다중 연속질의에서 슬라이딩 윈도우 집계질의 최적화를 위한 선형 자원공유 기법

(Linear Resource Sharing Method for Query Optimization of Sliding Window Aggregates in Multiple Continuous Queries)

백 성 하 † 유 병 섭 † 조 숙 경 †† 배 해 영 †††
(Seong-Ha Baek) (Byeong-Seob You) (Sook-Kyoung Cho) (Hae-Young Bae)

요 약 스트림 처리기는 다수의 연속질의에서 제한된 자원을 효율적으로 이용하기 위하여 자원공유 기법을 이용한다. 기존의 기법은 계층구조를 유지하여 집계질의를 처리한다. 그래서 삽입연산은 계층구조 재구성 비용이 필요하다. 또한 검색연산은 서로 다른 슬라이딩 윈도우 크기에 속하는 집계정보 검색비용이 필요하다. 그래서 본 논문에서는 보다 빠른 질의 처리를 위해 선형 자료구조를 사용한다. 제안기법은 팬(Pane)크기 결정단계와 팬 생성단계, 팬 삭제단계로 구성된다. 팬 크기 결정단계는 정확한 집계정보를 유지하기 위한 최적 팬 크기를 결정하는 단계이며, 팬 생성단계는 스트림 버퍼로부터 팬 크기만큼의 데이타에 대한 집계정보를 저장하는 단계이다. 팬 삭제단계는 더 이상 연속질의가 사용하지 않는 팬을 삭제하는 단계이다. 제안 기법은 선형 자료 구조를 이용하므로 계층구조를 이용하는 자료 구조에 비해 자원을 적게 사용한다. 또한 스트림 데이타가 입력되어도 팬 크기에 해당하는 집계정보만 계산하면 되므로 집계정보 삽입비용이 감소하고, 서로 다른 슬라이딩 윈도우 크기에 대해서도 선형검색으로 집계정보 검색비용이 감소한다. 성능평가를 통하여 제안기법이 적은 메모리 사용 결과를 보였으며, 질의 처리 속도가 증가하였다.

키워드 : 자원공유, 스트림, 슬라이딩 윈도우, 집계, 질의 최적화

Abstract A stream processor uses resource sharing method for efficient of limited resource in multiple continuous queries. The previous methods process aggregate queries to consist the level structure. So insert operation needs to reconstruct cost of the level structure. Also a search operation needs to search cost of aggregation information in each size of sliding windows. Therefore this paper uses linear structure for optimization of sliding window aggregations. The method comprises of making decision, generation and deletion of panes in sequence. The decision phase determines optimum pane size for holding accurate aggregate information. The generation phase stores aggregate information of data per pane from stream buffer. At the deletion phase, panes are deleted that are no longer used. The proposed method uses resources less than the method where level structures were used as data structures as it uses linear data format. The input cost of aggregate information is saved by calculating only pane size of data though numerous stream data is arrived, and the search cost of aggregate information is also saved by linear searching though those sliding window size is different each other. In experiment, the proposed method has low usage of memory and the speed of query processing is increased.

Key words : resource sharing, stream, sliding window, aggregate, query optimization

· 본 연구는 정보통신부 및 정보통신연구진흥원의 대학 IT연구센터 육성 지원사업의 연구결과로 수행되었음

† 학생회원 : 인하대학교 컴퓨터정보공학과
bshzeratul@dblab.inha.ac.kr
subi@dblab.inha.ac.kr
†† 정 회 원 : 인하대학교 컴퓨터정보공학과
skcho@dblab.inha.ac.kr
††† 종신회원 : 인하대학교 컴퓨터정보공학과 교수
hybae@inha.ac.kr

논문접수 : 2006년 8월 3일
심사완료 : 2006년 10월 30일

1. 서 론

최근 센서 네트워크 및 통신기술의 발달에 힘입어 스트림 데이타(Stream Data)[1]에 대한 연구가 활발히 진행되고 있다. 스트림 데이타에 대한 연구 결과로 DSMS (Data Stream Management System)[2,3]가 등장하였는데, 이는 새로운 형태의 질의인 연속질의(Continuous Query Language)[1,4]를 사용한다. 연속질의는 시스템

에 등록이 되고 조건에 맞는 데이타만 처리하는 점에서 기존의 질의와 차이가 있다. 모든 연속질의는 슬라이딩 윈도우를 포함하고 있고, 집계정보 처리를 요구한다. 그리고 이 연속질의는 슬라이딩 윈도우의 특성에 따라 두 가지로 구분된다. 첫 번째는 시간단위의 슬라이딩 윈도우를 포함하는 시간기반 질의이다. 다음은 튜플단위의 슬라이딩 윈도우를 포함하는 튜플기반 질의가 있다. 특히 이 두 종류의 연속질의 중 하나 이상이 동일한 자원에서 실행되는 경우가 발생할 수 있다.

주식 거래 모니터링 프로그램의 예를 고려해 보자. 이 시스템은 주식거래정보 데이타를 꾸준히 입력 받고 사용자가 등록한 연속집계질의를 처리한다. 먼저 초당 A 주식의 거래량 변화를 알기 위해 "A 주식의 초당 거래량의 평균"을 요청하는 시간기반 질의를 등록할 수 있다. 그리고 A주식의 최근 100개의 거래 중 가장 큰 가격을 알기 위해 "A 주식의 100개의 거래당 최대가격"을 요청하는 튜플기반 질의를 등록할 수 있다. 이와 같이 하나의 자원인 A 주식의 거래 정보에 두 가지 이상의 질의가 등록 될 수 있다. 그래서 동일 자원에 등록된 여러 개의 집계 질의를 효율적으로 처리할 수 있고 시간기반 질의와 튜플기반 질의를 동시에 처리 할 수 있는 시스템이 요구된다.

스트림 환경의 질의처리 최적화는 질의 처리 최적화도 중요하지만, 자원의 효율적인 사용이 더욱 중요하다. 그래서 동일한 자원에서 실행중인 여러 개의 집계 질의는 자원을 공유해야 한다. 그러므로 예에서 제시한 환경에서는 자원 공유를 이용해 질의 처리를 하는 시스템이 필요하다. 기존의 연구 중에 자원공유를 이용해 질의를 처리하는 기법이 몇 가지 있었다.

우선 BINT[7]가 다중 연속질의에서 집계정보 처리의 최적화를 위하여 연구되었다. BINT는 집계 값을 계층구조로 분할된 버퍼에 저장한다. 분할된 버퍼를 범위(Interval)이라고 부른다. BINT는 질의 요청이 왔을 때 질의 범위를 만족하는 범위들을 찾는다. 그리고 질의결과는 각 범위에 저장된 집계 값들을 계산하여 획득한다. 그래서 BINT는 질의 처리시 집계 값을 중복 계산 하지 않는다. 그러나 BINT는 계층별로 저장공간을 필요로 한다. 이것은 저장비용을 증가하게 한다. 그리고 스트림 데이타 입력 시 계층구조를 재구성 해야 하므로 삽입비용이 증가한다. 또한 질의 처리시 집계 값은 질의 범위를 만족하는 범위들을 찾고 재계산을 해야 하기 때문에 데이타 검색비용이 증가한다. 삽입비용과 검색비용의 증가는 전체적인 질의처리 성능의 저하를 가져오게 된다.

그 밖에 LINT[7]라는 공유기법도 있다. 이 방법은 BINT와 유사하게 범위와 계층 구조를 사용한다. 그러나 BINT의 단점인 질의 영역을 만족하는 범위를 검색하는 속도를 향상시켰다. 그러나 이 기법은 계층구조 구축시간과 공간 낭비가 BINT보다 크다. 그러므로 질의 지연 시 유지해야 하는 공간이 커지게 되고, 데이타가 빠르게 삽입 시 구축 시간이 길어져 전체적인 성능하락을 초래하게 된다.

위의 두 연구는 자원 공유를 위해서 제안되었다. 그러나 일반적으로 자원 공유를 위해 유지해야 하는 메모리 요구량이 튜플을 공유하는 경우보다 크다. 그리고 질의 처리를 위해 질의 영역을 만족하는 범위를 찾는 시간이 많이 필요하다. 게다가 시간기반 질의와 튜플기반 질의가 동시에 요청되는 경우를 고려하지 않았다. 그러므로 이 경우 추가적인 부하를 야기할 수 있다.

그러므로 다중 질의 처리 최적화는 검색과 삽입을 고려하고, 시간기반 질의와 튜플기반 질의가 동시에 수행 가능한 자료구조가 필요하다. 삽입을 최적화하기 위해서는 데이타 입력 시 자료구조를 재구성 하지 않아야 한다. 그리고 검색을 최적화 하기 위해서는 집계 값을 한 번만 계산해야 하고, 공유자원내에서 질의 범위를 빠르게 검색해야 한다. 그리고 데이타를 유지할 때 시간과 튜플 수를 모두 고려해야 한다. 이 조건을 모두 만족하는 자원공유 자료구조가 필요하다.

본 논문에서는 다수의 연속질의에서 슬라이딩 윈도우 집계질의를 최적화 하기 위한 선형 자료공유를 제안한다. 제안기법은 동시에 실행중인 튜플기반 질의의 윈도우 길이를 기반으로 데이타 관리 크기를 결정한다. 이를 팬(Pane)[9]이라고 한다. 그리고 시간단위의 질의를 위해 시간단위로 Pane을 그룹화 하는 Time Unit을 사용한다. 팬과 Time Unit은 선형 구조의 자원 공유를 제공한다. Pane은 데이타의 집계 정보만을 선형 구조로 저장하므로 저장공간의 비용을 감소시킨다. 또한 선형구조로 이루어진 제안기법은 데이타 입력 시 들어온 데이타에 대해 Pane의 집계정보만을 계산한다. 그러므로 기존 구조의 재구성 등으로 인한 비용이 없어 삽입비용이 감소한다. 또한 Pane은 데이타 대신에 집계 정보를 유지하므로 질의의 집계 결과를 빠르게 얻을 수 있다. 그리고 선형 구조를 사용하기 때문에 질의 범위를 만족하는 Pane을 더욱 빨리 찾을 수 있다. 이는 검색 비용의 감소시켜 질의 처리 속도가 증가시키고 질의 지연에 따른 메모리 낭비가 줄어들게 된다. 또한 Time Unit을 이용해 Pane을 시간 단위로 그룹화 하기 때문에, 시간기반 질의와 튜플기반 질의가 동시에 실행되는 환경에서도 적용 가능하다. 따라서 제안기법은 본 논문에서 요구하는 다중 질의 환경에서 매우 효율적이다.

본 논문의 구성은 다음과 같다. 2장에서 관련연구를 통해서 슬라이딩 윈도우에 대해서 다룬다. 그리고 기존의 다중 집계 질의를 위한 방법인 BINT와 LINT에 대

해서 다루고 마지막으로 튜플 재계산을 방지하기 위한 Pane에 대해서 간단히 언급한다. 3장에서는 본 논문에서 제안하는 기법에 대해서 자세히 설명한다. 우선 Pane의 크기를 결정하는 방법에 대해서 설명하고, Pane의 생성, Pane의 삭제에 대해서 설명한다. 4장에서는 이 기법을 사용하여 질의 처리를 하는 과정을 예를 들어서 설명한다. 5장에서 성능분석을 하고 마지막으로 6장에서 결론을 맺는다.

2. 관련연구

2.1 슬라이딩 윈도우

스트림 데이타는 연속적으로 들어오는 대용량의 데이타이다. 스트림 데이타상의 질의 처리는 디스크에 저장되어 있는 데이타를 처리하는 기존의 방식과 다른 질의 처리 방법을 요구한다. 따라서 기존의 질의와 다른 형태인 연속질의가 제안되었다. 모든 연속질의는 슬라이딩 윈도우를 포함하고 있다. 슬라이딩 윈도우는 일정시간 또는 지정된 개수의 튜플로 스트림 데이타를 분할하여 질의 처리에 이용한다. 그리고 슬라이딩 윈도우를 이동하면서 질의 처리를 계속적으로 진행한다. 현재까지 슬라이딩 윈도우에 대한 연구가 많이 진행되었다. 최근 J. Li[8]의 연구에서 슬라이딩 윈도우에 포함된 연산을 가지고 윈도우의 종류를 구분하였다.

슬라이딩 윈도우는 시간을 기반으로 윈도우 크기를 결정하는 시간 기반 슬라이딩 윈도우와 튜플을 기반으로 윈도우 크기를 결정하는 튜플 기반 슬라이딩 윈도우가 있다. 시간 기반 슬라이딩 윈도우를 이용한 질의를 시간 기반 질의 또는 TS 질의라고 하며, 튜플 기반 슬라이딩 윈도우를 이용한 질의를 튜플 기반 질의 또는 ROW 질의라고 한다. TS 질의와 ROW 질의는 RANGE와 SLIDE 절을 포함한다. RANGE는 슬라이딩 윈도우의 질의 영역의 크기를 명시하는 것이다. 시간 기반에서는 시간의 범위가 나타나며 튜플 기반에서는 튜플의 개수가 나타나게 된다. SLIDE는 슬라이딩 윈도우의 이동 크기를 명시하는 것이다. 시간 기반에서는 SLIDE의 값만큼 시간이 지난 데이타로 슬라이딩 윈도우를 이동하며, 튜플 기반에서는 슬라이딩 윈도우를 SLIDE의 값만큼 튜플의 개수를 이동한다. 이와 같은 특성을 갖는 슬라이딩 윈도우의 질의 구조는 다음과 같다.

SELECT * FROM S [RANGE r, SLIDE s WATTR type]

여기서 r과 s가 슬라이딩 윈도우의 이동과 범위를 결정 한다. 그리고 type에서 TS와 ROW를 결정 할 수 있다. 각 r과 s의 단위는 type이 TS이면 시간이고, ROW이면 튜플의 개수이다. 질의 수행은 지정된 type에 따라 s만큼 윈도우가 이동하고 r만큼 범위에서 질의를 수행한다. 그리고 다시 또 s만큼 이동하고 r만큼 수행을 반복한다.

이와 같은 처리 방식으로 인해 슬라이딩 윈도우는 두 가지 특징을 가진다. 첫 번째는 슬라이딩 윈도우가 중복된 영역을 포함하면서 이동한다는 점이다. r은 s보다 크므로 슬라이딩 윈도우의 동일한 튜플을 두 번 이상 포함하면서 이동한다. 따라서 슬라이딩 윈도우는 동일한 튜플을 질의 처리에 사용하게 되어 연산 비용을 낭비하게 된다. 두 번째는 질의 처리와 스트림의 입력 속도와의 연관성이다. 스트림의 입력 속도가 빨라지면 TS 질의는 RANGE와 SLIDE가 커진다. 그러나 ROW 질의는 언제나 고정이기 때문에 스트림의 입력 속도가 증가하면 튜플 기반 질의는 질의 수행 횟수가 증가하게 된다.

위 두 가지 특성은 다중 질의 환경에서 다음과 같은 문제를 야기 할 수 있다. 먼저 스트림의 입력 속도가 증가하는 상황에서 TS질의와 ROW질의가 같이 수행되는 경우에는 ROW질의는 질의 수행 횟수가 늘어나게 되어 질의 처리 시간이 늘어나며 이는 질의 지연을 가져온다. 더구나 튜플을 중복 계산하게 되면 현재 시점까지의 처리가 더욱 지연 될 수 있다. 이 경우 자원을 공유하기 위한 메모리 비용이 증가 하게 되므로 위의 두 가지 특성을 고려한 자원 공유 기법이 필요하다.

2.2 계층구조의 자원 공유 기법

스트림 환경에서 다중 질의를 위한 기존의 연구로는 계층구조의 자원 공유 기법인 BINT[7]가 있다. BINT는 범위(Interval)와 계층(level)을 사용한다. 범위는 집계정보를 갖는 단위를 의미하며 계층은 각 계층마다 포함하고 있는 집계 값의 튜플 개수를 의미한다. 이는 계층이 0일 때 2개의 튜플에 대한 집계 정보를, 계층이 1일 때 4개의 튜플에 대한 집계 정보를 포함하며 각각을 범위이라고 부른다. BINT는 자원 공유를 위해 계층별로 범위를 생성하여 집계 정보를 저장하므로 메모리 비용이 증가한다. BINT에 질의 요청이 오면 질의에 대한 슬라이딩 윈도우의 범위를 만족하는 계층을 찾고 각 계층별로 저장된 집계 값들을 가지고 전체 질의의 결과를 반환한다. 이는 계층별로 저장된 포함 관계가 없는(disjoin) 범위 중 슬라이딩 윈도우의 영역을 만족하는 최적의 범위들을 찾아야 하기 때문에 계층별 탐색 시간이 길어 지고, 각 범위들의 집계 값으로 결과 값을 얻는 비용도 요구된다.

그리고 유사한 기법인 LINT[7]도 있다. LINT는 BINT와 같이 범위와 계층 구조를 사용한다. LINT는 계층구조를 좌우로 구축하여, 두 개의 영역만 검색하면 질의 결과를 얻을 수 있다. 그래서 BINT의 단점인 질의 범위를 만족하는 범위 찾는 속도를 향상시켰다. 그러나 이 기법은 두 개의 계층구조를 사용하기 때문에 계층구조 구축시간과 공간 낭비가 BINT보다 크다. 그러

므로 질의 지연 시 유지해야 하는 메모리 공간이 커지게 되고, 데이타가 빠르게 삽입 시 계층구조 구축 시간이 길어진다.

2.3 튜플 재계산을 방지하는 Pane 구조

슬라이딩 윈도우는 질의 처리를 위해 버퍼를 이동할 때 중복된 영역을 포함 할 수 있다. 슬라이딩 윈도우의 RANGE의 범위가 SLIDE보다 큰 경우에 발생한다. 이 경우에는 이전에 질의 처리에 사용했던 부분을 다음 질의 처리에 사용하게 된다. 결국 중복된 영역의 질의처리에 따라 연산비용이 증가한다. 그래서 중복 영역을 재계산 하지 않기 위해 Pane[9] 구조가 제안되었다. Pane은 질의 영역을 분할하여 집계 값을 계산하고 유지한다. 그리고 질의 요청 시 분할된 영역에 저장된 집계 값을 이용해서 질의 결과를 반환한다. 자세한 처리 과정은 다음과 같다.

Pane은 SLIDE와 RANGE의 최대공약수로 만큼의 단위로 스트림을 분할하고 분할 영역만큼의 집계 값을 튜플 대신 저장한다. 질의 요청이 오면 분할 저장된 Pane의 집계 값을 이용하여 질의 결과를 반환하기 때문에 튜플에서 집계 값을 구하는 연산은 Pane을 생성할 때 한번만 수행하면 된다. 이는 Pane을 사용하지 않는 구조가 같은 튜플에 대해서 여러 번 집계 값을 구하는 것보다 비용이 크게 감소하게 된다. 그림1은 슬라이딩 윈도우의 튜플 재계산에 대한 예를 보여준다. 그림 1은 150개의 튜플에서 집계 값을 계산해야 하는 경우로 Pane을 사용하여 240개의 중복된 튜플 계산 비용을 감소시킨다.

스트림 데이타 처리에서 Pane을 사용하면 중복 계산이 줄어들어 질의 처리 속도가 크게 향상 된다. 그러나 현재까지 연구된 Pane 구조는 단일 질의에 대해서만 적용이 가능하였으며, 특히 ROW질의와 TS질의가 같이 수행되는 다중 질의 환경에서는 Pane의 크기를 결정할 수 없었다. 그리고 SLIDE와 RANGE가 서로소인 경우에 Pane의 크기를 결정할 수 없다. 이와 같은 단점을 보완하여 다중 질의 환경에서 Pane의 장점을 이용할 수 있도록 확장이 필요하다.

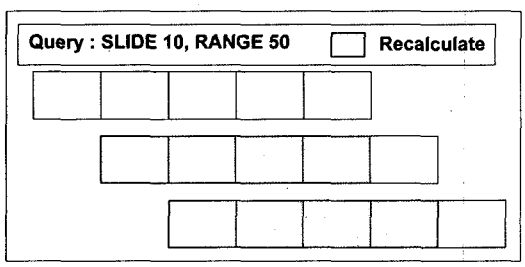


그림 1 슬라이딩 윈도우의 튜플 재계산

3. 다중 질의 처리를 위한 자원공유

본 장에서는 다중 질의 환경에서 질의 처리 속도를 향상시켜 질의의 지연을 줄이고 메모리의 효율성을 높이는 자원 공유기법에 대해 설명한다. 제안기법은 SPQP (Shared Pane Query Processing)라는 새로운 메모리 관리 기법을 이용하여 질의를 처리한다. SPQP에서 질의 처리는 크게 두 단계에 걸쳐서 이루어지는데, 첫 번째는 스트림 처리 영역으로 버퍼에 입력되는 스트림에서 질의들이 필요로 하는 집계 정보를 계산해서 Pane이라는 선형의 자료구조로 저장한다. 두 번째는 질의 처리 영역으로 Pane 버퍼에 저장된 집계 값들을 공유해 여러 개의 집계 질의가 동시 수행되고, 사용이 끝난 Pane들을 삭제한다.

제안기법은 집계 값을 한번만 계산 하는 Pane 구조를 다중 질의 환경으로 확장하였다. 이 Pane은 윈도우 이동에 따라 발생하는 중복된 영역의 연산 비용을 크게 감소 시킨다. 특히 공유구조가 선형이기 때문에 질의 요청 시 질의가 요구하는 집계 정보를 만족시키는 Pane들을 찾고 질의 결과를 반환하는 것이 BINT의 계층 탐색에 비해 매우 빠르다. 따라서 SPQP를 이용한 시스템은 다중 질의 환경에서 질의 지연을 감소시켜, 결과적으로 메모리 비용을 크게 감소시킨다. 또한 메모리에 튜플 대신 요약정보인 집계 정보를 공유하므로 질의를 위한 데이타를 유지하는 비용이 크게 감소한다.

3.1 SPQP(Shared Pane Query Processing)

SPQP는 Pane 자료구조에 집계 정보를 공유하고 질의를 수행하는 질의 처리 시스템이다. 이 집계 정보를 공유하여 다중 질의 환경에서의 메모리 비용과 질의 비용을 줄 일수 있다. SPQP는 크게 두 개의 처리 부분으로 나누어 진다. 스트림 영역과 질의 영역이다. 스트림 영역(Stream Area)은 스트림 버퍼(Stream Buffer)와 이를 처리 하는 SPC(Stream ProCess)로 구성되어 있고, 질의 영역은 집계 값(Aggregation Value)을 가지고 있는 Pane들의 버퍼와 이를 처리 하는 질의 실행기 (Query Executer)로 구성된다.

전체적인 질의 처리 과정은 그림 2와 같다. 우선 등록된 질의들의 정보를 가지고 Pane과 Time Unit의 크기를 구한다. (Time Unit은 Pane들의 시간단위 모임이다) 이 값들은 모든 질의들의 정확한 집계 정보를 유지하기 위한 최적의 크기이다. 그 다음에 스트림 영역의 SPC가 Pane과 Time Unit의 크기를 가지고 Pane을 생성하는 규칙을 생성한다. 그리고 이 규칙에 따라 스트림으로 입력되는 튜플들의 집계 값을 구해서 Pane 버퍼에 저장한다. 그리고 일정 시간 마다 이 작업을 반복적으로 수행한다. 이와 같이 스트림 영역를 통해 생성된 Pane 버퍼상에서 실제 질의가 수행된다. 질의 처리는 질의 실행기

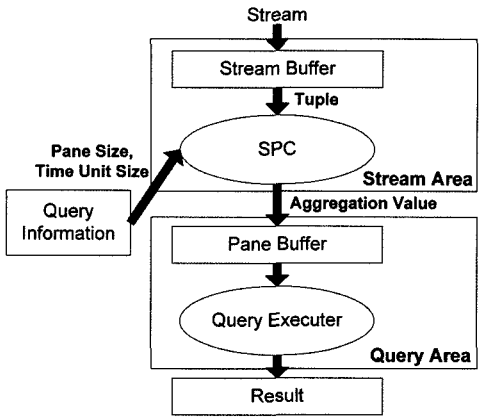


그림 2 SPQP 처리 과정

에서 등록된 모든 질의를 가지고 Pane버퍼에 저장된 집계 값들을 가지고 결과를 빠르게 제공한다.

SPQP의 자세한 구조는 그림 3과 같다. 먼저 스트림 영역에는 스트림 버퍼와 Calculate Pointer(CP)가 있다. 스트림 버퍼는 메모리를 이미 반환한 Discard(D)상태와 아직 사용 중인 Used(U) 상태를 가진다. CP는 현재 튜플들의 집계 정보를 계산하고 있는 위치를 가리킨다. 그 다음으로 질의 영역에서는 Pane(P_i)들이 Time Unit 단위로 저장되어 있다. 그리고 현재 질의의 수행위치를 가리키는 Query Pointer(QP_i)와 가장 느리게 실행되는 질의의 위치를 가리키는 Last Pointer(LP) 두 가지를 가지고 있다. 그리고 각 Pane은 Time Unit에 포함관계에 따라서 완전히 포함되는 Complete(C)상태와, 부분적으로만 포함되는 Incomplete(I) 상태를 가진다.

그림 3은 SPQP에서의 질의 수행 예를 보여주고 있다. 그림 3의 시스템에서는 Q1,Q2 두 개의 질의가 수행중이다. Q1은 TS질의 이고 Q2는 ROW질의 이다. 그래서 Q2는 정확한 Pane크기의 배수로 실행하면서 저장된 Pane들의 집계 정보를 가지고 질의를 수행한다. 그러나 TS질의인 Q1은 길이가 가변이기 때문에 정확히 Pane의 배수가 아니다. 그러므로 Time Unit 단위로 저장된 Pane들을 가지고 질의 결과를 반환한다. Q1은 7개의 C 상태 Pane과 3개의 I상태로 저장된 Pane을 가지고 있는 Time Unit을 사용해서 질의 결과를 반환한다.

특별히 그림 3의 경우는 질의 Q2가 지연되고 있는 상황을 보여주고 있다. 만약 Pane 구조가 아닌 튜플을 공유하는 구조를 사용하면 질의 처리 시마다 튜플들로부터 집계 정보를 계산해야 한다. 그리고 윈도우 이동에 따라 중복되는 영역의 재계산의 추가 비용 때문에 질의가 더욱 지연 될 수 있다. 그러나 본 논문의 구조에서는 질의 처리시 집계 값의 중복 계산이 필요 없다. 그리고 선형 구조의 자원 공유를 사용하기 때문에 질의 영역을 위해 저장된 Pane들을 검색하는 속도가 매우 빠르다. 그래서 지연 된 질의를 보다 빠르게 처리해 비교적 빨리 지연 상태에서 벗어 날 수 있다.

지금까지 SPQP에 대해서 설명하였다. SPQP는 Pane과 Time Unit 단위로 집계 값을 저장하기 때문에 Pane과 Time Unit의 크기에 따라 성능이 크게 차이 날 수 있다. Pane의 크기가 매우 작으면 튜플을 유지하는 비용보다 집계 정보를 유지하는 비용이 더 클 수 있다. 그리고 Time Unit의 크기가 매우 작으면 Incomplete 상태의 Pane이 많아지므로 저장해야 하는 Pane의 수가 증가하므로 역시 비효율 적이다. 그런데 Pane과 Time Unit을 가지고 각각 ROW질의와 TS질의의 질의영역을 정확히 구성할 수 없으면 정확도를 보장 할 수 없다. 그러므로 Pane과 Time Unit의 크기 결정이 매우 중요하다.

그래서 시스템에 등록된 모든 질의의 길이와 최적의 메모리 효율을 동시에 만족하는 Pane과 Time Unit의 크기 결정 방법을 3.2절에서 설명한다. 3.3절에서는 Pane을 생성하는 알고리즘을 설명하고, 3.4절에서는 생

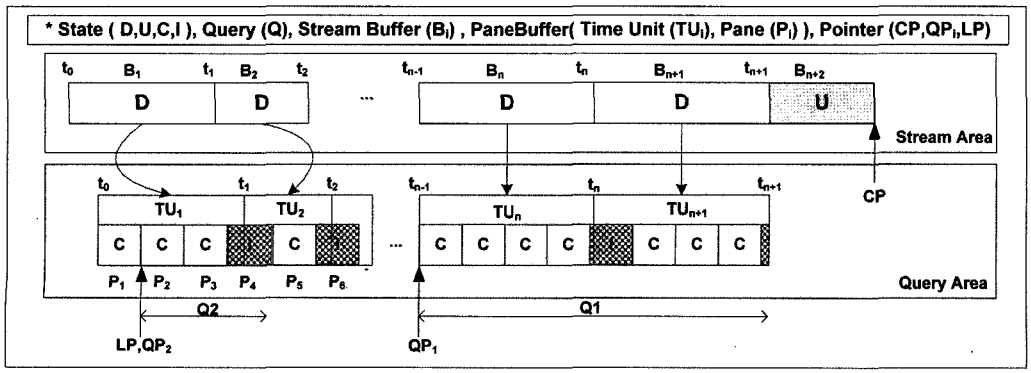


그림 3 SPQP 구조

성된 Pane에서 질의를 수행하는 방법에 대해서 설명한다. 마지막으로 3.5절에서는 더 이상 사용하지 않는 Pane을 삭제하는 알고리즘에 대해서 설명한다.

3.2 Pane과 Time Unit의 크기 결정

Pane은 일정한 단위로 튜플들의 집계정보를 저장하는 자료 구조이다. Pane은 집계 정보를 저장하고 질의 수행을 하기 때문에, 튜플 재계산을 막는데 유용하다[9]. 특히 본 논문의 환경은 여러 개의 질의가 동시에 수행되기 때문에, 튜플 재계산을 줄여주면 각 질의 처리 속도가 증가해 전체 시스템의 성능이 크게 향상 될 수 있다. 그러나 3.2절에서 언급했듯이 Pane의 크기에 따라서 성능이 더욱 떨어질 수 있다. 그래서 Pane을 사용하는 SPQP에서는 Pane크기 결정이 매우 중요한 부분이고, Pane은 크기는 SPQP의 실행을 위해 가장 먼저 요구되는 사항이다. 본 절에서는 이 최적의 Pane의 크기를 결정하는 방법에 대해서 설명한다.

최적의 Pane의 크기는 두 가지 조건을 만족 해야 한다. 첫 번째는 생성된 Pane으로 실행중인 모든 ROW질의의 질의 영역을 나타낼 수 있어야 한다. 이 Pane을 그림4에서처럼 Dividable Pane이라고 부르기로 한다. 두 번째는 Pane을 유지하기 위해 필요한 메모리 비용이 튜플을 유지하기 위한 메모리 비용 보다 작아야 한다.

그러나 Dividable Pane의 크기를 찾는 것은 매우 어렵다. 만약 동시에 실행중인 질의들의 각 SLIDE들과 RANGE의 길이들의 최대공약수(GCD:Great Common Divisor)를 사용하면 간단히 Dividable Pane의 크기를 얻을 수 있다. 그러나 만약 하나의 질의라고 SLIDE나 RANGE의 크기가 서로소 관계에 있다면 최대공약수는 1이 된다. 이 경우는 튜플을 유지하는 것과 동일하므로 Pane을 사용하는 이득이 전혀 없게 된다. 그러므로 좀 더 개선된 방법으로 Pane의 크기를 구해야 한다.

Pane의 크기를 결정하는 방법은 SLIDE와 RANGE의 관계에 따라서 3가지 경우로 나눌 수 있다. 우선 Q1,Q2 두 개의 질의에 대해서 본다. Q1질의와 Q2질의는 각각 RANGE가 r_1, r_2 이고 SLDIE가 s_1, s_2 라고 가정한다. 그렇다면 첫 번째로 r_1, r_2, s_1, s_2 가 모두 서로소가 아닌 경우이다. 이 경우를 "완전 나누어진다"(AD)라고 한다. 두 번째는 단일 질의의 RANGE와 SLIDE가 서로소인 경우이다. 이 경우는 "불완전 나누어진다"(ID)라고 한다. 그리고 그 이외의 경우는 "나누어지지 않는다"(UD)라고 한다. 그래서 Pane을 구하는 방법은 AD,ID,UD 세 가지로 나눌 수 있다.

그럼 우선 AD의 경우에 대해서 Dividable Pane 크기를 결정하는 방법에 대해서 설명 한다. 현재 수행중인 n개의 질의를 $Q_i(1 \le i \le n)$ 이라고 할 때, 각 질의의 SLIDE와 RANGE를 각각 s_i 와 r_i 라 하자. a,b 의 최대공약수는 $g = \gcd(a,b)$ 라고 표현하기로 하자.

정리. 모든 질의가 AD일 때 Dividable Pane의 크기는 $P_{size} = \gcd(g_1,...,g_n)$ 이다.

Proof) 모든 질의는 $f : N -> R$ 의 수열로 표현될 수 있다. 각 질의는 $f_i(n) : s_i n + r_i$ 로 표현될 수 있다. 그리고 P_{size} 를 사용한 Pane의 수열은 $f_P(n) : P_{size} n + P_{size}$ 라고 하면, f_P 의 P_{size} 는 모든 i에 대해 s_i 와 r_i 의 최대공약수이므로, $f_i \subset f_P$ 가 성립한다. 그러므로 P_{size} 는 Dividable Pane의 크기이다.

정리에 따라 모든 ROW질의의 SLIDE와 RANGE의 GCD가 Dividable Pane의 크기이다. 그러므로 이 $P_{size} = \gcd(g_1,...,g_n)$ 를 Pane의 크기로 사용할 수 있다. 이 경우는 AD 조건으로 SLIDE와 RANGE가 서로소가 아니므로 1이 아닌 값을 P_{size} 로 얻을 수 있다. 알고리즘 1은 AD의 경우 Pane의 크기를 구하기 위한 알고리즘이다. 시스템에 등록된 모든 ROW질의의 SLIDE와 RANGE의 크기를 가지고 모든 질의의 GCD가지고 Pane의 크기를 얻을 수 있다.

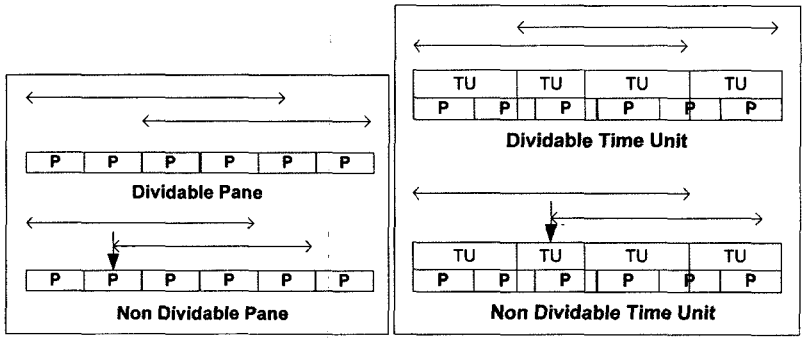


그림 4 Dividable Pane and Dividable Time Unit

알고리즘 1 Pane 크기 결정 알고리즘

Input
s_i : SLIDE size of query, r_i : RANGE size of query, N : number of querys
Output
P : The size of Dividable pane

Procedure GetPaneSize()
Begin
01 P = gcd(s_1,r_1)
02 For i=2 to N Do
03 {
04 g = gcd(s_i,r_i)
05 P = gcd(P,g)
06 }
End

ID의 경우에는 최대공약수만으로 P_{size}를 구할 수 없다. ID의 경우는 SLIDE와 RANGE 가 각각 서로소이기 때문에 P_{size}가 1이 된다. 그래서 ID의 경우 최대 공약수로 Pane의 크기를 결정하면 비 효율적이다. 이때는 하나의 Pane의 크기가 아닌 두 개의 Pane의 크기를 이용할 수 있다. 이 두 개의 Pane크기를 반복적으로 사용하면 Dividable Pane을 만들 수 있다.

ID의 경우를 설명하기 위해 다음과 같은 상황을 고려한다. RANGE가 30이고 SLIDE가 7인 경우에 Dividable Pane이 되기 위해서는 Pane을 나누는 값들이 윈도우에 이동에 따라 결정되는 영역의 좌측,우측 값을 모두 포함해야 한다. 즉 Pane은 SLIDE의 크기에 비례하는 좌측영역의 값인 7의 배수에서 나누어져야 한다. 그리고 우측 값은 SLIDE와 RANGE의 크기에 동시에 영향을 받는다. 이 좌측은 우측과 비교하여 값이 증가하는 크기가 같은 특성이 있다. 그래서 우측영역의 값은 30+7a와 같이 7의 배수에 30을 더한 형태로 표현할 수 있다. 이는 30 = 7·4 + 2이므로 결국 7에 배수에 항상 2를 더한 형태이다. 결국은 두 RANGE와 SLIDE는 2의 차이를 유지하면서 계속 이동한다. 그러므로 이 차이인 2와 5=7-2를 사용하여 처음에는 2 다음에는 5로 나누면서 반복적으로 Pane을 생성하면 Dividable Pane이 될 수 있다. 그림 5에서 위 예를 보여준다.

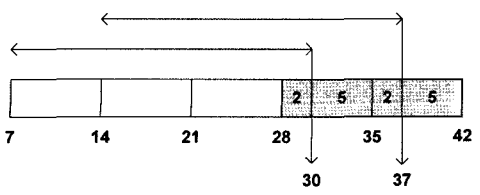


그림 5 ID에서 두 개의 Pane 크기

그림 5와 같이 2,5와 단위로 연속적으로 Pane을 분할하면 Dividable Pane이 될 수 있다. 이 ID의 특성을 일반화 하면 다음과 같다. 질의의 SLIDE와 RANGE를 각각 s와 r라 하자. 모든 SLIDE들의 GCD를 g_s라 하고 RANGE들의 GCD는 g_R라고 하고 P_a는 $g_R \equiv P_a \pmod{g_s}$라고 하자. 이때 Dividable Pane의 크기는 P_a와 $g_R - P_a$로 만족 시킬 수 있다. 이 두 값의 단위로 번갈아 가면서 연속적으로 Pane을 생성하면 Dividable Pane이 될 수 있다.

이와 같이 AD와 ID가 하나만 존재하는 경우에 Pane의 크기를 결정 할 수 있다. 그러나 AD와 ID가 동시에 존재하는 경우, 즉 UD의 경우에는 본 논문에서 제안하는 기법을 바로 적용할 수 없다. 이 UD의 경우에는 질의들을 잘 그룹화해서 AD나 ID형태의 질의끼리 분리할 수 있다. 이 여러 개의 그룹이 생성되는 UD의 Pane 생성비용은 중간 집계 값을 계산하면서 여러 개의 그룹에 해당하는 Pane의 위치에서만 적용하면 되므로, 집계값 재계산이 필요 없다. 그러나 그룹의 증가에 따른 추가적인 메모리 유지 비용이 필요하다. 이 비용은 AD와 ID의 그룹의 증가횟수와 비례한다. 그러나 Pane의 경우 메모리 사용량에서 매우 효율적이므로 그룹의 수가 Pane의 크기 보다 많지 않으면, 메모리 측면에서 여전히 우수하다.

튜플을 공유하는 경우와 Pane을 사용하는 경우 간단한 비용함수는 다음과 같다. Tc는 튜플의 수이고 Ts는 튜플의 크기이다. As는 집계 값을 저장하는데 필요한 크기이고, Pc는 팬의 크기이다.

$$Tuple : Tc \times Ts \tag{1}$$

$$Pane : \frac{Tc \times As}{Pc} \tag{2}$$

(1)은 튜플을 공유하는 일반적인 공유 기법의 비용이다. 그리고 (2)는 그룹이 하나일 때 Pane을 사용할 경우의 비용이다. 일반적으로 As는 Ts 비해서 작다. Pc가 크면 메모리 사용에서 매우 효율적이다. 그러므로 Pc의 크기가 크게 결정되면 그룹이 많이 생성되어도 여전히 메모리 사용이 효율적이다. 그러므로 UD의 경우 그룹화를 이용해도 상당히 효율적이다.

그룹화 기법은 다소 많은 내용의 알고리즘이 요구 되므로 본 논문에서는 간단히 그룹화 기법에 대한 소개만 다룬다. 그룹화 기법은 그룹을 나눌 때, 그룹별 Pane의 크기가 최대가 되도록 분할 해야 한다. 질의 별 SLIDE와 RANGE의 GCD가 1000,500,310,155 일 때, 1000,500과 310,155로 그룹을 생성하는 것이 1000,310와 500,155

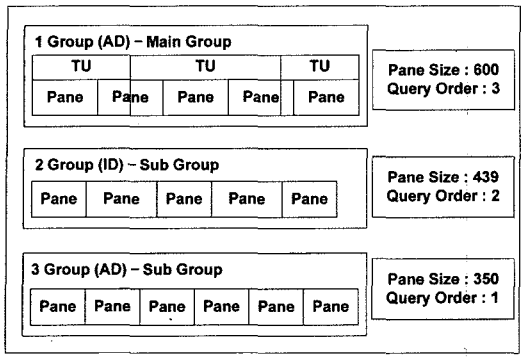


그림 6 Pane 그룹화 기법

로 그룹을 생성하는 것 보다 효율적이다. 그리고 메모리 사용량이 큰 Pane의 크기가 작은 그룹부터 먼저 질의를 수행하여 자원 사용률이 높은 그룹의 메모리를 빠르게 반환하는 방법으로 자원 사용률을 높일 수 있다.

그림 6에서 AD나 ID 관계가 하나 이상이 공존할 경우(UD의 경우) Pane을 그룹화 하는 기법을 소개한다. 그림 6에는 메인 그룹(Main Group) 하나와 서버 그룹(Sub Group) 두 개로 총 세 개의 그룹이 존재한다. 메인 그룹은 본 논문에서 사용하는 Time Unit과 Pane이 모두 가지고 있는 그룹이다. TS질의는 메인 그룹에서만 실행된다. 서브 그룹은 오직 Pane만 사용하는 그룹이다. 이때는 그룹에 속하는 ROW질의만 자원을 사용하고, 질의의 슬라이딩 윈도우가 이동 시 즉시 자원반환이 가능하다. 그래서 Pane의 크기에 따라서 질의를 우선 처리하여 자원효율을 높일 수 있다.

그룹1과 3은 Pane 크기가 모두 동일한 AD이다. 각각의 Pane 크기는 600과 350이다. 그룹2는 Pane의 크기가 두 가지가 필요한 ID이고 평균 Pane 크기는 439이다. 여기서 Pane의 크기가 가장 작은 그룹3이 동일한 튜플들에 대해서 가장 많은 Pane이 필요하므로 메모리가 많이 필요하다. 그러므로 Pane의 크기 별로 질의를 우선 실행해서, 빠르게 메모리 반환이 가능하게 할 수 있다. 그럼에서는 그룹3,그룹2,그룹1의 순서로 우선순위를 할당한다.

지금까지는 튜플수에 의해 결정되는 Pane에 대해서 언급하였다. 마지막으로 시간에 의해 결정되는Time Unit에 대해 설명한다. Time Unit은 Pane을 시간 단위로 그룹화하여 TS질의가 사용하기 위한 자료구조이다. TS질의는 시간마다 다른 수의 튜플을 가지므로, 특정 개수로 Dividable 하게 만들 수 없다. 그렇지만 TS질의 역시 시간 단위로는 고정이기 때문에 Pane과 유사하게 시간단위를 이용해서 Pane을 그룹화 하는 것이 Time Unit이다. 이를 Dividable Time Unit 이라고 하자. Dividable Time Unit의 크기 결정 알고리즘은 시간 단위인 것을 제외하고는 Pane의 경우와 동일하므로 생략한다. 그러나 Time Unit은 Pane보다 우수한 조건이 있다. TS질의는 길이가 가변이기 때문에 RANGE나 SLIDE가 소수여도 사용이 가능하다. 특히 스트림 환경에서 소수의 GCD인 1초면 상당히 많은 양의 데이타가 들어오므로 GCD가 최소 단위인 1초여도 사용이 가능하다. 결론적으로 Time Unit은 Pane의 AD와 동일하게 모든 RANGE와 SLIDE의 GCD로 Time Unit의 크기에 사용한다.

3.3 Pane 생성 알고리즘

본 절에서는 주어진 Dividable Pane과 Dividable Time Unit 크기를 가지고 Pane에 집계정보를 생성하는 방법에 대해서 설명한다. SPQP에서는 스트림 처리와 질의 처리가 분리되어 있다. Pane 생성은 스트림 처리 부분에서 이루어지고 질의 처리 부분에서 생성된 Pane을 사용한다. 질의들을 위한 최적의 Pane과 Time Unit의 크기를 가지고 스트림으로부터 입력되는 튜플로부터 계속해서 Pane을 생성한다.

Pane은 두 가지 경우 생성된다. 첫 번째는 스트림으로 부터 입력되는 튜플들이 Pane의 크기만큼 계산 된 경우이다. 두 번째는 튜플들이 Pane의 크기만큼 계산되지 않았지만 Time Unit에 해당하는 시간이 된 경우이다. 두 경우 모두 Pane이 임시 메모리에 생성된다. 그리고 첫 번째 경우의 Pane은 C 상태이고 두 번째 경우는 I 상태이다. 그리고 이렇게 임시메모리에 모인 Pane들의 집합이 하나의 Time Unit으로 이루어져 Pane 버퍼에 저장된다.

그림 7에 Pane 생성 과정이 자세히 나와 있다. 현재 2t부터 3t시점까지 B3버퍼에서 집계 정보를 계산 중이다. 버퍼에서 CP(Current Pointer)가 이동하면서 현재 위치까지의 집계 정보를 임시 메모리에서 계산하고 있다. 현재 P_6,P_7,P_8까지 C 상태로 Pane을 임시메모리에 생성하였고, P_9를 생성 중이다. 시간이 지나면 CP가 P_9를 생성하고 P_{10}을 수행하다가 Time Unit의 끝을 알리는 타임스탬프(timestamp) 값이 3t인 튜플이 들어오게 될 것이다. 그러면 I 상태인 P_{10}이 생성되고, TU_3 [P_6,P_7,P_8,P_9,P_{10}]와 같이 Pane들을 그룹화해서 Pane 버퍼에 저장된다. 그리고 이것이 질의 수행에 이용된다.

Pane생성시 고려할 사항이 한가지 더 있다. Pane은 지속적으로 버퍼에서 튜플을 읽어서 집계값을 저장하고, 최종 Pane생성시까지 생성된 집계 값을 Pane에 저장한다. MIN,MAX,CNT,SUM,AVG 중에서 MIN,MAX, CNT,SUM은 C와 I 상태의 Pane 모두에서 정확도가 유지된다. 그러나 AVG는 Pane의 크기에 따라 정확도가 차이가 날수가 있다. 그러므로 AVG는 직접 값을 저장하지 않고 CNT와 SUM의 두 값으로 나누어 저장하

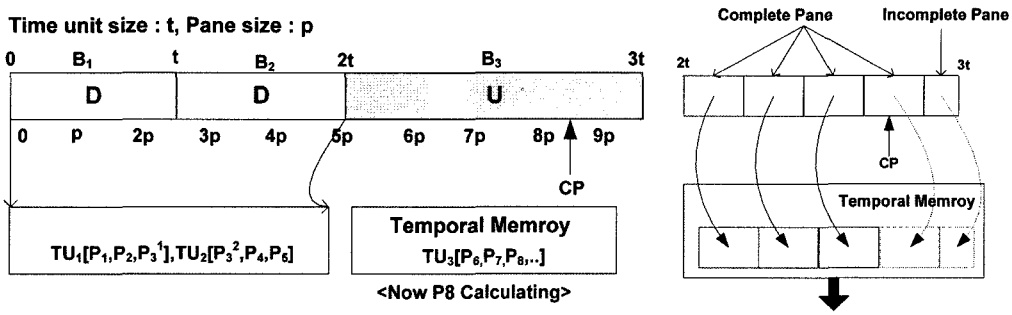


그림 7 Pane 생성과정

알고리즘 2 Pane 생성 알고리즘

```
Input
  T : next of Time Unit, P : size of Pane, CP :
  Current Pointer
Output
  TU : Time Unit (The group of Pane)
Procedure MakePane()
Initialize
01 Init(Pane);
02 Count = prevCnt; // 이전까지 저장된 Pane의
   개수(C,I 상태를 구분)
03 GT = GT + T; //현재 까지의 Global Time
   Unit
Begin
01 While(true)
02 {
03 Tuple = Buffer(CP); //버퍼에서 CP에 가리키는
   튜플을 얻어온다
04 CP++;
05 ComputeAggregation(Tuple,Pane); //Pane에
   집계 값 갱신
06 Count++;
07 if( PaneSizeFull(Count) ) //Pane이 꽉 찼는지 확인
08 {
09  if( prevCnt == 0 ) //이전 Time Unit의
    C상태이면
10    Pane.state = "C";
11  else //이전 Time Unit의 I 상태이면
12    Pane.state = "I";
13  InsertPane(TU,Pane); //TU에 Pane을 추가
14  Init(Pane);
15 }
16 if(tuple.timestamp >= GT) //시간이 꽉 찼는지
   확인
17 {
18  if( PaneSizeFull(Count) )
19    Pane.state = "C";
20  else
21    Pane.state = "I"
22    prevCnt = EmptyPaneCnt();
23    InsertPane(TU,Pane);
24  Return TU;
25 }
26 }
End
```

여 정확도를 유지할 수 있다.

알고리즘 2는 Pane을 생성하고 정해진 주기에 따라서 Pane들의 집합인 Time Unit을 반환하는 알고리즘이다. 우선 Count와 GT를 초기화 한다. Count는 튜플의 개수를 세기 위한 변수이다. 이 변수는 Pane을 저장하는 기준으로 사용한다. Count는 초기에 prevCnt로 초기화 되는데, 이는 이전 Time Unit에서 C혹은 I상태로 저장된 것을 판단하기 위한 값이다. GT는 Time Unit의 다음 위치를 가리킨다. 이 값은 Pane들을 생성하여 Time Unit의 집합으로 반환하는 시점을 나타내는 기준으로 사용된다. 알고리즘의 실행을 보면 03-04줄에서 CP를 이동하면서 버퍼에서 튜플을 가지고 온다. 그리고 05 현재 Pane에 튜플들의 집계 값을 계산한다. 그리고 07줄에서 PaneSizeFull은 AD와 ID에 따라 Pane의 꽉 찼는지 확인하는 함수이다. 그래서 Pane이 꽉 찬 경우 Pane의 상태를 C로 설정하고 TU에 저장한다. 그러나 Pane이 꽉 차지 않은 경우는 시간이 지난 것이므로 I 상태로 설정한다. 그리고 위 과정을 반복적으로 수행하여 튜플의 timstamp가 GT에 도달하면, 마지막으로 17줄에서 작업중이던 Pane의 상태를 Pane의 저장상태에 따라 I혹은 C로 할당하고 TU에 저장하고 최종 TU를 반환한다.

3.4 Pane 버퍼에서 질의 수행

여기서는 SPQP에서 선형 Pane 구조에 공유된 집계정보를 이용해서 질의를 처리하는 기법에 대해 설명한다. 먼저 질의 수행을 위해서 쿼리의 현재 수행 위치를 가리키는 Query Pointer(QP)를 사용한다. 질의 수행은 ROW 질의에 대해서 결과를 반환하는 방법과 TS 질의에 대해서 결과를 반환하는 두 가지 방법이 있다. ROW

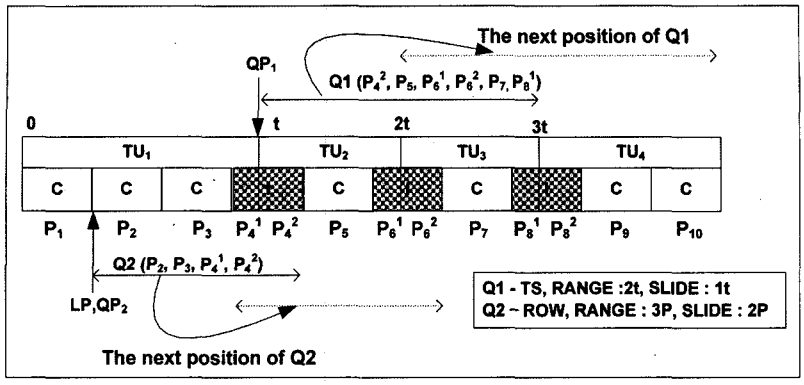


그림 8 Pane 버퍼에서 질의 수행

는 Pane 버퍼에 저장된 Pane들을 기준으로 결과를 반환하고, TS는 Time Unit을 기준으로 결과를 반환한다.

그림 8을 보면 TS질의 Q1은 Time Unit을 단위로 QP를 이동시키면서 질의 결과를 구하고 있다. Q1은 t 시점에서 Time Unit 두 개에(RANGE 2t) 해당하는 6개의 Pane(P_4^2, P_5, P_6^1, P_6^2, P_7, P_8^1)에 저장된 집계 값들을 가지고 결과를 반환한다. ROW질의 Q2는 Pane 단위로 QP를 이동시키면서 질의 결과를 구하고 있다. Q2는 세 개의 Pane 크기(RANGE 3P)에 해당하는 4개의 Pane(P_2, P_3, P_4^1, P_4^2)에 저장된 집계 값들을 가지고 결과를 반환한다.

3.5 Pane 삭제 알고리즘

여기서는 생성된 Pane 버퍼에서 더 이상 사용되지 않는 Pane들을 삭제하는 방법에 대해서 설명한다. 삭제를 위해서 질의의 마지막 수행 위치를 가리키는 Last Pointer(LP)를 사용한다. 질의 삭제는 간단한 알고리즘으로 질의가 수행될 때 마다, QP를 이동하면서 가장 느린 QP에 해당하는 위치를 LP에 설정한다. 그리고 LP가 이전 Time Unit의 위치에서 다음 Time Unit의 위치로 이동 되면 이전 위치의 Time Unit을 삭제하여 Pane 버퍼에서 제거한다.

그림 9는 Pane 삭제과정을 보여주고 있다. Pane 버퍼에서 QP_1, QP_2, QP_3 세 개의 질의가 수행 중이다. 현재 QP_3가 마지막 위치이므로 LP는 QP_3에 위치한다. 그리고 이전 Time Unit과 포함되는 Pane들을 삭제하고 있다.

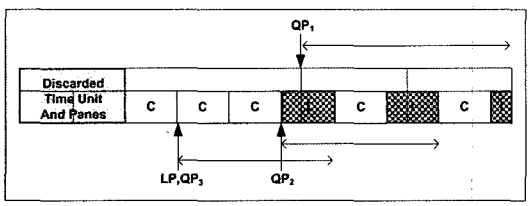


그림 9 Pane 삭제과정

알고리즘 3 Pane 삭제 알고리즘

```
Input
  QP : Query Pointer, LP : Last Pointer, TU
Output
  True or False
Procedure DeletePane()
Initialize
01 TU = GetTimeUnit (LP);
Begin
01 DeletePane()
02 {
03   QP = GetNextQP(QP);
04   if ( IsLatestPointer(QP) )
05   {
06   LP = QP;
07  }
08  if( TU != GetTimeUnit(LP) )
09  {
10   Delete(TU);
11  }
12 }
End
```

알고리즘 3은 더 이상 사용하지 않는 Pane을 Time Unit 단위로 삭제하는 알고리즘 이다. 삭제를 위해서 QP와 LP를 사용한다. 우선 현재 LP의 현재 Time Unit을 TU에 설정한다. 그리고 QP가 이동할 때 마다 QP의 위치를 갱신하고(3줄) 해당 QP가 현재 가장 느린 Pointer이면 LP를 이 QP로 변경한다.(4-7줄) 그리고 LP가 이전 TU에서 다음 TU로 변경된 경우 이전 TU를 완전히 삭제한다.(8-10줄)

지금까지 다중 질의 환경에서 질의 처리를 빠르게 수행하고 메모리 효율적으로 사용하기 위한 자원 공유기법에 대해서 설명하였다. 다음 장에서는 이 기법을 적용해서 실제 질의 처리를 수행 하는 과정에 대해서 예를 들어 설명 한다.

4. SPQP을 이용한 질의 처리

이번 장에서는 몇 가지 질의 예를 가지고 본 논문에서 제안하는 SPQP에서 처리 과정을 설명한다. 도로에서 현재 지나가는 자동차의 속도의 집계 값을 가지고 도로의 상황을 모니터링 하는 시스템에 본 논문에서 제안하는 기법을 적용한다. 그림 10에는 주어진 도로의 6개 영역에서 속도를 측정하고 있다. 현재 스트림으로 들어오는 데이타의 스키마는 <Area id, Car id, speed, ts>와 같다. Area id는 해당 도로의 지역 id에 해당한다. Car id는 자동차의 고유 id이고, speed는 측정된 시기의 자동차의 속도이고 ts는 측정된 시간이다. 이 상황에서 표 1과 같이 Q1,Q2,Q3 세 개의 집계 질의가 동시에 수행 중이다. Q1은 TS질의이고, Q2와 Q3는 ROW 질의이다.

SPQP의 실행 순서의 첫 번째로 Pane과 Time Unit의 최적의 크기를 결정해서 Pane 생성을 위한 규칙을 생성한다. 그리고 지정된 규칙에 따라서 지속적으로 Pane을 생성하고, 생성된 Pane에서 질의를 수행하는 부분이 동시에 수행된다. 그리고 더 이상 사용하지 않는 Pane을 버퍼에서 제거하면서 메모리를 관리한다. 그럼 위의 예를 가지고 SPQP실행순서에 따라 적용한다.

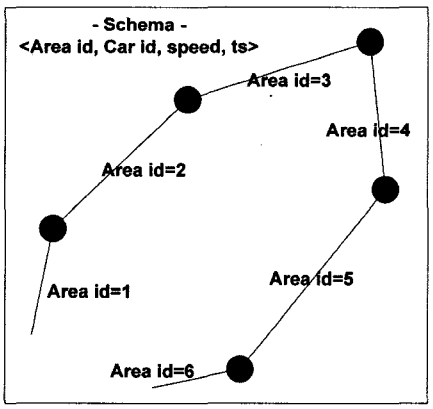


그림 10 도로와 스키마

표 1 도로 상황 시스템에 등록된 3개의 질의

```
Q1 :  SELECT min(speed),max(speed),AreaID FROM S
      [ RANGE 180 seconds SLIDE 60 seconds
       WATTER TS GROUP BY AreaID]
Q2 :  SELECT avg(speed), AreaID FROM S
      [ RANGE 200 SLIDE 50
       WATTER ROW GROUP BY AreaID]
Q3 : SELECT max(speed),avg(speed), AreaID FROM S
      [ RANGE 400 SLIDE 100
       WATTER TS GROUP BY AreaID]
```

먼저 SPQP의 실행을 위해서 우선 최적의 Pane과 Time Unit의 크기를 결정해야 한다. ROW 질의인Q2와 Q3는 AD관계이다. 그리고 Q1은 단일 TS질의이다. 그러므로 Pane의 크기는 Q2와 Q3질의를 가지고 구한다. Pane의 크기를 위해 ROW질의인 Q2와 Q3의 SLIDE와 RANGE의 GCD를 구한다. Pane의 크기 P_{size}는 gcd(gcd(200,50),gcd(400,100)) = 50 이므로 50이다. 그리고 Time Unit의 크기는 Q1의 GCD인 gcd(180,60) = 60 이므로 1분이다. Pane과 Time Unit의 크기를 결정했으므로, 이제는 Pane 생성 알고리즘을 이용해서 실제로 Pane을 생성한다. 스트림을 통해서 다음과 같이 튜플들이 입력되고 있다.

그림 11에는 입력중인 튜플 들이 있고 현재 시점은 16:31:00이다. Time Unit이 1분이므로 튜플로부터 집계정보가 임시메모리에 16:32:00까지 계산된다. 그리고 임시메모리에 계산하면서 입력된 튜플 수가 Pane의 크기인 50이 되면 임시메모리에 Pane을 생성한다. 그림 10은 CP가 이동하면서 50인 크기에서 Pane을 하나 생성했다. Pane에는 질의가 필요로 하는 집계정보가 저장된

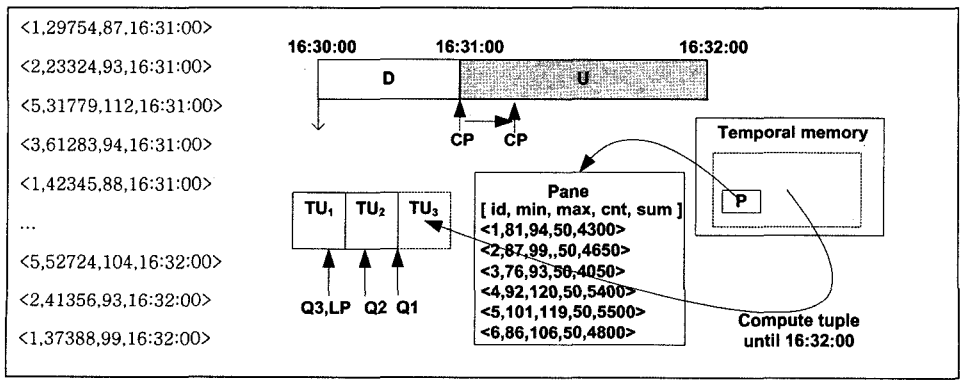


그림 11 Pane 생성 및 처리 과정

다. 현재 질의는 id별로 그룹화된 min, max, avg을 필요로 한다. 특별히 avg는 cnt와 sum로 나누어 저장되어야 한다. 그래서 Pane에는 <id, min, max, cnt, sum>형태로 id의 개수만큼 저장한다. 그리고 계속해서 집계 정보를 계산하다가 16:32:00의 튜플이 도착하면 계산을 완료한다. 이 시점에 저장된 Pane들의 집합인 Time Unit을 Pane버퍼의 TU_3에 저장한다.

이제 Pane이 생성되었고 Q1,Q2,Q3 질의가 수행된다. Q1은 1초씩 이동하면서 3분에 해당하는 만큼을 수행한다. 현재 Q1은 TU_3에 위치한다. Q1은 여기서부터 3분의 범위 만큼 수행해야 하므로 TU_5가 생성될 때 까지 질의 수행을 대기한다. Q2는 현재 TU_2안에 있다. Q2는 RANGE가 200이므로 Pane 4개에 저장된 집계 값들을 가져온다. 그래서 각 id별로 4개의 cnt와 sum를 가지고 다시 평균을 내서 결과 값을 반환한다. Q3는 Range가 400이므로 Pane 8개에 대해서 동일한 작업을 수행한다. 그리고 Q3가 수행하면서 TU_2의 범위 안에 들어오면 LP가 TU_1에서 완전히 빠져 나오기 때문에 TU_1을 버퍼에서 삭제한다. 위의 과정을 반복적으로 수행하면서 질의 처리를 계속적으로 수행한다.

5. 성능평가

SPQP는 TS와 ROW 질의가 동시에 수행이 되어도 질의가 지연되지 않고 빠르게 응답이 가능하다. 그리고 질의 처리를 위해서 튜플을 보유하지 않고, 요약 정보의 집계 값만 보유하여 메모리 효율을 높인다. 성능평가는 이 두 가지를 확인하는 관점에서 진행 하였다.

5.1 실험 환경 및 실험 데이타

실험 평가에 사용된 시스템 환경은 CPU가 Pentium 4 3.0 GHz이고 메모리는 2GB이고 운영체제는 Fedora Linux 4이다. 개발 환경은 gcc-4.0 버전이다. 그리고 실험에 사용된 데이타는 스트림의 입력 속도를 증가시키기 위해 초당 500,000개까지의 튜플을 인위적으로 생성할 수 있는 프로세스에 의해 생성되는 데이타이다. 튜플 스키마는 <area id, car id, speed, ts 와> 같으며 튜플 당 기는 16byte이다. 실험 데이타는 집계 질의를 위해 area id가 크게 6개의 그룹으로 분할 되서 생성 된다. 그리고 다중 집계 질의 환경과 TS와 ROW의 동시 수행 테스트를 위해 시스템에는 TS질의 한 개와 ROW 질의 두 개를 사용한다. 실험에 사용된 질의는 표 2와 같다.

5.2 실험평가

본 논문에서 제안하는 기법의 우수성을 증명하기 위해 5.1절에서 정의한 환경에서 질의 처리 속도를 측정하고, 튜플의 입력 속도에 따른 질의 지연율과 메모리 사용량을 측정한다. 측정을 위해서 3가지 기법을 사용한다. 첫 번째는 튜플을 공유하는 가장 기본적인 형태인 Normal이다. 그리고 계층별로 범위를 가지는 BINT, 마지막으로 본 논문에서 제안하는 SPQP에 대해서 각각 측정하고 결과를 비교한다.

표 2 실험에 사용된 질의 3개의 질의(Q1-TS, Q2,Q3-ROW)

```
Q1 : SELECT min(speed),max(speed), AreaID FROM S
     [ RANGE 10 seconds SLIDE 5 seconds
      WATTER TS GROUP BY AreaID]
Q2 : SELECT avg(speed), AreaID FROM S
     [ RANGE 200 SLIDE 50
      WATTER ROW GROUP BY AreaID ]
Q3 : SELECT max(speed),avg(speed), AreaID FROM S
     [ RANGE 400 SLIDE 100
      WATTER TS GROUP BY AreaID]
```

먼저 질의 처리 속도를 측정하기 위해, 질의 처리에 영향을 미치는 두 가지 요인을 평가한다. 하나는 자원공유를 위해 사용하는 집계 정보를 생성하는 시간과, 생성된 집계정보에서 질의 처리에 필요한 부분을 검색하고 질의를 처리하는데 소요된 시간을 측정한다. 이 실험은 집계정보 생성시간을 측정하기 위해서, 집계 정보 생성에 사용할 튜플들을 미리 생성한다. 그리고 생성된 튜플들에서 집계 정보 생성을 완료하기까지 걸리는 시간을 측정한다. 그리고 생성되는 튜플 수를 계속 변화시키면 반복 수행한다. 그리고 생성된 집계 정보에서 질의 처리에 필요한 데이타를 검색하고 질의를 처리하는 시간을 측정하기 위해서는 집계 정보를 미리 생성한다. 그리고 생성된 집계 정보에서 질의처리를 완료하는데까지 걸리는 시간을 측정한다. 그리고 집계 정보를 위한 튜플 수를 변화시키면서 반복 수행한다. 이 실험은 질의 처리 속도를 측정하기 위한 실험이므로 단일 ROW질의로 수행한다. TS질의는 가변이지만 ROW질의는 고정 길이이므로 정확한 질의 처리 속도 평가를 위해서 보다 적합하다. 그래서 표 2의 Q2질의 하나를 가지고 실험하였다. 그리고 SPQP의 Time Unit의 크기는 2초이고 Pane의 크기는 50이다.

질의 처리 속도 비교를 위해 그림 12의 실험 결과를 얻었다. 우선 집계 정보를 생성하는 그래프를 보면 Normal은 튜플을 자체를 공유하기 때문에 집계 정보 생성 시간이 필요 없다. BINT는 계층 별로 Interval을 모두 생성해야 하고, 집계 정보 재구성이 필요하다. 반면에 SPQP는 선형 구조로 튜플 한 개는 하나의 Pane에만 속한다. 그래서 중복이 없기 때문에 수행시간이 상대적으로 빠르다. 그 다음 집계 정보에서 실제로 질의 수행 시간을 평가한 두 번째 그래프의 실험 결과를 보면 SPQP가 가장 빠르고 그 다음이 BINT이고 Normal

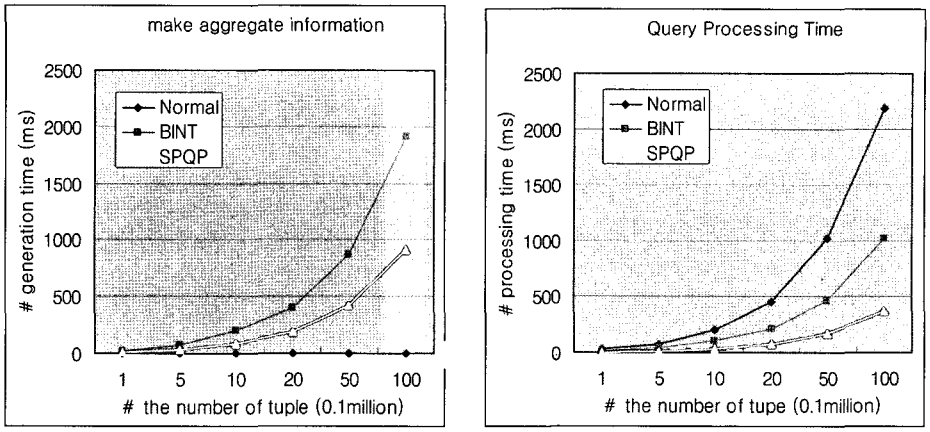


그림 12 주어진 튜플에서 집계 값 정보를 생성하는 시간과 결과값을 얻는 시간

이 제일 느리다. Normal은 튜플 자체를 공유하기 때문에 슬라이딩 윈도우가 이동하면서, 동일한 튜플을 재계산하게 되므로 처리 속도가 상당히 느리다. 그리고 BINT는 계층 구조로 집계 정보를 분할하기 때문에, 질의 수행영역을 만족하는 집계 정보를 찾는 비용이 비교적 크다. 마지막 SPQP는 선형구조로 바로 집계 정보를 얻을 수 있기 때문에 처리속도가 가장 빠르게 된다.

다음은 스트림 입력 속도에 따른 질의 지연과 메모리 사용량을 측정하기 위해 다음과 같은 실험 방법을 사용하였다. 질의 지연을 측정하기 위해 스트림을 1분 동안 입력을 하면서 동시에 질의를 처리한다. 그리고 1분후 스트림의 입력을 멈춘다. 스트림의 입력이 멈춘 시점과 질의의 마지막 결과 시점을 비교해서 그 차이를 측정한다. 위 과정을 스트림의 입력속도를 바꾸면서 반복 수행한다. 메모리 사용량을 측정하기 위해서는 메모리 관리자를 사용한다. 메모리 관리자는 직접 OS로부터 미리 큰 메모리를 할당 받고, 이후에 메모리 할당과 해제를 OS를 대신해 직접 관리한다. 그리고 할당과 해제 할 때의 양을 자체적으로 측정한다. 그래서 SPQP에서 메모리 사용량을 측정하기 위해 메모리 관리자를 사용한다. 질의 지연 측정과 마찬가지로 메모리 측정도 스트림을 1분 동안만 입력한다. 그리고 스트림 입력이 종료되는 시점에서 보유하고 있는 메모리의 양을 측정한다.

스트림의 입력속도에 따른 비교를 위해 그림 13의 결과를 얻었다. 질의지연을 측정한 그래프를 보면튜플의 입력 속도가 매우 크지 않을 때는 대부분이 지연 없이 처리가 가능하다. 그러나 튜플의 입력속도가 빨라질수록 질의 지연의 차이가 커지는 것을 알 수 있다. 그리고 메모리 사용량도 질의 지연과 동일하게 차이가 발생한다. 결과적으로 스트림의 입력 속도가 증가할 때 SPQP가 가장 우수함을 알 수 있다. 그러므로 튜플의 입력속도가 빠른 스트림 환경에서 다중 질의 수행에는 SPQP가 가

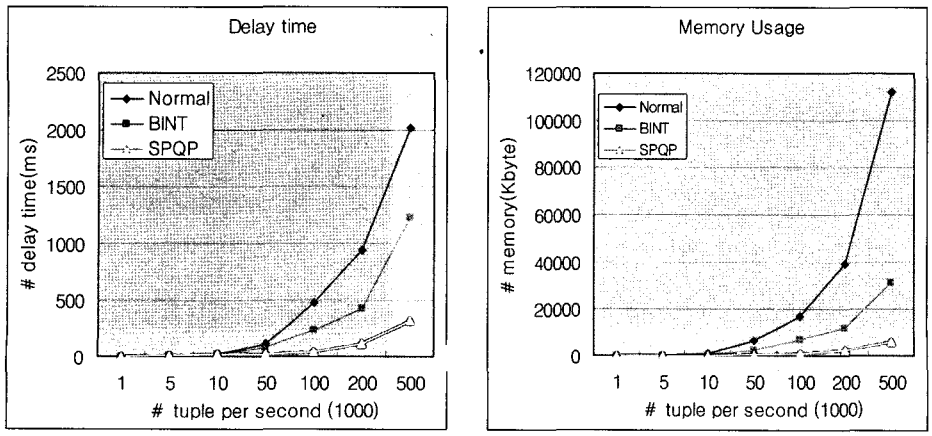


그림 13 스트림의 입력속도에 따른 질의 지연과 메모리 사용량

장 적합하다.

6. 결론 및 향후 연구

본 연구에서는 다중 질의 수행 환경에서 질의 지연을 줄이고 메모리 효율성을 높이는 자원 공유기법을 제안하였다. 제안기법에서 SPQP의 핵심은 Pane과 Time Unit 단위로 집계 값을 공유하는 것으로 슬라이딩 윈도우가 가지는 튜플의 중복 계산 문제를 해결 하여 질의 처리 속도를 향상 시켰다. 또한 집계 값을 Pane에서 한 번만 공유하기 때문에, 계층별로 모두 집계 값을 유지하는 기존기법보다 메모리 효율이 증가하였다. 선형 구조의 이용은 삽입연산과 집계값 검색연산의 비용을 감소시켰으며 이는 전체적인 질의 지연을 감소시켰다. 성능 평가를 통하여 본 논문에서 제안하는 환경에서 제안기법이 기존 기법보다 스트림 데이타 환경에서 질의 지연 시간이 감소하고 메모리 사용량이 감소하였음을 보였다.

향후 연구로는 등록된 질의들 중에 AD와 ID 관계가 공존하는 경우나, UD의 관계가 발생 하는 경우 Pane을 그룹화해서 관리하는 기법에 대해 연구할 것이다. 또한 제안기법에 다중 집계 값 공유[13]에 관한 연구를 적용하여 집계 정보 처리시 더욱 효율적으로 처리가 가능하도록 연구할 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] R. Motwani, J. Widom, A. Arasu, B. Babcock, S. Babu, M. Datar, G. Manku, C. Olston, J. Rosenstein, and R. Varma., "Query Processing, Resource Management, and Approximation in a Data Stream Management System," In Proc of CIDR, 2003.

[3] Abadi, D. J, Carney, D., Centintemel, U., Cherniack, M., Convey, C., Lee, S., Stonebraker, M., Tatbul, N., Zdonik, S., "Aurora: A New Model and Architecture for Data Stream Management," VLDB Journal, 2003.

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

[5] S. Chandrasekharan and M. J. Franklin., "Streaming queries over streaming data," In Proc of the 28th Intl. Conf. On VLDB, pp. 203-214, Aug. 2002.

[6] Hammand, M., Franklin, M., Aref, W., and Elmagarmid, "A. Scheduling for shared window joins over data streams," In Proc of the 29th VLDB Sep, 2003.

[7] Arvind Arasu, Jennifer Widom, "Resource Sharing in Continuous Sliding-Window Aggregates," In Proc. of the 30th VLDB 2004.

[8] 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.

[9] J. Li, D. Maier, "No Pane, No Gain : Efficient Evaluation of Sliding-Window Aggregates over Data Streams," SIGMOD Record, Vol. 34, No. 1, March 2005.

[10] M. Datar, A. Gionis, P. Indyk, and R. Motwani. "Maintaining stream statistics over sliding windows," In Proc. of the 13th Annual ACM SIAM Symp. On Discrete Algorithms, pp. 635-644, Jan. 2002.

[11] J. Gehrke, F. Korn, and D. Srivastava. "On computing correlated aggregates over continual data streams," In Proc. of the 2001 ACM SIGMOD Intl. Conf. on Management of Data, pp. 13-24, May 2001.

[12] P.B. Gibbons and S. Tirthapura, "Distributed streams algorithms for sliding windows," In Proc. of the 14th Annual ACM Symp. On Parallel Algs. And Architectures, pp. 63-72, Aug. 2002.

[13] R. Zhang, Nich. Koudas, "Multiple Aggregations Over Data Streams," In Proc. of the 2005 ACM SIGMOD Intl. Conf. on Management of Data, pp. 299-310, June 2005.

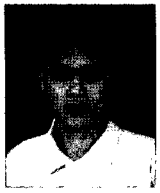
[14] Tucker, P., Maier, D., Sherad, T. and Fegaras, L. "Exploiting Punctuation Semantics in Continuous Data Streams," Transactions on Knowledge and Data Engineering, 15,3, May 2003.

[15] Hammand, M., Aref, W., Franklin, M., Mokbel, M., and Elmagarmid, A.K. "Efficient Execution of Sliding Window Queries over Data Streams," Purdue University Department of Computer Sciences Technical Report Number CSD TR 03-035, Dec 2003.

[16] A. Dobra, M. N. Garofalakis, J. Gehrke, and R. Rastogi, "Sketch-based multi-query processing over data streams," In EDBT, 2004.

[17] N. Koudas and D. Srivastava, "Data stream query processing: A tutorial," In VLDB, 2003.

[18] R. E. Gruber. B. Krishanmurthy, and E. Panagos. "READY: A high performance event notification system," In proc. of the 16th Intl. Conf. on Data Engineering, pp. 668-669, Mar. 2000.



백 성 하

2005년 인하대학교 수학과 졸업(이학사) 2006년~현재 인하대학교 컴퓨터정보공학과 석사과정. 관심분야는 스트림 데이타베이스, 공간 데이타베이스, 분산 데이타베이스, 하이브리드 데이타베이스

유 병 섭

정보과학회논문지 : 데이타베이스
제 33 권 제 5 호 참조

조 숙 경

1990년 인하대학교 전자계산학과(이학사). 1994년 인하대학교 대학원 전자계산공학과(공학석사). 2002년 인하대학교 대학원 전자계산공학과(공학박사). 2003년~2006년 8월 인천대학교 컴퓨터공학과 강의전담교수. 2006년 9월~현재 인하대 지능형 GIS 센터 연구원. 관심분야는 데이타베이스, 실시간 데이타베이스 시스템, 스트림 데이타베이스

배 해 영

정보과학회논문지 : 데이타베이스
제 33 권 제 5 호 참조