 DSMS(Data Stream Management System)로 구체화되었다. DSMS에서 스트림의 처리 방법은 질의언어를 이용하여 선언적으로 정의될 수 있으며, 정의된 명세로부터 생성된 질의 트리는 실행시간에 스트림 튜플의 처리에 이용된다[4,5]. DSMS의 대표적인 예로는 STREAM[6], TelegraphCQ[7], Aurora/Borealis[8,9] 등을 들 수 있다.

† 학생회원 : 한국과학기술원 전산학과
hgkim@dbserver.kaist.ac.kr
†† 정 회 원 : 한국정보통신대학교 전산학과 교수
hgkim@dbserver.kaist.ac.kr
††† 정 회 원 : 한국전자통신연구원 텔레메틱스·USN연구단 선임연구원
leech@etri.re.kr
†††† 종신회원 : 한국과학기술원 전산학과 교수
mhkim@dbserver.kaist.ac.kr
논문접수 : 2006년 4월 19일
심사완료 : 2007년 5월 31일

DSMS에서는 조인이나 집계함수와 같은 관계 연산자(Relational Operator)를 이용하여 연속 스트림을 처리하기 위해 슬라이딩 윈도우(Sliding Window)를 이용한다[5,10]. 슬라이딩 윈도우 연산자는 무한한 스트림을 윈도우 익스텐트(Window Extent)라 불리는 유한한 부분 집합으로 분리하는 역할을 수행한다[10,11]. 윈도우 익스텐트를 생성할 때 기존의 윈도우 연산자는 입력 튜플이 순서화되어 전달되는 것을 가정하며, 비순서화된 튜플은 무시하는 방식을 취한다[8]. 그러나 DSMS에서 튜플은 일반적으로 네트워크를 통해 전달되는 경우가 많으므로 튜플의 전달순서가 지켜지지 않을 수 있다. 비순서화된 전달은 튜플의 손실을 유발하며, 집계 질의에 있어서 결과의 질적인 저하로 이어질 수 있다.

기존의 연구에서는 비순서화되어 전달되는 튜플을 정렬하기 위해 윈도우 연산자 앞에 버퍼를 둔다. 만약 네트워크 지연(Network Delay)의 최대 크기(Maximum Bound)가 미리 알려져 있다면, 이를 지원할 수 있는 고정 크기의 버퍼를 정의하여 이용할 수 있다[8,9]. 그렇지 않은 경우, 변화하는 네트워크의 지연에 맞추어 버퍼의 크기를 조정해야 한다. 그러나 변화하는 지연에 따라 버퍼의 크기를 조정하는 것은 쉬운 문제는 아니다. 기존의 연구에서는 현재까지 보여진 네트워크 지연의 최대값을 기반으로 버퍼의 크기를 추정한다[13,-16]. 그러나 이러한 방법은 버퍼의 크기를 불필요하게 크게 추정할 수 있으며, 튜플의 처리속도를 늦출 수 있다.

본 논문에서는 변화하는 네트워크의 지연에 따라 적응적으로 버퍼의 크기를 추정하기 위한 확률론적인 접근방법(Probabilistic Approach)을 제안한다. 제안하는 기법은 튜플의 생성 간격과 네트워크 지연과 같은 입력 스트림의 성질을 반영하는 추정식(Estimation Function)을 이용한다. 추정식은 튜플의 손실율(Drop Ratio)을 입력인자(Input Parameter)로 이용하며, 이는 실시간에 튜플의 손실에 있어서 허용 가능한 백분율을 의미한다. 질의 명세(Query Specification)에서 튜플의 손실율을 정의함으로써, 사용자는 응용의 요구에 따라 질의 결과의 정확성이나 처리속도 중 원하는 특성에 중점을 둘 수 있다.

그림 1은 추정식을 기반으로 비순서화된 튜플을 처리하기 위해 제안된 연산자인 비순서 제어기(Disorder Controller)의 구조를 보여준다. 비순서 제어기는 슬라이딩 윈도우 연산자 앞에 위치하며, 튜플의 타임스탬프를 기준으로 튜플을 정렬하여 윈도우 연산자로 전달한다. 비순서 제어기는 동적 버퍼(Dynamic Buffer)와 추정기(Estimator)로 구성되어 있으며, 각각 튜플을 순서대로 유지하고, 추정식을 이용하여 동적 버퍼의 크기를 추정하는 역할을 한다. 추정식을 유도하기 위해, 본 논문에서는 튜플의 생성이 포아송 분포를 따르며, 네트워크 지연은 정규 분포를 따른다고 가정한다. 이러한 가정은 네트워크 성질에 대한 일반적인 가정으로서 네트워크 분야에서 흔히 이용되고 있다.

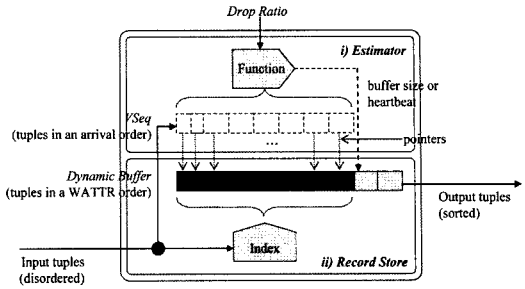

그림 1 비순서 제어기의 구조

본 논문의 구성은 다음과 같다. 먼저 2장에서는 비순서화된 튜플을 처리하기 위한 기존 연구와 문제점을 언급한다. 3장에서는 손실율을 정의하기 위한 질의 구문과 비순서 제어기의 구조를 제안한다. 4장에서는 추정식의 유도과정을 설명한다. 5장에서는 실험을 통해 제안하는 추정식과 기존의 추정 방식을 비교한다. 마지막으로 6장에서는 추후연구 방향을 제시하고 결론으로 마무리한다.

2. 기존 연구

비순서화되어 전달되는 튜플을 처리하기 위해 일반적으로 버퍼를 이용하여 튜플을 정렬한 후 윈도우 연산자로 전달하는 방법을 이용한다. 버퍼의 유지에 있어서 고정 크기의 버퍼를 이용하는 방법과 가변 크기의 버퍼를 이용하는 방법으로 구분할 수 있으며, 두 방식을 기반으로 한 기존의 접근 방법과 문제점을 언급한다.

2.1 고정 크기의 버퍼

Aurora[8]에서 윈도우 연산자는 비순서화된 튜플을 단순히 무시하는 방법을 취한다. 비순서화된 튜플의 손실을 막기 위해 슬랙(Slack)이라 불리는 버퍼를 이용하여 튜플을 정렬한 후 윈도우 연산자로 전달할 수 있도록 지원한다. Aurora는 네트워크 지연의 최대 크기가 미리 알려져 있다는 가정하에 이를 지원할 수 있는 충분히 큰 고정 크기의 슬랙 버퍼를 이용하도록 한다.

그림 2는 입력 시퀀스 (<2, 10>, <1, 20>, <4, 10>, <5, 15>, <3, 25>, <6,10>)이 주어졌을 때, 크기가 1인 슬랙 버퍼의 처리과정을 보여준다. 튜플은 원격 센서로부터 전달되었으며, 각 튜플은 자신이 생성된 시간을 알리는 타임스탬프(timestamp)와 센서 값을 포함하고 있다고 가정하였다. 그림 2에서 슬랙 버퍼는 튜플이 전달될 때마다 가장 적은 타임스탬프를 지니는 튜플을 출력한다. 그러나 <3, 25>와 같은 튜플은 버퍼를 통과한 후

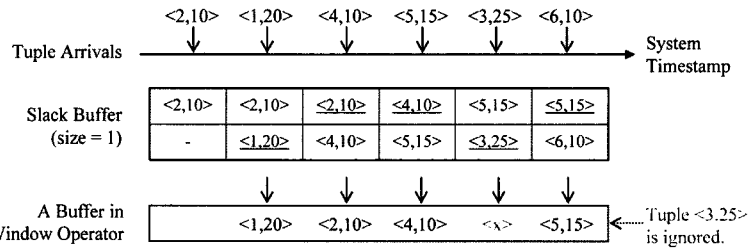

그림 2 슬랙 버퍼의 처리 과정(n=1)

에도 비순서화될 수 있다. 이와 같은 튜플은 윈도우 연산자에서 무시된다.

고정 크기의 버퍼를 이용하는 방법은 네트워크 지연에 대한 정보가 알려지지 않을 경우 이용에 제한이 있다. 만약 버퍼의 크기를 적게 유지하면 튜플의 손실이 생길 수 있으며, 반면에 버퍼의 크기를 크게 유지하면 튜플의 손실은 막을 수 있으나 튜플의 처리시간이 길어진다. 이와 함께, 주어진 버퍼의 크기로 얼마나 많은 튜플의 손실을 막을 수 있을지 역시 알아내기 힘들다. 예를 들어 위의 입력 시퀀스 (2, 1, 4, 5, 3, 6)은 그림 2에서 보이듯이 한 개의 튜플 손실을 유발하며, 손실 비율(Drop Ratio)은 20%(=1/5)가 된다. 그러나 같은 크기의 버퍼에 대해 (2, 4, 1, 5, 3, 6)과 같은 시퀀스는 2개의 튜플의 손실을 유발하며, 40%의 손실율을 보인다. 같은 방법으로 (4, 5, 1, 2, 3, 6)은 60%의 손실율을 보인다.

질의 명세에서 버퍼의 크기를 정의하는 기존의 방법에 비해, 제안하는 기법에서는 질의 명세에서 튜플의 손실율을 정의하도록 지원한다. 제안하는 기법에서 버퍼의 크기는 주어진 손실율을 만족시키기 위해 변화하는 입력 스트림의 성질에 따라 가변적으로 변한다. 사용자는 질의 명세에서 튜플의 손실율을 정의함으로써, 응용의 요구에 따라 질의 결과의 정확성이나 처리속도 중 원하는 특성에 중점을 둘 수 있다.

2.2 가변 크기의 버퍼

STREAM[6]에서 윈도우 연산자는 비순서 처리를 위한 메커니즘을 가지지 않는다. 대신 네트워크로부터 전달되는 튜플은 입력 관리자(Input Manager)[13,14]로 불리는 외부 모듈에서 정렬된 후 DSMS로 전달된다. 입력 관리자는 전달되는 튜플을 정렬하여 버퍼에 저장하며, 저장된 튜플로부터 윈도우를 생성하기 위한 시점을 결정하기 위해 현재까지 발생한 네트워크 지연의 최대값에 근거하여 구두점을 추정한다. 구두점(Punctuations)[17,18]이란 어느 시점에 시스템에 구두점 τ_p가 전달되었을 때, 해당 시점 이후로는 τ_p보다 적은 값을 지니는 튜플이 전달되지 않음을 의미한다. 따라서 구두점 τ_p가 전달되거나 식별되었을 때 윈도우 연산자에서는 τ_p보다 적은 값을 지니는 튜플을 모두 처리할 수 있다.

입력 관리자는 구두점의 추정에 있어서 네트워크에서 발생한 지연(Latency)의 최대값을 반영하는 방법[13, 14]을 이용한다. 이러한 방식에서는 튜플이 전달될 때마다 시스템에 도착한 시간과 튜플이 생성된 시간의 차이, 즉 튜플의 시스템 타임스탬프(System Timestamp)와 응용 타임스탬프(Application Timestamp)를 계산하며, 계산된 차이값 중 최대값을 항상 유지한다. 그리고 튜플이 전달될 때마다 튜플의 시스템 타임스탬프에서 계산된 최대값을 뺀 값을 구하여 구두점으로 활용한다. 이러한 방식은 현재까지 발생한 네트워크 지연의 최대값보다 더 큰 비순서(Disorder)가 발생하지 않을 것임을 가정하고 구두점을 추정하는 방식에 해당한다. 편의상 이와 같은 방식을 최대값-기반 추정 방식(Max-based Estimation)으로 명명한다.

그림 3은 스트림 시퀀스가 (3, 3, 4, 2, 5, 6)으로 주어졌을 때 최대값-기반 방식의 동작 상황을 도식화하고 있다. 그림 3에서 *maxDelay*는 네트워크 지연의 최대값을 의미하며, 주어진 시퀀스 이전의 스트림에서 발생한 지연의 최대값을 2로 가정하였다. maxDelay 값은 4번째 튜플 <2, 25>가 들어올 때 4로 바뀐다. 이는 4번째 튜플의 시스템 타임스탬프가 6이고 응용 타임스탬프는 2이므로 지연 값이 4가 되며, 기존의 최대값 2보다 크기 때문이다. 구두점은 단순히 전달된 튜플의 시스템 타임스탬프와 지연의 최대값의 차이를 통해 구할 수 있다.

위에서 설명한 내용에 대해 식으로 정리하면 식 (2.1)과 같다. 아래에서 r_i는 i번째 튜플을 의미하며, n은 가장 최근에 전달된 튜플의 순서를 의미한다. 그리고 $sysTS$는 시스템 타임스탬프를, 그리고 $appTS$는 응용 타임스탬프를 나타낸다. τ_p는 구두점을 의미한다.

$$maxDelay = max(r_i.sysTS - r_i.appTS) \quad (\text{where } 0 \le i \le n)$$

$$\tau_p = r_n.sysTS - maxDelay \qquad (2.1)$$

NiagaraCQ[15]와 Gigascope[16] 역시 이와 유사한 추정법을 이용한다. NiagaraCQ는 스트림 소스로부터 전달되는 구두점을 이용하거나, 네트워크 지연의 최대값이 미리 알려져 있다는 가정으로부터 구두점을 추정한다. Gigascope는 네트워크 지연의 최대값이 일정하다는

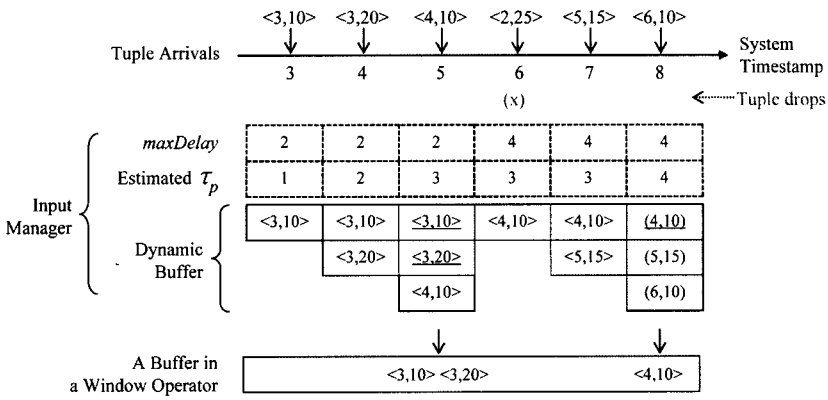

그림 3 최대값-기반 추정 방식의 처리 과정

가정(Banded-increasing property)으로부터 일정 시간 동안 데이타의 처리를 지연시키는 방법을 이용한다. 그러나 이러한 방법은 버퍼의 크기를 불필요하게 크게 추정할 수 있으며, 튜플의 처리속도를 늦출 수 있다.

3. 비순서 제어기

본 논문에서 제안하는 방식에서는 질의문에서 버퍼의 크기를 기술하기 보다는 사용자가 원하는 튜플의 손실율(Drop ratio)을 정의할 수 있도록 지원한다. 손실율이 주어지면 버퍼의 크기는 주어진 손실율을 만족시키기 위해 가변적으로 변하게 된다. 튜플의 손실율을 질의문에서 기술하기 위해 부가적인(Optional) 입력인자로서 *DRATIO*를 정의하였다. 사용자는 DRATIO의 정의를 통해 응용의 특성에 따라 튜플의 손실을 조절할 수 있다. 예를 들어 정확성이 요구되는 응용의 경우 DRATIO의 값을 적게 주며, 반대로 빠른 처리속도가 요구되는 응용의 경우 DRATIO의 값을 크게 정의할 수 있다.

Q1은 질의문에서 DRATIO를 정의한 예를 보여준다. DRATIO의 값은 백분율 단위로 정의된다. Q1에서는 CQL[12]과 유사한 질의 구문을 이용하였으며, 윈도우의 정의를 위해 [10]에서 제안한 구문을 이용하였다. [10]에서 윈도우의 정의 구문은 윈도우 익스텐트의 크기를 나타내는 *RANGE*, 해당 익스텐트의 슬라이드 주기를 나타내는 *SLIDE*, 그리고 RANGE와 SLIDE의 기준이 되는 윈도우 애트리뷰트(Windowing attribute)를 나타내는 WATTR 등으로 구성된다. 현재 제안하는 방식에서는 윈도우 애트리뷰트가 타임스탬프인 슬라이딩 윈도우를 처리하는 것을 가정하고 있다.

```
Q1: SELECT max(temp)
    FROM Sensors [RANGE 5 minutes
                  SLIDE 1 minute
                  WATTR timestamp]
    DRATIO 1%]
```

질의 계획을 생성할 때 윈도우 명세는 윈도우 연산자와 비순서 제어기로 변환된다. 비순서 제어기(Disorder Controller)는 슬라이딩 윈도우 연산자 앞에 위치하며, 윈도우 애트리뷰트를 기준으로 튜플을 정렬하여 윈도우 연산자로 전달한다. 비순서 제어기는 그림 1에서 보인 바와 같이 동적 버퍼(Dynamic Buffer)와 추정기(Estimator)로 구성되어 있다. 동적 버퍼는 윈도우 애트리뷰트의 오름차 순으로 튜플을 유지한다. 입력 튜플을 버퍼에 저장할 때, 정렬에 드는 비용을 줄이기 위해 [19]에서 제안한 인덱스 기법을 이용할 수 있다. 그리고 추정기는 추정식을 이용하여 동적 버퍼의 크기를 결정하는 역할을 담당한다. 추정식의 입력인자에 해당하는 손실율은 질의 명세로부터 주어지며, 이를 통해 사용자가 튜플의 손실을 조정할 수 있도록 지원한다. 제안하는 추정식은 버퍼의 크기 이외에 구두점의 생성 역시 지원한다.

추정기에서 추정식은 내부적으로 튜플의 생성 간격의 평균 δ와 네트워크 지연의 표준 편차 σ 등을 추정에 이용한다. 이들 인자는 입력 튜플을 모니터링하여 유도될 수 있으며, 입력 튜플을 유지하기 위해 *VSeq*라는 버퍼를 이용한다. VSeq는 전달된 순서를 기반으로 입력 튜플을 유지한다. 따라서 윈도우 애트리뷰트의 값을 기반으로 하는 동적 버퍼와는 구분된다. 제안하는 기법에서는 VSeq의 구현에 있어서 VSeq에서 별도로 튜플의 사본을 유지하기 보다는 동적 버퍼 내의 튜플에 대한 포인터를 유지한다.

4. 추정식의 유도

이 장에서는 변화하는 입력 스트림의 성질에 따라 적응적으로 버퍼의 크기를 추정하기 위한 추정식의 유도

과정에 대해 설명한다. 제안하는 추정식은 버퍼의 크기 뿐 아니라 구두점을 생성할 수도 있다. 입력 스트림의 성질과 관련된 가정(Assumption)에 대해 소개한 후 이를 바탕으로 추정식을 유도하는 과정에 대해 설명한다.

4.1 입력 스트림에 대한 가정

제안하는 기법에서는 입력 스트림의 성질과 관련하여 두 가지 사항을 가정한다. 먼저 스트림 소스에서 튜플 생성은 포아송 분포(Poisson Distribution)를 따른다고 가정한다. 이는 튜플 생성의 주기(Interval)가 지수 분포(Exponential Distribution)을 따른다는 것과 동일하다. 그리고 네트워크의 지연은 정규 분포(Normal Distribution)을 따른다고 가정한다. 아래에서 θ, μ 및 o는 각각 튜플의 생성 간격의 평균, 네트워크 지연의 평균 및 표준 편차를 의미한다.

$$(T_i - T_{i-1}) \sim Exp(\theta) \quad (1)$$

$$(T_i - T_i) \sim N(\mu, o) \quad (2)$$

편의상 T_i는 i-번째 튜플의 응용 타임스탬프, 즉 스트림 소스(Stream Source)에 의해 할당되어 튜플에 포함된 타임스탬프를 의미하며, T_i는 i-번째 튜플의 시스템 타임스탬프, 즉 시스템에 전달되었을 때 시스템에 의해 부여된 타임스탬프를 의미한다.

θ, μ 및 o와 같은 인자는 VSeq의 튜플로부터 유도될 수 있다. VSeq는 해당 인자 값의 적절한 산정을 위해 최소 30개 이상의 값을 유지한다. 먼저 θ는 (3)에 의해 유도될 수 있다. (3)에서 n은 VSeq의 크기를 나타낸다.

$$\theta = (T_n - T_1) / n \quad (3)$$

다음으로 μ와 o 역시 VSeq의 튜플로부터 유도될 수 있으며, 제안하는 기법에서는 효율을 위해 점증적인(Incremental) 방법을 이용한다. 예를 들어 비순서 제어기는 VSeq의 모든 튜플의 네트워크 지연의 합(Sum)을 유지한다. 새로운 튜플이 들어올 때마다 네트워크 지연 값을 구하며, 가장 오래된 튜플의 지연 값을 빼고, 새로운 튜플의 지연 값을 더해서 합을 수정한다. 그리고 n으로 나누어 μ를 구한다. o 역시 이와 같은 점증적인 방법으로 구해질 수 있다.

4.2 추정식의 유도

튜플의 손실은 최근 산정된 구두점보다 적은 타임스탬프를 지닌 튜플이 들어올 경우 발생한다. n번째 튜플이 전달되었을 때 산정된 구두점을 τ_p라 하고, 산정 이후 전달될 튜플 r_{n+1}의 타임스탬프를 τ_{n+1}이라고 하면, r_{n+1}이 손실될 확률은 $\Pr(\tau_{n+1} < \tau_p)$로 나타낼 수 있다. 이는 질의문에서 주어진 손실율 R을 넘지 않아야 하며 아래와 같이 표현될 수 있다.

$$\Pr(\tau_{n+1} < \tau_p) \le R \quad (4)$$

위 식으로부터 버퍼의 크기를 산정하기 위한 식을 유도하기 이전에, 우선 VSeq내의 T_1과 T_n 간의 간격이 어떤 분포를 가지는지 소개한다.

$$(t_i - t_{i-1}) \sim N((n-1)\cdot\theta, (n-1)\cdot\theta^2) \quad (5)$$

(5)의 유도과정

VSeq의 튜플 간의 간격 (t_i - t_{i-1})를 V_i라고 하면, 지난 절에서 가정한 사항으로부터 $V_i \sim \mathrm{Exp}(\Theta)$라 할 수 있으며, V_i의 평균과 분산은 각각 θ와 θ^2이 된다. 그리고 V_i의 개수는 (n - 1)이며 VSeq가 30개 이상의 값을 유지하므로, V_i의 합 (t_n - t_1)은 중심극한정리(Central Limit Theorem)[20]로부터 정규분포로 근사화할 수 있다. 이 때 (t_n - t_1)의 평균과 분산의 값은 V_i의 평균과 분산 θ와 θ^2에 대해 각각 (n - 1)을 곱한 값이 된다. 이는 V_i가 독립적인 사건이라는 사실에 기인한다. □

앞에서 유도된 (5)와 지난 절에서 소개한 가정을 바탕으로, 산정 이후 전달될 튜플 r_{n+1}이 손실될 확률 $\Pr(\tau_{n+1} < \tau_p)$를 구할 수 있다. 편의를 위해 τ_p 대신 τ_1으로 대체하여 $\Pr(\tau_{n+1} < \tau_1)$으로 두고 식을 유도한다. τ_1은 항상 τ_p보다 크므로, 수정된 식 $\Pr(\tau_{n+1} < \tau_1)$은 원래의 식에 비해 항상 큰 손실율을 보이게 된다. 따라서 τ_p 대신 τ_1으로 대체해도 무방하다.

$$\Pr(\tau_{n+1} < \tau_1) = Z\left(\frac{n\theta}{\sqrt{2\sigma^2 + n\theta^2}}\right) \quad (6)$$

(6)의 유도과정

위에서 왼쪽 항은 아래와 같이 변환될 수 있다.

$\Pr(\tau_{n+1} < \tau_1) = \Pr(\tau_{n+1} - \tau_1 < 0)$

이 때 $\Pr(\tau_{n+1} - \tau_1 < 0)$은 그림 4와 같이 분해될 수 있다. 아래 식에서 시스템 타임스탬프 t_i는 응용 타임스탬프 τ_i와 대응된다.

$(\tau_{n+1} - \tau_1) = (\tau_{n+1} - t_{n+1}) + (t_{n+1} - t_1) + (t_1 - \tau_1)$

$(\tau_{n+1} - t_{n+1})$와 $(t_1 - \tau_1)$은 (2)로부터 정규분포를 따른다.

$$(\tau_{n+1} - t_{n+1}) \sim N(-\mu, o) \quad (6.1)$$

$$(t_1 - \tau_1) \sim N(\mu, o) \quad (6.2)$$

(5)로부터 $(t_{n+1} - t_1)$은 정규분포를 따른다.

$$(t_{n+1} - t_1) \sim \mathrm{N}(n\cdot\theta, n\cdot\theta^2) \quad (6.3)$$

따라서 (6.1)에서 (6.3)까지의 분포와 정규분포의 적률생성함수(Moment Generating Function)[20]로부터 $(\tau_{n+1} - \tau_1)$는 다음과 같이 변환될 수 있다. 편의상 아래에서 $(\tau_{n+1} - \tau_1)$을 x, 그리고 $(\tau_{n+1} - t_{n+1})$와 $(t_{n+1} - t_1)$, $(t_1 - \tau_1)$를 각각 y_1, y_2, y_3로 표기하였다.

$$\begin{aligned} Mx(s) &= My_1(s)\cdot My_2(s)\cdot My_3(s) \\ &= e^{(\delta^2 s^2/2)-\mu s}\cdot e^{(n\theta^2 s^2/2)+ns\delta}\cdot e^{(\delta^2 s^2/2)+\mu s} \\ &= e^{\{(\delta^2 s^2/2)-\mu s\}+\{(n\theta^2 s^2/2)+ns\delta\}+\{(\delta^2 s^2/2)+\mu s\}} \\ &= e^{\{(2\delta^2+n\theta^2)s^2/2+ns\delta\}} \end{aligned}$$

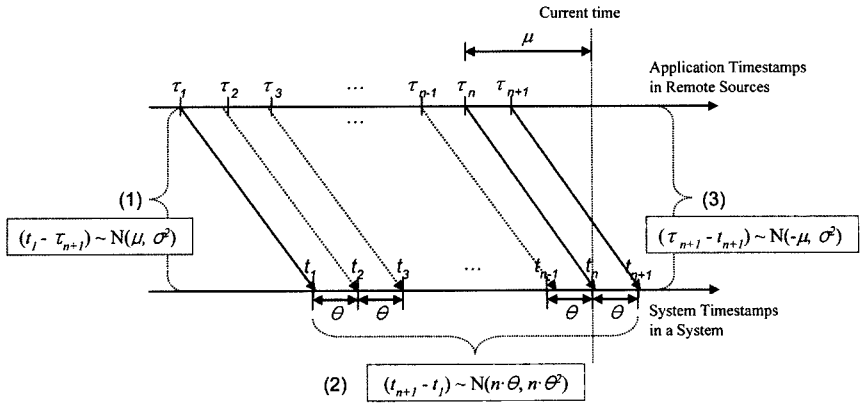

그림 4 (T_{n+1} - T_1)의 분해

결과는 다시 정규분포의 적률생성함수의 형태이다. 따라서 (τ_{n+1} - τ_1)은 정규분포를 따른다.

$$(\tau_{n+1} - \tau_1) \sim N(n\cdot\theta, \sqrt{2\sigma^2 + n\cdot\theta^2})$$

마지막으로 위 식에 대한 정규화(Normalization)를 통해, 다음에 전달되는 튜플의 손실 확률에 대해 아래 식으로 정리할 수 있다. 아래 식에서 z는 (τ_{n+1} - τ_1)에 대한 표준화된(Normalized) 변수에 해당한다.

$$\Pr(\tau_{n+1} < \tau_1) = \Pr\left(z < \frac{n\theta}{\sqrt{2\sigma^2 + n\theta^2}}\right) \quad \square$$

위 결과로부터 (5)를 아래와 같이 다시 쓸 수 있다.

$$\Pr\left(z < \frac{n\theta}{\sqrt{2\sigma^2 + n\theta^2}}\right) \le R \qquad (5')$$

위 식으로부터 질의문에서 주어진 R을 만족하는 버퍼의 크기 (일반적인 n과 구분하여) N을 구할 수 있다. 이는 비순서 제어기에서 VSeq의 크기인 동시에 동적 버퍼의 크기와 같다. 위 식을 등식으로 두고 풀면 아래의 (7)과 같은 결과를 얻을 수 있다. 아래에서 Z는 주어진 R에 대응되는 z값을 반환하는 함수이며, C는 주어진 R에 대한 z값을 제곱한 값, 즉 $\{Z(R)\}^2$을 의미한다. 그리고 *floor*는 주어진 입력값보다 적거나 같은 정수 중 가장 큰 수를 반환하는 함수를 나타낸다.

$$N = floor\left(\frac{C + \sqrt{C^2 + \frac{8\cdot C\cdot\sigma^2}{\theta^2}}}{2}\right) \qquad (7)$$

(where $C = \{Z(R)\}^2$)

(7)의 유도과정

식의 유도를 위해 (5')를 아래와 같이 Z를 이용하여 다시 쓸 수 있다.

$$\frac{n\theta}{\sqrt{2\sigma^2 + n\theta^2}} \le Z(R)$$

위 식을 등식으로 두고 풀면 다음과 같은 식을 얻을 수 있다.

$$n^2 - C\cdot n - 2\cdot C\cdot\sigma^2 / \theta^2 = 0$$

근의 공식을 이용하여 위 식을 만족하는 n값, 즉 N을 아래와 같이 구할 수 있다.

$$N = \frac{C \pm \sqrt{C^2 + \frac{8\cdot C\cdot\sigma^2}{\theta^2}}}{2}$$

위 값에서 -를 취할 경우는 항상 음수를 얻게 된다. 이에 반해 버퍼의 크기는 양의 정수여야 하므로, 위 값에서 +를 취한 후 *floor*함수를 적용하여 N을 얻을 수 있다. $\square$

N이 주어졌을 때, (3)의 응용 타임스탬프와 시스템 타임스탬프 간의 변환 방식으로부터 구두점 τ_p를 구할 수 있다. (8)은 N이 주어졌을 때 구두점 τ_p를 구하기 위한 식에 해당한다. 아래에서 t_n은 현재 시스템 타임스탬프를 의미한다.

$$\tau_p = \tau_n - N\cdot\theta \qquad (8)$$

(8)의 유도과정

R을 만족하는 버퍼의 크기 N이 주어졌을 때, 그에 해당하는 구간은 $N\cdot\theta$의 형태로 계산될 수 있다. 그리고 (3)으로부터 VSeq의 응용 타임스탬프의 최대값은 τ_n (= t_n - μ)이다. 따라서 τ_n에서 $N\cdot\theta$을 빼면 구두점 τ_p를 구할 수 있다. $\square$

5. 실험 결과

이 장에서는 실험을 통해 적응성(Adaptivity)의 측면에서 본 논문에서 제시한 추정식과 기존의 최대값-기반(Max-based) 추정식을 비교하였다. 적응성이란 입력 스트림의 성질에 나타나는 변화를 추정에 얼마나 잘 반영하는지를 의미한다. 제시한 방식의 적응성을 보이기 위해 세 가지 실험을 수행하였다. 첫번째 실험은 제시한 추정식이 주어진 손실율을 만족하는지 확인하였다. 두번째 실험은 두 방식으로부터 추정된 버퍼의 크기를 비교

하였다. 마지막 실험은 안정성의 관점에서 예외적인 크기의 비순서가 발생할 경우 두 방식의 추정 결과가 어떤 영향을 받는지 비교하였다.

5.1 데이타 생성

실험을 위해 데이타 생성기(Data Generator)를 구현하였다. 구현된 데이타 생성기는 스트림 소스에서 데이타의 생성이 임의로(Randomly) 이루어지며, 네트워크의 지연은 주어진 바운드 내에서 정규 분포를 따르도록 구성하였다. 현재 주로 이용되고 있는 TinyDB[21]나 SENSIM[22] 등을 이용해 데이타를 생성해 보았으나, 실험에 적용하기에는 한계가 있었다. 예를 들어 TinyDB의 경우 네트워크 지연의 최대 크기가 꾸준히 증가하는 경향을 보였으며, 네트워크에 대한 구체적인 특성(예를 들어 네트워크 지연의 최대 크기나 표준 편차 등)을 설정할 수 없었다. 그리고 SENSIM이나 NS2 센서 네트워크 확장 버전[23]은 비순서화된 튜플이 발생하지 않았다. 이는 두 시뮬레이터에서 MAC 계층으로 802.11을 이용함으로써, 전송 속도가 센서의 액티베이션 속도보다 빠르다는 점에서 기인하였다.

본 실험은 윈도우 XP을 탑재한 인텔 펜티엄 4 2.4 MHz 머신에서 수행되었다. 실험에 이용된 각 데이타 세트는 17M 정도의 크기를 지니며, 약 100만개 정도의 튜플로 구성되었다.

5.2 실험 결과

제안한 추정식의 정확성: 첫번째 실험은 제안한 추정식이 질의문에서 주어진 손실율을 만족하는지 확인하고자 하였다. 실험을 위해 R을 1%에서 5%까지 변화시키고, 네트워크 지연의 최대 바운드 B를 4에서 20까지 주었다. 그리고 네트워크 지연의 표준 편차 σ를 1에서 6까지 변화시켜 보았다. 그림 5에서 보여진 실험 결과는 네트워크 지연이 강한 정규분포를 가질 경우 (예를 들어 네트워크 지연의 표준편차가 2 이하인 경우) 주어진 손실율을 만족하는 것으로 나타났으며, 이와는 반대로 네트워크 지연이 약한 정규분포를 가질 경우 손실율을 위반하는 결과가 발생하였다. 손실율을 위반하는 결과는 그림 5에서 점선으로 표시하였다.

위 실험에서 위반 결과는 B가 14 이상이며, σ가 3에서 6 사이일 경우만 발생하였다. σ가 6 이상인 경우는 오히려 손실율을 만족하는 결과가 나오나, 버퍼의 크기가 지나치게 커졌다. 그리고 R이 1%로 주어질 경우 위반하는 결과가 없었다. 만약 사용자가 결과의 정확성에 중점을 두는 경우 R을 1% 이하로 줄 확률이 높다. 따라서 실험 결과는 제안하는 방식이 정확성에 중점을 두는 응용에서 적절히 이용될 수 있음을 보인다.

버퍼 크기의 비교: 두번째 실험에서는 네트워크의 어떤 특성이 두 추정식에 영향을 주는지 비교하고자 하였다. 먼저 네트워크 지연의 표준편차 σ를 1로 고정시키고 최대 크기 B를 3에서 20까지 변화시킨 다음 두 추정식에서 보인 버퍼의 크기를 비교하였다. 그림 6(a)에서 보인 결과는 최대값-기반 방식의 결과가 변화하는 B값에 따라 꾸준히 증가하는 반면, 제안한 방식의 결과는 3600 근처에서 거의 변화가 없는 것으로 나타났다. 이는 제안한 방식이 추정에 있어서 B를 반영하지 않고 σ만을 반영함으로써 나타나는 결과로 볼 수 있다. σ를 1, G를 1/1,000,000, 그리고 B을 1%로 주었을 때 (7)로부터 버퍼의 크기에 있어서 3,740이라는 이론적인 상한을 얻을 수 있다. 그림 6(a)의 결과는 실험 결과가 이러한 상한에 근접함을 나타낸다.

다음으로 네트워크 지연의 최대 크기 B를 10으로 고정시키고 표준편차 σ를 0.8에서 3까지 변화시킨 다음 두 추정식에서 보인 버퍼의 크기를 비교하였다. 그림 6(b)에서 보인 결과는 그림 6(b)와는 반대로 최대값-기반 방식의 결과는 거의 변화가 없는 반면, 제안한 방식의 결과는 변화하는 값에 따라 꾸준히 증가하였다. σ가 2.5보다 적을 때는 제안한 방식에 의해 추정된 버퍼의 크기가 적으나, 2.5보다 커질 때는 최대값-기반 방식이 더욱 우수하였다.

안정성 비교: 안정성의 관점에서 우선 두 추정식으로

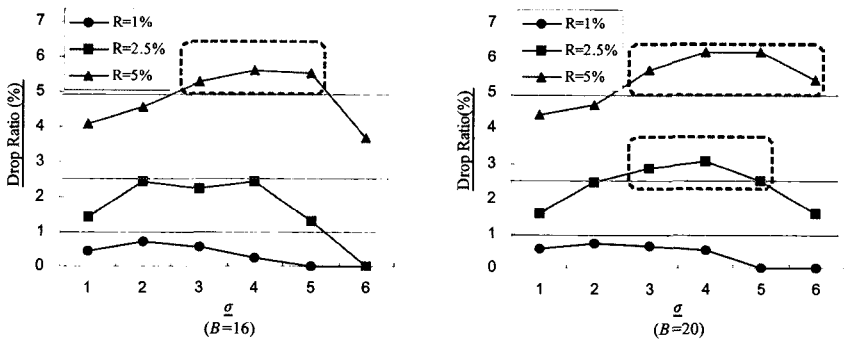

그림 5 제안한 추정 방식의 손실율

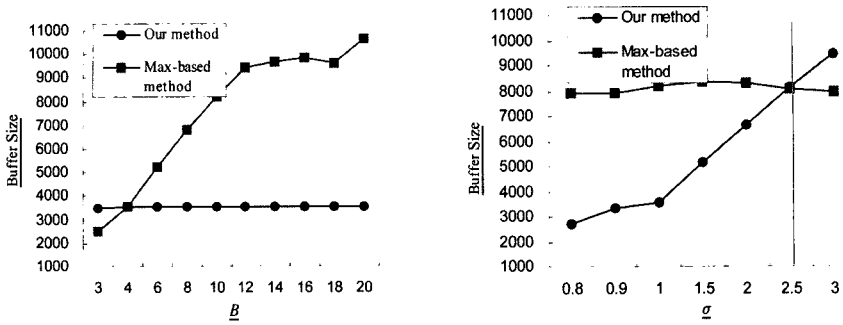

그림 6 두 추정식의 버퍼 크기의 비교

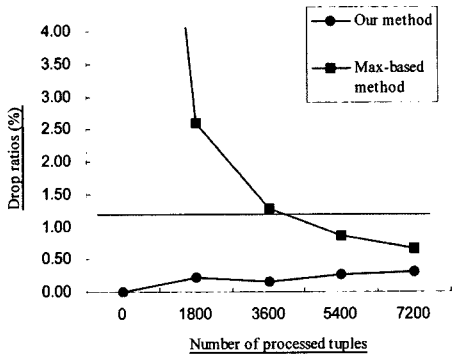

그림 7 두 추정식의 튜플 손실 과정 비교

부터 발생한 튜플 손실의 과정을 비교하였다. 제안한 추정식에 대해 B를 1%로 정의한 후 실험하였다. 그림 7의 결과에서 볼 수 있듯이 최대값-기반 추정식의 경우 초반에 튜플의 손실이 많았으며, 반면 제안한 추정식에서는 튜플의 손실이 고른 분포를 나타내었다. 최대값-기반 방식에서 초반에 튜플의 손실이 많은 이유는 추정 초반에 네트워크 지연의 최대 크기를 알아내기 위한 조정 과정이 필요하기 때문이다.

다음으로 데이타에 예외적인 크기의 비순서를 지닌 튜플을 위치시킨 후, 비순서의 크기를 증가시킬 때 버퍼의 크기나 평균 대기시간에 어떠한 결과를 미치는지 알아보고자 하였다. 그림 8은 임의의 데이타 세트에서 초반에 하나의 튜플에 대해 비순서의 크기를 증가시켜본 결과를 보여주고 있다. 실험 결과 최대값-기반 추정식의 경우 주어진 비순서 크기에 민감한 반응을 보이며, 제안한 추정식에서는 이와는 반대로 거의 영향을 받지 않음을 알 수 있었다.

7. 결 론

본 논문에서는 변화하는 네트워크의 지연에 따라 적응적으로 버퍼의 크기를 추정하기 위한 확률론적인 접근방법을 소개하였다. 제안하는 기법은 튜플의 생성 간격과 네트워크 지연과 같은 입력 스트림의 성질을 반영한 추정식(Estimation Function)을 이용하였다. 추정식은 질의문에서 정의된 튜플의 손실율(Drop Ratio)을 기준으로 버퍼의 크기를 추정하며, 사용자는 튜플의 손실율을 정의함으로써 응용의 요구에 따라 질의 결과의 정확성이나 처리속도 중 원하는 특성에 중점을 둘 수 있도록 지원하였다. 제안한 추정식에 대해서는 실험을 통해 적응성의 관점에서 기존의 최대값-기반 방식에 비해 우수함을 보였다. 추후에는 비동기화된 스트림을 병합(Merging)하거나 연산자의 처리결과 등에서 생길 수 있는 보다 다양한 형태의 비순서를 처리하기 위한 방법에 대해 연구를 진행할 예정이다.

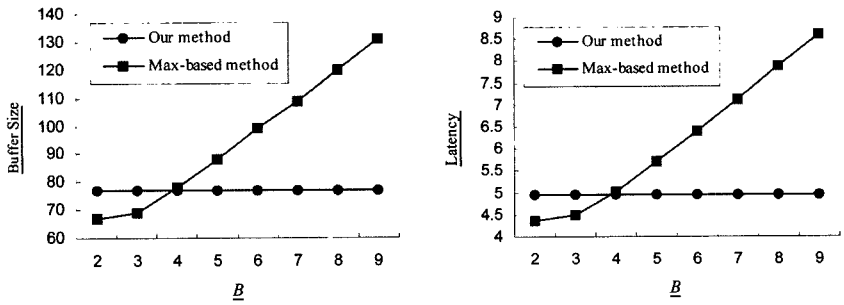

그림 8 비순서 크기에 따른 버퍼 크기 및 평균 대기시간의 증가

참 고 문 헌

[1] Douglas Terry, David Goldberg, David Nichols, and Brian Oki, *Continuous Queries over Append-Only Databases*. ACM SIGMOD, 1992.

[2] Samuel R. Madden, Mehul A. Shah, Joseph M. Hellerstein and Vijayshankar Raman, *Continuously Adaptive Continuous Queries over Streams*. ACM SIGMOD Conference, Madison, WI, June 2002.

[3] S. Babu and J. Widom, *Continuous Queries over Data Streams*. ACM SIGMOD Record, Sep. 2001.

[4] Rajeev Motwani et al, *Query Proessing, Resource Management, and Approximation in a Data Stream Management System*. CIDR 2003, Jan. 2003.

[5] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom, *Models and Issues in Data Stream Systems*. Invited paper in Proc. of the 2002 ACM Symp. on Principles of Database Systems (PODS 2002), June 2002.

[6] Arvind Arasu et al, *STREAM: The Stanford Data Stream Management System*. IEEE Data Engineering Bulletin, Vol. 26 No. 1, March 2003.

[7] Sirish Chandrasekaran et al, *TelegraphCQ: Continuous Dataflow Processing for an Uncertain World*. CIDR 2003.

[8] D. Abadi, D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, S. Zdonik. *Aurora: A New Model and Architecture for Data Stream Management*. VLDB Journal (12)2: 120-139, August 2003.

[9] D. Abadi at al, *The Design of the Borealis Stream Processing Engine*. CIDR 2005, Asilomar, CA, January 2005.

[10] Jin Li, David Maier, Kristin Tufte, Vassilis Papadimos, Peter A. Tucker, *Semantics and Evaluation Techniques for Window Aggregates in Data Streams*. ACM SIGMOD 2005, June 14-16, 2005, Baltimore, Maryland, USA.

[11] Jin Li, David Maier, Kristin Tufte, Vassilis Papadimos, Peter A. Tucker, *No Pane, No Gain: Efficient Evaluation of Sliding Window Aggregates over Data Streams*. SIGMOD Record, Vol 34, No. 1, March 2005.

[12] A. Arasu, S. Babu and J. Widom, *The CQL Continuous Query Language: Semantic Foundations and Query Execution*. Stanford University Technical Report, Oct. 2003.

[13] U. Srivastava and J. Widom. *Flexible Time Management in Data Stream Systems*. ACM PODS 2004, June 2004.

[14] S. Babu, U. Srivastava and J. Widom, *Exploiting k-Constraints to Reduce Memory Overhead in Continuous Queries over Data Streams*. ACM TODS, Sep. 2004.

[15] J. Chen, D. J. DeWitt, F. Tian, and Y. Wang. *NiagaraCQ: A scalable continuous query system for internet databases*. ACM SIGMOD pages 379-390, May 2000.

[16] Chuck Cranor, Theodore Johnson, Oliver Spataschek and Vladislav Shkapenyuk, *Gigascope: A Stream Database for Network Applications*. ACM SIGMOD, June 9-12 2003.

[17] Peter A. Tucker, David Maier, Time Sheard, Leonidas Fegaras, *Exploiting Punctuation Semantics in Continuous Data Streams*. IEEE Transactions on Knowledge and Data Engineering, May/June 2003.

[18] David Maier, Jin Li, Peter A. Tucker, Kristin Tufte and Vassilis Papadimos, *Semantics of Data Streams and Operators*. ICDT 2005, LNCS 3363, pp.37-52, 2005.

[19] Lukasz Golab, Shaveen Garg, and M.Tamer Ozsu, *On Indexing Sliding Windows over Online Data Streams*, EDBT 2004, LNCS 2992, pp.712-729, 2004.

[20] Dimitry P. Bertsekas and John N. Tsitsiklis, *Introduction to Probability: International Edition*, Athena Scientific, Belmont, Massachusetts, 2002.

[21] TinyDB: http://www.tinyos.net.

[22] SENSIM: http://csc.lsu.edu/sensor_web/simulator.html.

[23] NS2 Sensor Network Extension: http://pf.itd.nrl.navy.mil/nrlsensorsim.

김 현 규

1997년 울산대학교 전산학과 학사. 2000년 울산대학교 전산학과 석사. 2000년~2001년 한국국방연구원 연구원. 2001년~2004년 LG전자 단말연구소 선임연구원. 2005년~현재 한국과학기술원 전산학과 박사과정. 관심분야는 데이타베이스 시스템, 스트림 데이타 처리 등

김 철 기

1996년 한국과학기술원 전산학과 학사 1998년 한국과학기술원 전산학과 석사 2005년 한국과학기술원 전자전산학과 박사. 2001년~2004년 LG전자 단말연구소 선임연구원. 2004년~현재 한국 정보통신대학교 연구원, 연구교수. 관심분야는 이동적응망, 무선인지기술, 센서네트워크 등

이 충 호

1997년 인하대학교 전자계산공학과(공학사). 1999년 인하대학교 전자계산공학과 전산학전공(공학석사). 2003년 인하대학교 전자계산공학과 전산학전공(공학박사). 2003년~2004년 인하대학교 연구교수. 2004년~현재 한국전자통신연구원 텔레매틱스 USN연구단 선임연구원. 관심분야는 시공간데이타베이스, 스트림 질의처리, GIS 등

김 명 호

1982년 서울대학교 컴퓨터공학과 학사 1984년 서울대학교 컴퓨터공학과 석사 1989년 미국 Michigan State University 전산학과 박사. 현재, 한국과학기술원 전산학전공 교수. 관심분야는 데이타베이스 시스템, 분산시스템, XML, Sensor Networks 등