ISSN 1598-0170 (Print)
ISSN 2287-1136 (Online)
http://www.jksii.or.kr

데이터 스트림에서 다중 조인 연속질의의 효과적인 처리를 위한 전처리 기법☆

Preprocessing Method for Handling Multi-Way Join Continuous Queries over Data Streams

서 기 언* 이 주 일** 이 원 석***
Ki Yeon Seo Jooil Lee Won Suk Lee

요 약

데이터 스트림이란 빠르게 연속적으로 무한히 발생하는 데이터 집합을 의미한다. 최근 다양한 산업의 발달로 인해 이러한 스트림 데이터의 효율적인 처리를 위한 요구 사항들이 늘어나고 있다. 특히 많은 연산 비용을 요구하는 조인 연산의 효율적인 처리는 데이터 스트림 관리 시스템의 성능 향상에 많은 영향을 미친다.

본 논문에서는 다중 조인 연속질의의 효율적인 처리를 위하여 최종 질의 결과에 포함되지 않는 불필요한 중간 조인 결과들을 사전에 제거함으로써 조인 연산의 비용을 감소시키는 방법을 제안한다. 이를 위해 스트림 데이터의 모니터링을 위한 매트릭스 기반의 구조체를 제안하고, 제안된 구조체를 이용한 매트릭스 연산을 통하여 최종 조인 결과의 투플 수를 예측함과 동시에 불필요한 중간 결과들을 만들어내는 투플들을 찾아낸다. 이를 통해 해당 투플을 이용한 조인 연산의 수행 여부를 결정하여 최종 조인 결과를 만들지 않는 투플을 조인 연산에서 배제함으로써 효율적으로 다중 조인 연속 질의를 처리한다.

ABSTRACT

A data stream is a series of tuples which are generated in real-time, incessant, immense, and volatile manner. As new information technologies are actively emerging, stream processing methods are being needed to efficiently handle data streams. Especially, finding out an efficient evaluation for a multi-way join would make outstanding contributions toward improving the performance of a data stream management system because a join operation is one of the most resource-consuming operators for evaluating queries.

In this paper, in order to evaluate efficiently a multi-way join continuous query, we propose a novel method to decrease the cost of a query by eliminating unsuccessful intermediate results. For this, we propose a matrix-based structure for monitoring data streams and estimate the number of final result tuples of the query and find out unsuccessful tuples by matrix multiplication operations. And then using these information, we process efficiently a multi-way join continuous query by filtering out the unsuccessful tuples in advance before actual evaluation of the query.

☞ keyword : 데이터 스트림(data stream), 질의 처리(query processing), 조인 연산(join operation)

1. 서 론

최근 고도산업사회에 접어들면서 과학 기술이나 공학 분야 이외에 경제·사회 등의 다양한 분야에서도 각종 데이터들의 중요성이 강조되고 있으며, 데이터 종류의 다양화, 대용량화, 그리고 연속적인 발생 특성으로 인하여 일정한 저장 공간에 데이터를 저장한 후 관리 활용하기에 어렵게 되었다. 따라서 한정된 저장 공간을 효율적으로 처리하는 방식으로 점차 변화하고 있다. 이러한 종류의 데이터로는 센서 네트워크, 주식거래, 금융거래, 전화 통화, 교통관리 및 교통상황수집자료, SNS 등에서 찾아볼 수 있으며, 이처럼 연속적으로 발생되어 빠른 속도로 관리 시스템에 입력되는 데이터를 데이터 스트림(Data Stream)이라고 한다. 따라서 기존에 데이터 처리에 사용되는 DBMS(Data Base Management System)에 데이터를 저장한 후 처리하는 방식을 적용할 수 없다. 이미 많은 연구에서 오랫동안 데이터 스트림의 처리 기술을 연구해 Aurora[1], STREAM[2,3], TelegraphCQ[4], Borealis[5] 등과

* 정 회 원 : 삼성전자 DMC 연구소 근무
skybuilder9@gmail.com
** 정 회 원 : 연세대학교 대학원 컴퓨터과학과 박사과정
tad@database.yonsei.ac.kr
*** 정 회 원 : 연세대학교 컴퓨터과학과 정교수 (교신저자)
leewo@database.yonsei.ac.kr
[2012/03/15 투고 - 2012/03/20 심사 - 2012/05/15 심사완료]
☆ 이 논문은 2012년도 정부(교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(No. 2011-0016648).

같은 DSMS(Data Stream Management System)가 개발된 바 있다. 이러한 시스템들은 모두 데이터 스트림 처리를 목적으로 만들어졌으며 효과적인 질의 처리를 위해서 질의 계획을 최적화 하고, 특정 연산들을 통해 연속질의로 입력되는 데이터 스트림에 대한 질의를 처리하고 있다. 또한 과다한 입력 데이터로 인한 시스템 부하를 방지하기 위해 데이터를 선택적으로 처리할 수 있도록 하는 기능을 제공한다. 그 중 STREAM은 CQL(Continuous Query Language)이라는 언어를 제안하고 CQL을 통해 관계연산자 집합을 만들어 입력·저장되는 데이터에 적용함으로써 연속적으로 질의를 처리하도록 하였다.

연속질의 처리에는 크게 선택조건(selection)의 처리와 조인 연산의 처리가 있다. 각 투플의 지정된 속성 값이 주어진 조건을 만족하는지 여부를 판단하는 단일 입력 연산인 선택조건 처리에 비해 조인 연산은 조인에 참여하는 두 입력 데이터의 모든 투플을 비교해야 하므로 연속질의 처리에 있어서 큰 비용을 요구한다. 조인 연산에서 가장 큰 비용을 차지하는 부분은 새로 입력된 투플과 조인 상대 스트림의 투플과 비교 하는 부분이다. 특히 다중 조인의 경우 여러 개의 조인 연산을 거치면서 결과적으로 최종 결과를 만들지 않는 중간 결과들이 많이 생성되므로 중간 결과를 생성하는 비용을 줄이면 전체적으로 조인 비용을 많이 줄일 수 있을 것이다.

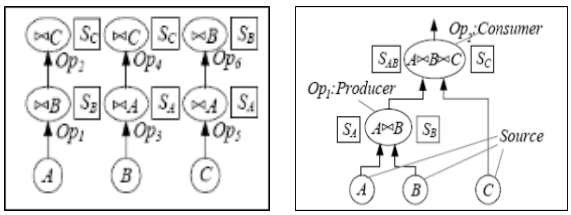
본 논문에서는 질의의 최종 결과에 영향을 미치지 않는 중간 결과들을 생성하지 않음으로써 효율적으로 다중 조인 연속 질의를 처리하는 방법(MBJE: Matrix-based Join Evaluation)을 제안한다. 이를 위해 실시간 데이터 모니터링을 위한 매트릭스 기반의 구조체를 제안하고, 이 구조체들을 활용하여 실제 조인 연산을 하기 전에 질의의 최종 결과에 포함되지 않는 투플들을 찾아내는 방법을 제안한다. 이를 통해 최종적으로 조인 결과를 생성하지 않는 투플들에 대해서는 처음부터 조인 연산에 참여하지 않도록 함으로서 필요 없는 연산을 제거하고, 실제 조인 결과가 필요 없는 모니터링 질의 등에 대해서 빠른 응답을 줄 수 있다.

2. 관련 연구

센서 네트워크, RFID 처리 등 빠르게 발생하는 데이터 스트림을 처리하기 위한 연구가 최근 몇 년 간 집중적으로 수행되어 왔다. 특히 Aurora, Eddy[6-8], STREAM 등 범용 목적의 DSMS 개발과 함께 스트림 질의어, 로드 쉐딩(load shedding), 효율적인 질의 처리 기법 등 많은 이슈들에 대해서 연구가 진행되었다. DSMS는 무한히 들어오는 입력 데이터들을 처리하는 것이 목적이기 때문에 기존의 DBMS에서 사용하는 방법으로는 조인 연산이나 집계(aggregation) 연산과 같은 블로킹 연산은 처리할 수 없다. DSMS는 슬라이딩 윈도우(sliding window)[9]를 적용하여 위와 같은 문제들을 처리하지만 선택조건이나 항목선택(projection)과 같은 단일 입력 연산[10]보다 더 복잡하고 더 많은 시간이 필요하다. 주어진 연속질의의 최적화된 질의 계획을 생성하기 위하여 STREAM에서는 현재 등록된 연산자들의 선택도(selectivity)를 모니터링하여 선택도가 작은 연산자들을 먼저 처리하도록 적응적으로 연산자들의 순서를 바꾸는 greedy 방법을 제안하였고, 연산자 스케줄링[2]을 통해서 메모리의 사용량을 최소화하고 처리량을 최대화한다. 또한 Eddy에서는 입력되는 투플들을 정해진 질의 계획 없이 현재 상황에서 가장 적합한 순서를 결정하여 라우팅함으로써 연속질의들을 처리하며, 동적으로 질의 계획을 바꾸는 방법(Dynamic Plan Migration)[10]도 제안하였다.

질의 계획을 효율적으로 처리하기 위한 연구 중 특히 다중 조인 연산을 빠르게 처리하기 위한 연구가 많이 진행되고 있다. MJoin[12]은 중간 결과를 만들지 않고 각 데이터 소스마다 정해진 조인 순서에 따라서 새로 들어온 데이터를 처리한다. 적응적 캐싱(Adaptive Caching)[13]은 같은 조인 연산이 반복되는 MJoin의 단점을 보완하기 위하여 중간 결과들을 적응적으로 캐싱함으로써 메모리를 더 사용하는 대신 보다 빠르게 조인 연산을 수행하는 방법이며, [14]에서는 MJoin의 각 데이터 소스마다 최적화된 조인 순서를 결정하는 방법을 제안한다. 반대로 X-Join[15]은 다중 조인을 n-1개의 이진 조인 트리로 구성하고, 각 조인들의 중간 결과들을 저장하는 방법이다. 그림 1은 MJoin과 XJoin의 동작 방식을 보여준다. 이러한 조인 방법들은 중간 결과의 저장 여부와 관계없이 최종적으로 조인 결과를 만들지 않는 중간 조인 결과를 생성한다. 이렇게 생성된 중간 조인 결과들은 다음 조인 연산을 수행하면서 언젠가 버려질 불필요한 중간 결과를 계속해서 만들어 내게 된다. 따라서 조인 연산을 수행하는데 있어 더 많은 투플을 비교해야 하므로 연산 비용을 증가시키게 된다. 본 논문에서는 이러한 불필요한 연산을 줄이기 위해 각 조인 단계마다 최종 조인 결과를 생성하는 투플 만을 선택하여 조인 연산을 할 수 있도록 하는 방법을 제안한다.

Just-In-Time[10]에서는 각 조인 연산 단계에서 생성되는 중간 결과들 중에서 다음 조인 연산 단계에서 조인 결



(a) MJoin (b) XJoin

(그림 1) XJoin과 MJoin 동작 방식

과를 생성하지 않는 투플의 처리를 지연시켜 불필요한 조인 연산의 수행을 줄이는 방법을 제안한다. 하나의 투플에 대하여 우선 조인 연산을 수행 한 후 조인 결과를 생성하지 않으면 현재 조인의 입력 소스를 생성하는 이전 단계의 조인 연산에게 해당 투플이 현재 불필요함을 알리고 투플은 별도의 버퍼에 저장한다. 이후 해당 투플이 조인 결과를 생성하게 되면, 이전 단계의 조인 연산에게 해당 값을 갖는 투플이 연산에 필요함을 알리고, 이전 단계의 조인 연산은 해당 값을 갖는 투플의 조인 결과를 내보내게 된다. 이를 통해 불필요한 중간 결과의 생성을 지연시킴으로써 조인 연산을 효율적으로 할 수 있게 한다.

대부분의 DSMS가 질의의 색인만을 이용하는데 반해 [16]은 질의 색인과 데이터 색인을 모두 이용한다. 특히 데이터를 다차원 공간의 하나의 점으로 색인하고 공간 조인을 이용하여 질의를 수행한다. 본 논문에서 제안한 알고리즘은 데이터를 특정 영역에 대한 벡터와 행렬로 표현한다는 점에서 유사하지만 [16]이 연속질의의 선택 조건 처리를 대상하고 본 논문은 조인 연산을 대상으로 한다는 점에서 차이가 있다. 이 외에도 다중 조인을 최적화하기 위해서 스트림의 내용물을 기반으로 질의 계획을 최적화하는 방법[17]도 제시되었다. [18]은 속성선택체(ASC)를 제안하여 질의를 도메인 상의 영역으로 색인하는 방식을 소개하였다. ASC가 질의에 따라 도메인을 가변 크기의 영역으로 분할하는 반면 본 논문에서 사용하는 속성벡터구조체(AVS)는 고정 크기로 분할하는 방식이다.

3. 효율적인 다중 조인 연속질의 처리 방법

본 논문에서 제안하는 알고리즘은 배치(batch) 기반의 슬라이딩 윈도우 조인[9] 연속질의 처리를 한다. 배치 처리 방법은 데이터 스트림 소스로부터 입력되는 데이터를 일정 기간 동안 모아두었다가 한 번에 연속질의를 처리하는 방법으로써 매번 질의 처리를 수행하는 방법에 비해서 단위 시간당 더 많은 데이터에 대해서 연속질의를 수행할 수 있다는 장점이 있다.

본 논문에서 제안하는 알고리즘을 이용해 데이터 스트림에서 조인 연산을 처리하는 방법은 크게 네 가지 단계를 거친다. 첫 번째 단계는 질의에 대한 색인 생성 단계이다. 이 단계에서는 주어진 질의에 대해서 정의 1의 AVS와 정의 2의 AMS 및 조인 계획 그래프를 생성한다. 두 번째 단계는 데이터 스트림으로부터 새로운 데이터가 입력되었을 때, 첫 번째 단계에서 생성된 색인을 이용해 입력된 데이터로부터 생성될 조인 결과 투플 수를 계산하는 단계이다. 이 단계에서는 새로 입력된 데이터가 조인 연산을 통해 최종적으로 생성할 결과 투플 수와, 도메인 공간 내의 특정 구간별로 조인 결과를 생성할 수 있는 투플 수를 계산하게 된다. 세 번째 단계는 두 번째 단계에서 계산된 각 연산의 결과들을 역추적해서 최종적인 조인 결과에 참여하는 투플들 만을 선택한다. 마지막 단계는 실제로 조인 연산을 수행하여 결과 투플을 얻는 단계이다. 세 번째 단계에서 계산된 결과에서 생성할 결과 투플 수가 0으로 계산된 구간에 속한 투플들에 대해서는 조인 연산을 수행하지 않는다.

3.1 질의 색인 생성 단계

질의 색인 생성 단계에서는 주어진 연속질의 Q를 만족하는 투플들을 모니터링하고 조인 결과 투플 수를 계산하는데 필요한 두 가지 형태의 구조체를 생성한다. 정의 1과 2는 각각 두 구조체를 정의한다.

(정의 1) 속성벡터구조체 (AVS;Attribute Vector Structure)

주어진 질의 Q에 사용된 데이터 스트림의 집합을 $D=\{S_1,S_2,\cdots,S_n\}$라 하고, i번째 스트림의 k번째 속성을 $S_i.k$라 하자. $S_i.k$의 도메인 공간을 일정한 크기(δ)를 갖는 n개 영역으로 나누었을 때 나누어진 각각의 영역을 셀(cell)이라고 한다. 이 셀의 배열을 $AVS(S_i.k)$로 표기하고, 스트림 S_i의 속성 k에 대한 속성벡터구조체(AVS)라고 한다.

속성 k의 도메인 최대값을 D_{max}라고 할 때, 셀은 다음과 같은 속성을 갖는 구조체이다.

i) 셀 번호(cid) : $1 \le cid \le \left\lceil \frac{D_{max}}{\delta} \right\rceil$

ii) 셀 영역($range$) :

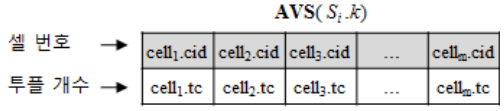
$range = [(cid-1)\times\delta,\ \min((cid\times\delta), D_{max}))$

iii) 투플 개수(tc) : 슬라이딩 윈도우 내에 있는 투플 중에서 속성 k의 값이 해당 셀의 영역에 포함되는 투플의 개수. 시간 도메인 $\mathfrak{I}$에 대해서 현재 시점을 t라 하고 배치 크기를 β라 하면 투플 개수 tc는 $t-\beta$와 t 사이의 시간에 들어온 투플 중에서 속성 k의 값이 현재 셀의 $range$에 포함되는 투플의 개수로 결정된다.

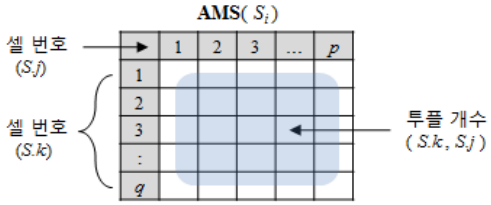
(정의 2) 속성행렬구조체 (AMS:Attribute Matrix Structure)

주어진 질의 Q에 사용된 데이터 스트림의 집합을 $D=\{S_1,S_2,\cdots,S_n\}$라 하고, i번째 스트림 S_i에 대하여 질의 Q에 사용된 속성의 집합을 $A(S_i)=\{S_i.1,S_i.2,\cdots,S_i.k,\cdots\}$라고 하자. 이때 2개 이상의 원소를 갖는 $A(S_i)$ 대하여 집합 $A(S_i)$의 원소를 각 축으로 하고 해당 속성에 대한 투플 개수를 값으로 갖는 행렬을 $AMS(S_i)$로 표기하고 속성행렬구조체(AMS)라고 한다.

(그림 2(a))와 (그림 2(b))는 각각 AVS와 AMS의 예시를 보여준다. 본 논문에서 제안한 알고리즘의 질의 색인 생성 단계에서는 첫 번째로 다중 조인이 포함되어 있는 질의 Q에 대해서 AVS와 AMS를 생성한다. Q에서 두 개 이상의 속성이 조인 조건으로 사용된 데이터 스트림에



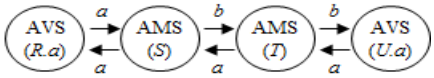
(a) AVS



(b) AMS

(그림 2) AVS 및 AMS 예시

select * from R, S, T, U
where $R.a=S.a$ **and** $S.b=T.a$ **and** $T.b=U.a$



(그림 3) 조인 계획 그래프

대해서는 AMS를 생성하고, 하나의 속성만 조인 조건으로 사용된 스트림에 대해서는 AVS를 생성한다. 다음으로 (그림 3)과 같은 조인 계획 그래프를 만든다. 우선 AVS가 만들어진 스트림에 대해서는 AVS노드를, AMS가 만들어진 스트림에 대해서는 AMS노드를 생성한다. 이후 Q에서 등호로 연결된 스트림에 대한 노드 사이에 다리를 연결한다. 다리의 방향은 등호를 기준으로 좌측에서 우측으로 방향을 갖고 좌측 노드에서 사용된 속성 식별자를 속성으로 갖는 다리가 0번 다리가 되고, 우측 노드에서 좌측 노드로 방향을 갖고 우측에 사용된 속성 식별자를 다리의 속성으로 갖는 다리가 1번 다리가 된다.

예를 들어 (그림 3)에 주어진 질의에 대한 조인 계획 그래프를 생성하는 경우를 살펴보자. 스트림 R과 스트림 U는 모두 조인 속성 a 하나 만을 사용하므로 각각 $AVS(R.a)$와 $AVS(U.a)$ 노드를 생성한다. 스트림 S와 T의 경우 모두 조인속성 a와 b 두 개가 사용되었으므로 각각 $AMS(S)$와 $AMS(T)$ 노드를 생성한다. 다음으로 첫 번째 조인 조건이 $R.a=S.a$이므로 $R\rightarrow S$의 방향을 갖고 속성으로 a를 갖는 다리와 $S\rightarrow R$의 방향을 갖고 속성으로 a를 갖는 두 개의 다리를 $AVS(R.a)$와 $AMS(S)$ 사이에 생성한다. 이어서 두 번째 조인조건 $S.b=T.a$에 대해서는 $S\rightarrow T$의 방향을 갖고 속성으로 b를 갖는 다리와 $T\rightarrow S$의 방향을 갖고 속성으로 a를 갖는 다리를 생성한다. 마지막으로 세 번째 조인조건 $T.b=U.a$에 대해서 $T\rightarrow U$의 방향과 속성으로 b를 갖는 다리와 $U\rightarrow T$의 방향을 갖고 속성으로 a를 갖는 다리를 생성한다.

3.2 조인 결과 투플 수 계산 단계

각각의 AVS와 AMS는 현재 윈도우 내에 포함되는 투플에 대해서 투플의 개수를 모니터링 한다. 스트림 S_i에 새로운 투플이 들어오면 해당 스트림을 사용하는 조인 조건에 해당하는 속성에 대해서 (1)과 같이 셀 번호 cid를 구한다.

$$cid = \left\lfloor \frac{S_i.k}{\delta} \right\rfloor \quad \cdots\cdots\cdots (1)$$

셀 번호를 구한 후에 해당 셀의 투플 개수인 tc를 증가시켜 현재 윈도우 내에 있는 투플의 개수를 AVS 및 AMS 구조체 내에 유지하도록 한다. 그리고 새로 입력되어 아직 조인 연산에 참여하지 않은 투플에 대해 배치 크기에 해당하는 투플들에 대한 AVS와 AMS를 만든다. 즉, 스트림 S_i에서 배치 크기 β 기간 동안 들어온 데이터에 대해서 S_i가 AVS 노드를 가지고 있을 경우에는 $AVS(S_i.k^{new})$를, AMS노드를 가지고 있을 경우에는 $AMS(S_i^{new})$와 조인에 사용된 각 속성에 대한 $AVS(S_i.k^{new})$를 생성하고 $AVS(S_i.k^{new})$와 $AMS(S_i^{new})$는 아직 처리되지 않은 투플들의 개수를 갖도록 한다.

다음으로 앞서 설명한 질의 색인 생성 단계에서 만든 조인 계획 그래프에 따라 조인 결과 투플 수를 계산한다. 현재 처리해야 할 스트림이 S_i인 경우 $AVS(S_i.k^{new})$의 각 셀에 저장된 투플 개수를 값으로 갖는 벡터와 $AMS(S_{i+1})$의 행렬과의 벡터의 내적을 구하면, $AVS(S_i.k^{new})$와 $AVS(S_{i+1})$의 세로축 속성에 대한 조인 결과 투플 수를 $AMS(S_{i+1})$의 가로축 속성에 대한 벡터로 얻을 수 있다. 예를 들어 $S_1.a=S_2.a \wedge S_2.b=S_3.a$와 같은 연속 조인 질의를 수행 한다고 할 때 (2)에 따라 $AVS(S_1.k^{new})\cdot AVS(S_2)$ 연산 결과는 $S_1.a=S_2.a$ 질의 결과의 $S_2.b$에 대한 벡터이다. 다음 연산이 AMS와의 연산(즉 행렬과의 연산)일 때는 내적을 구하고 AVS와의 연산(즉 벡터와 벡터의 연산)인 경우에는 정의 3에서 정의한 벡터-원소곱 연산자 '⊙'를 사용한다.

새로 입력된 데이터가 있는 스트림과 관련된 노드에 대해서는 $AVS(S_i.k^{new})$ 또는 $AMS(S_i^{new})$를 해당 노드 대신에 치환해서 계산하도록 한다. 치환식을 만드는 방식은 AVS를 치환하는 경우와 AMS를 치환하는 경우의 두 가지로 나누어진다. AVS를 치환하는 경우 간단히 $AVS(S_i.k)$를 $AVS(S_i.k^{new})$로 치환해서 (3)과 같이 계산하면 된다. AMS를 치환하는 경우 즉 두 개 이상의 상대 스트림과 조인되어야 하는 스트림에 새로운 투플이 들어왔을 경우 새로 들어온 투플에 대해서 두 개의 AVS 구조체를 생성한다. 예를 들어 $AMS(S_i^{new})$가 생성 되었을 때 조인 결과 투플 수 계산을 위해서 $AVS(S_i.k^{new})$와 $AVS(S_i.j^{new})$의 두 개의 AVS를 생성한다. 이때 (4)와 같이 $S_{i+1}.k$와 조인되는 $S_i.j^{new}$는 (3)을 이용하여 계산하고, 스트림 $S_{i-1}.j$과 조인되는 $S_i.k^{new}$는 스트림 S_{i-1}이 AMS인 경우 $AMS(S_{i-1})$의 전치 행렬인 $AMS(S_{i-1})^T$를 사용해 연산하도록 한다. 이는 $AMS(S_{i-1})$이 속성 k의 행과 j의 열로 구성된 행렬이므로 $S_i.k$와 조인되는 $S_{i-1}.j$에 결과를 $S_{i-1}.k$에 대해 정렬된 형태의 벡터로 알 수 있어야 하기 때문이다.

n개의 아이템을 원소로 갖는 벡터와 n행 m열의 행렬과의 내적은 항상 1행 n열의 벡터로 표현되므로 정의 4에 주어진 모든 연산은 벡터와 행렬의 내적 또는 벡터와 벡터의 벡터-원소곱 연산으로 이루어진다.

(정의 3) 벡터-원소곱 연산자(⊙)

R_n 공간의 두 벡터를 $A=(a_1,a_2,\cdots,a_n)$, $B=(b_1,b_2,\cdots,b_n)$라 할 때, A와 B에 대한 벡터-원소곱을 $A\odot B=(a_1\times\times b_1,a_2\times b_2,\cdots,a_n\times b_n)$라고 정의한다.

(정의 4) 조인 결과 투플 수 계산

a. 기본식

$$AVS(S_1.k)\cdot AMS(S_2)\cdot AMS(S_3)\cdots \cdot AMS(S_{n-1})\odot AVS(S_n.j) \quad (2)$$

b. AVS 치환식

$$AVS(S_i.k^{new})\cdot AMS(S_{i+1})\cdot AMS(S_{i+2})\cdots \cdot AMS(S_{n-1})\odot AVS(S_n.l) \quad (3)$$

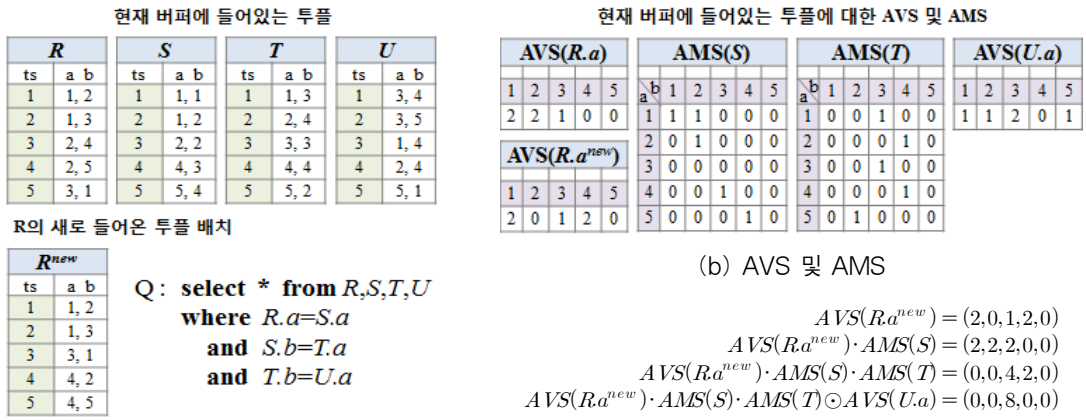
c. AMS 치환식: 스트림 S_i에 새로운 데이터가 입력된 경우

$$AVS(S_i.l^{new})\cdot AMS(S_{i+1})\cdot AMS(S_{i+2})\cdots \cdot AMS(S_{n-1})\odot AVS(S_n.k) \quad (4)$$

$$AVS(S_i.k^{new})\cdot AMS(S_{i-1})^T\cdot AMS(S_{i-2})^T \cdots \odot AVS(S_1.k)$$

정의 4에서 각 연산의 결과로 얻은 벡터는 새로 들어온 투플의 배치로부터 생성되는 조인 결과 투플 수를 의미한다. 따라서 이를 통해 실제로 조인 연산을 수행하지 않고 현재 새로 들어온 투플로부터 생성되는 조인 결과의 수를 각 도메인 상에서 나누어진 셀에 해당하는 영역별로 알 수 있다. 이는 실제로 조인의 결과가 중요하지 않거나 필요 없는 모니터링 질의와 같은 경우 이 값만을 이용하여 조인 여부를 판단하여 해당 투플의 조인 결과가 있음을 알릴 수 있다.

지금까지 설명한 조인 결과 투플 수 계산 단계를 예를 들어 설명한다. (그림 4(a))에 질의 Q가 주어졌다. 각각의 데이터 스트림 R, S, T, U에 현재 그림 4(a)와 같은 투플들이 들어와서 버퍼에 저장되어 있고 스트림 R에 새로



(그림 4) 조인 결과 튜플 수 계산 단계 예제

들어온 투플이 R^{new}로 표현되어 있다. 이때 Q에 대해서 생성된 AVS 및 AMS와 현재 버퍼에 들어와 있는 데이터에 대한 모니터링 값은 (그림 4(b))와 같다. 조인 결과 투플 수 계산을 위해서 정의 4의 수식을 사용하면 각 연산 단계의 결과는 (그림 4(c))와 같다. 첫 번째 $AVS(R.a^{new})$는 스트림 R에 새로 들어온 투플의 배치 크기 만큼의 데이터의 조인 속성 a에 대한 셀별 투플 개수이다. 두 번째 $AVS(R.a^{new})\cdot AMS(S)$의 결과는 질의 $R.a=S.a$의 결과에 대해서 $S.b$의 값으로 정렬된 AVS 결과이다. 즉 $R.a=S.a$의 결과는 $S.b$가 1인 투플이 2개, 2인 투플이 2개 3인 투플이 2개 임을 계산을 통해서 알아낸 것이다. 세 번째 결과 (0,0,4,2,0)은 질의 $R.a=S.a\wedge S.b=T.a$의 결과 투플 수가 $T.b$에 대해서 표현된 것이고 마지막 연산 결과는 질의 $R.a=S.a\wedge S.b=T.a\wedge T.b=U.a$의 조인 결과 투플 수가 $U.a$에 대해서 표현된 것이다.

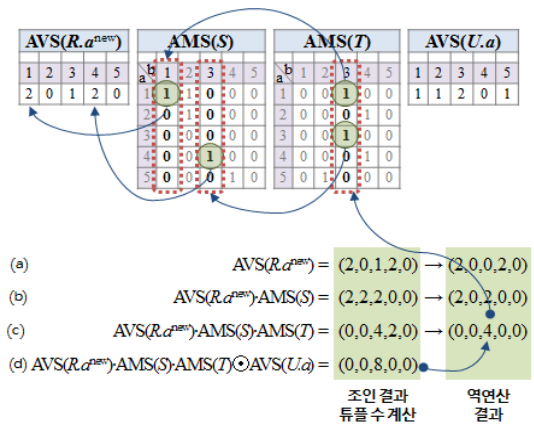
지금까지의 연산을 통해서는 조인 연산의 중간 및 최종 결과에 대한 투플 수 만을 알 수 있으며, 어떤 투플들이 최종 조인 결과를 생성하는지를 정확하게 알 수는 없다. 다음 절에서 소개하는 역연산 단계에서는 모든 스트림에 대해서 정확하게 조인 결과 투플 수를 계산함과 동시에 조인 연산 과정에서 반드시 필요한 투플들 만을 선택하기 위한 연산 수행 방법을 소개하도록 하겠다.

3.3 역연산 단계

역연산 단계는 3.2절에서 구해진 결과 벡터들을 역추적하여 실제 조인 결과를 생성하게 될 투플을 선택하는 단계이다. 수학적으로 n개의 원소를 갖는 벡터와의 내적을 구할 수 있는 행렬은 n개의 행으로 이루어져야 한다. 즉 n개의 원소를 갖는 벡터 V와 n개의 행과 m개의 열을 갖는 행렬 M의 내적의 결과가 O라고 했을 때 결과 벡터 O의 i번째 원소 $O(i)$의 값은 M의 i번째 열의 각 원소와 V의 각 원소의 곱의 합이다. 따라서 $O(i)$의 값이 0이면 M의 i번째 행의 값에 관계없이 V의 어떠한 원소와도 곱해지는 값이 없음을 알 수 있다. 따라서 결과 벡터 O의 값을 결정하는 V의 원소는 $O(i)\neq 0$을 만족하는 i에 대한 M의 i번째 행들 중에 0이 아닌 값을 갖는 열들이다. 이렇게 0이 아닌 값을 갖는 열들을 찾으면 이 열과 곱해지는 $V(i)$를 제외한 다른 V의 원소들은 어떠한 값을 갖더라도 O의 값에 영향을 주지 않는다는 것을 알 수 있으며 이 성질을 이용해서 역연산 단계를 수행한다.

(그림 4(a))에 주어진 질의 Q에 대해서 스트림 R에 새로운 투플이 들어온 경우 (그림 5)와 같은 AVS 및 AMS가 생성된다. 이에 대한 조인 투플 수 계산 결과는 (그림 5)에 주어진 것과 같다. 이때 (그림 5(d))의 결과 (0,0,8,0,0)은 (그림 5(c))의 결과와 $AVS(U.a)$의 ⊙연산 결과이므로 (0,0,4,2,0)에서 네 번째 원소에 대응되는 셀에 포함되는 투플은 스트림 U와 조인되지 않고 세 번째 셀에 포함되는 투플만이 스트림 U와의 조인 결과를 만드는 것임을 알 수 있다. 따라서 (그림 4(c))의 결과 벡터에서 실제로 조인에 참여하는 투플을 표현한 벡터는 (0,0,4,0,0)라고 할 수 있다. 이렇게 얻어진 결과 (0,0,4,0,0)를 (그림 5(c))의 역연산 결과라고 정의하며 $R(T.b)$와 같이 표기한다.

이때 (그림 5(c))의 결과는 $AVS(R.a^{new})\cdot AMS(S)$의 결



(그림 5) 역연산 결과

과 벡터와 $AMS(T)$의 내적이므로 (그림 5(c))의 결과에서 3번째 원소의 값을 결정하는 것은 $AMS(T)$의 세 번째 행이다. (그림 5(c))의 역연산 결과를 통해 스트림 U와 조인에 성공할 수 있는 투플은 세 번째 원소뿐이므로 (그림 5(b))의 결과와 $AMS(T)$의 세 번째 결과의 연산만이 의미있는 연산임을 알 수 있다. 따라서 $AMS(T)$의 3열을 살펴보면 1행과 3행만이 연산의 결과를 만들고 이는 (그림 5(b))의 결과 벡터에서 첫 번째 원소와 세 번째 원소만이 조인에 참여하는 투플들을 포함하는 셀임을 나타낸다. 따라서 (그림 5(b))의 역연산 결과는 (2,0,2,0,0)이 된다. 계속해서 스트림 $AMS(S)$에서 연산의 결과를 생성하는 행은 1행과 3행임을 알 수 있다. $AMS(S)$의 1,3행에서 값을 갖는 열은 1열과 4열이므로 이를 통해 $AVS(R.a^{new})$의 벡터 중에서 1,4번째 원소만이 실제로 조인 연산의 결과를 만드는 값임을 알 수 있다. 역연산 단계를 통해 $AVS(R.a^{new})$에 속한 투플들, 즉 스트림 R에 새로 들어온 투플들 중에서 조인 연산의 결과를 만드는 투플은 $R.a$의 값이 AVS의 첫 번째와 네 번째 셀의 영역에 속하는 투플들임을 알 수 있다.

3.4 조인 연산의 수행 단계

마지막 단계는 실제로 조인을 수행하여 결과를 얻는 단계이다. 앞서 조인 결과 투플 수 계산과 역연산 단계를 거쳐서 실제로 조인이 이루어 질 수 있는 영역을 찾았다. 이제 실제로 각각의 투플에 대해서 조인을 수행하기 앞서 해당 투플의 조인에 사용될 속성이 앞서 계산한 역연산 결과에서 조인이 가능한 셀에 포함되는지 여부를 검사한다. 검사 결과 조인이 가능한 셀에 해당하는 값을 갖는 투플이면 상대 스트림의 윈도우 안에 있는 투플과 조인이 되는 투플을 찾는 과정을 수행한다. 또는 검사 결과 조인이 가능한 셀에 해당하는 값이 아니면 조인 연산을 수행하지 않고 바로 해당 스트림의 윈도우 내에 삽입하는 과정을 수행하게 된다. 이를 통해 조인되지 않을 투플은 처음부터 조인 연산을 수행하지 않음으로써 불필요한 조인 연산을 줄일 수 있다. 또한 다중 조인 연산의 수행 중에 버려지게 되는 중간 결과를 처음부터 생산하지 않음으로써 조인 연산의 비용을 감소시킬 수 있다.

이때 투플의 조인 가능 여부를 검사하는 방법은 해당 스트림이 갖는 구조체가 AVS이거나 또는 AMS인 경우에 따라서 다르다. 해당 스트림이 AVS를 갖고 있는 경우 (5)와 같이 AVS내의 해당 셀을 찾는다.

$$R(S_i.k) > 0 \wedge R(S_i.j) > 0\text{인 } S_i\text{의 투플 } d \quad \cdots(5)$$

해당 셀에 대응하는 원소 값이 1 이상인 경우 조인 가능한 투플로 판단하고 조인을 수행한다. 해당 스트림이 AMS를 갖고 있는 경우 (4)의 결과에 대한 역연산 결과 $S_i.k$와 $S_i.j$에 대한 두 개의 벡터를 검사해야 한다. 이때 S_i의 투플은 (5)를 만족하는 경우에 조인 가능한 투플로 판단한다.

예를 들어 (그림 5)의 경우 스트림 R에 새로 들어온 투플들 중에서 $R.a$ 값에 대응하는 셀 번호가 1이거나 4인 투플 만을 대상으로 조인 연산을 수행한다. 이 결과 생성된 $R.a = S.a$의 결과 중에서 다시 $S.b$ 값에 대응하는 셀 번호가 1이거나 3인 투플 만을 다음 조인 연산의 대상 투플로 사용한다. 이 결과인 $R.a = S.a \wedge S.b = T.a$의 결과 중에서 $T.b$ 값에 대응하는 셀 번호가 3인 투플만을 다음 조인 연산의 대상으로 참여시킨다. 이를 통해 나온 결과는 정확히 모든 조인 조건을 만족하는 투플 들이며 매 단계마다 생성되는 조인 결과들 중에서 최종 조인 결과가 없을 것으로 예상되는 투플을 미리 알고 조인 연산에서 제거함으로써 조인 연산의 속도를 빠르게 할 수 있다.

4. 조인 결과 투플 수 계산 구현 최적화

이 장에서는 3.4절에서 설명한 조인 결과 투플 수 계산식을 컴퓨터 프로그램으로 구현하는데 있어 연산 속도를 빠르게 하는 방법을 소개한다. 3.4절에 소개된 연산은 1차원 벡터와 n차원 행렬의 내적 연산을 수행해야 하므로 셀의 개수가 많아지면 연산 양은 $O(n^2)$의 형태로 증가하게 된다. n개의 원소를 갖는 벡터 V와, $n \times m$ 행렬

의 연산은 $n^2 \times m$번의 곱셈과 n번의 덧셈으로 이루어지기 때문이다. 따라서 이 연산 비용을 줄이면 전체 연산 시간을 절약 할 수 있다.

이와 함께 전처리 연산의 최적화 방법을 제안한다. 대부분의 경우 조인 결과 투플 수를 정확히 알 필요가 없고 단지 조인의 성공 여부만이 필요하기 때문에 벡터와 행렬의 구조를 비트 형태로 바꾸고 논리 연산을 이용하는 방법이다. 연산 과정에서 결과 투플의 개수를 계산할 필요가 없으므로 내적 연산에서 각 원소곱의 덧셈 연산을 생략하고 32비트 단위의 논리곱 연산 중에 하나라도 1이상의 값이 나오면 바로 결과가 존재함을 알 수 있다.

비트 연산을 위해서 AVS는 각 셀의 튜플 개수 tc대신에 단지 해당 셀의 데이터 존재 유무만을 표현하는 비트를 갖는다. AMS는 각 열과 행에 대해서 셀의 데이터 존재 유무를 표현하는 비트를 갖는다. 질의 색인 생성 단계 수행 후 조인 결과 투플 수 계산 단계에서 스트림 S_i에 새로운 투플이 들어왔을 때 (1)에 의해 구해진 셀의 투플 개수를 증가시키는 대신에 해당 셀 번호 번째의 비트를 1로 설정한다. 이 결과 배치 시간 β동안 들어온 투플에 대한 모니터링 값은 해당 셀의 투플의 존재 유무만을 표시하는 비트열이 된다. 투플이 있는 셀을 표현하는 비트는 1로 투플이 없는 셀을 표현하는 비트는 0으로 설정 된다. 따라서 이후의 모든 연산은 조인 결과 투플 수를 계산하는 것이 아니라 조인 결과 여부를 나타내는 비트열을 구하기 위한 연산으로 바뀌게 된다. 비트 연산을 이용한 최적화 방법에서 정의 4의 벡터와 행렬의 모든 내적 연산은 벡터의 비트열과 행렬의 각 비트열에 대한 논리곱(and) 연산의 결과로 치환한다. 벡터-원소곱 연산 역시 두 벡터의 비트열에 대한 논리곱 연산으로 치환한다.

5. 성능 평가 및 실험

본 장에서는 이전에 제안한 방법을 여러 가지 실험을 통하여 성능을 비교하였다 실험 환경은 Intel Core2 Duo CPU(2.4GHz), 메모리 2G, Fedora Core5를 사용하였으며 알고리즘 구현 언어로 C를 사용하였다. (표 1)은 본 실험에서 사용한 변수이다.

도메인 크기 D_{max}는 범위 $[1, D_{max}]$에 해당하는 균등 분포(Uniform Distribution) 형태의 정수를 랜덤하게 생성하였다. 직관적으로 D_{max}의 값이 커지면 조인 연산에서 조인 성공률이 낮아지고 D_{max}의 값이 작아지면 조인 성공률이 높아짐을 알 수 있다. 따라서 D_{max}의 값으로 데

(표 1) 실험 변수

변 수	값
Domain Size D_{max}	100, 200, 300, 400, **500**
Cell Count C_{cnt}	**100**, 200, 300, 400, 500
Join Count J_{cnt}	3, **4**, 5, 6, 7
Batch Size B_{size}(sec)	**20**
Window Size W(sec)	10, 20, 30, 40, **50**, 60, 70

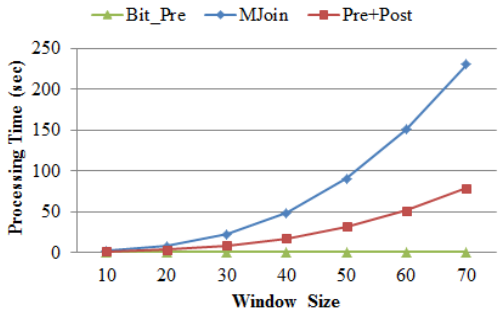
이터의 조인 성공률 값을 조절할 수 있다. 조인 수 J_{cnt}는 하나의 연속 질의에 있는 조인 조건의 개수를 의미하며, 모든 실험에 사용된 질의는 N개의 스트림에 대해서 $N-1$개의 동등 조인 조건이 있는 연속질의를 수행하였다. 예를 들어 4개의 스트림 R,S,T,U의 경우 조인 조건은 $(R.x_1 = S.x_1) \wedge (S.x_2 = T.x_1) \wedge (T.x_2 = U.x_1)$와 같다. 셀의 수 C_{cnt}는 D_{max}를 δ로 나눈 값이며 이는 전처리 연산의 속도에 영향을 미치는 변수이다. 또한 모든 조인은 윈도우 크기 W를 갖는 슬라이딩 윈도우 조인을 사용하였다. 실험에 사용한 변수 인자들은 표 1에 주어진 값들을 변화시키면서 비교 사용하였으며, 굵은 글씨체로 표현된 값을 기본값으로 사용하였다.

모든 실험의 Y축은 각 스트림별로 200,000개의 데이터가 입력으로 들어왔을 때 이를 모두 처리 하는데 걸리는 수행 시간이다. 비교 알고리즘은 MJoin을 사용하였으며, 본 논문에서 제안한 알고리즘은 전처리를 통하여 투플을 걸러낸 후에 MJoin을 통하여 실제 조인을 수행하였다.

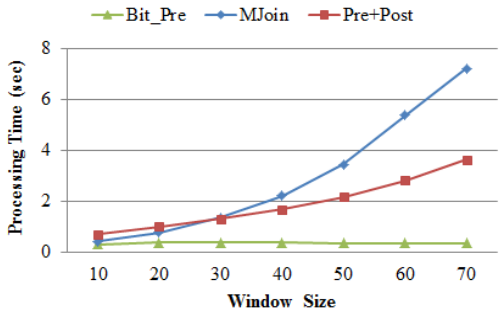
5.1 윈도우 크기 변화에 대한 비교

(그림 6(a))와 (그림 6(b))는 각각 도메인 크기 D_{max}가 100, 500인 경우에 대해서 윈도우 크기를 다르게 하여 실험한 것이다. C_{cnt}는 100으로 고정이고 투플의 값은 정수이므로 D_{max}가 100일 때는 AMS 와 AVS에서 각각의 셀이 도메인 상의 하나의 값 만을 대표 하는 경우이다. 이 경우 조인 결과 투플 수 계산과 역연산 계산의 결과를 통해 정확히 각 투플의 조인 참여 여부를 판단 할 수 있다. D_{max}가 500인 경우 $\delta = 5$가 되므로 하나의 셀은 5의 크기를 갖게 되고 전처리의 필터링 적중도는 떨어지지만 전처리 시간이 증가하지 않는다.

실험 결과 윈도우 크기가 커짐에 따라 본 논문에서 제안한 알고리즘을 적용하여 MJoin을 수행한 Pre+Post의 수행 시간과 단지 MJoin만을 수행한 수행 시간의 차이가 점차 커짐을 알 수 있다. 이는 윈도우 크기가 커지면 조

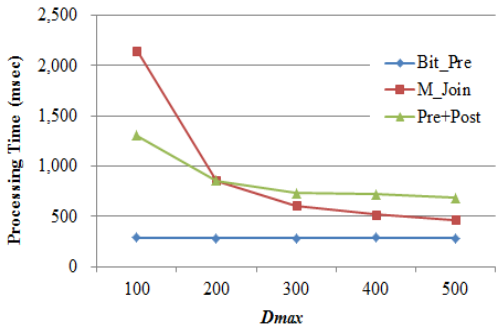


(a) $D_{max}=100$

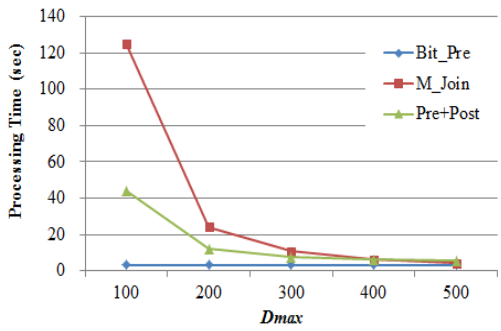


(b) $D_{max}=500$

(그림 6) 윈도 크기 변화 비교



(a) $W=10$



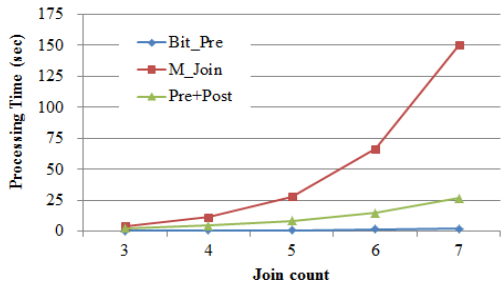
(b) $W=50$

(그림 7) D_{max} 값의 변화에 따른 연산 속도 비교

인의 결과가 많이 생기게 되어 조인 연산의 중간 과정에서 최종적으로 결과를 만들지 않는 투플들 역시 많이 생성되기 때문이다. 따라서 역연산 결과를 통해 필터링 되는 투플의 수도 더 많아지게 되므로 위와 같은 결과를 얻을 수 있다. 다만 D_{max}가 500일 때 윈도우 크기가 20 이하에서 제안한 알고리즘이 좋지 않은 결과를 보이는 것은 실험 조건에서 D_{max}로 인해 결정된 조인 성공률이 D_{max}가 100일 때 보다 작아져서 조인 비용이 전체적으로 감소하였지만 전처리 시간(Bit_Pre)은 변하지 않았기 때문에 상대적으로 전처리 시간이 커졌기 때문이다.

5.2 도메인 크기 변화와 조인 수 변화 비교

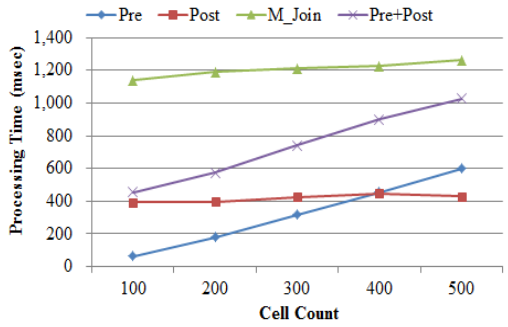
(그림 7(a))와 (그림 7(b))는 D_{max}가 작을수록 제안한 알고리즘을 적용한 경우 성능이 좋아지는 것을 보여준다. 다만 윈도우 크기가 매우 작을 경우 D_{max}의 값이 커



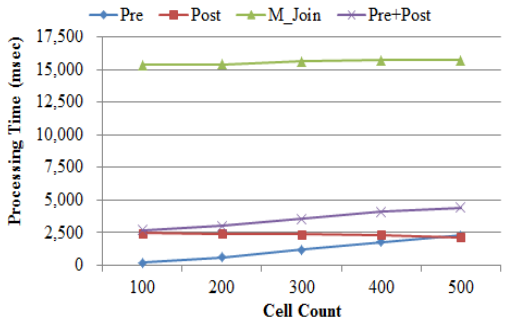
(그림 8) J_{cnt} 값의 변화에 따른 연산 속도 비교

져서 조인 성공률이 낮아지면 낮은 성능을 보임을 알 수가 있다.

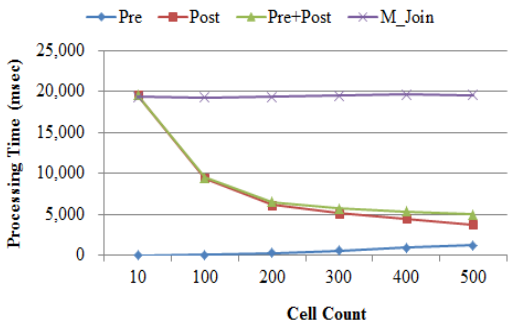
(그림 8)의 결과 조인 개수 즉 스트림 수가 증가함에 따라 제안한 알고리즘을 적용한 경우의 성능이 매우 좋아짐을 알 수 있다. 조인 결과 투플 수 계산과 역연산 등



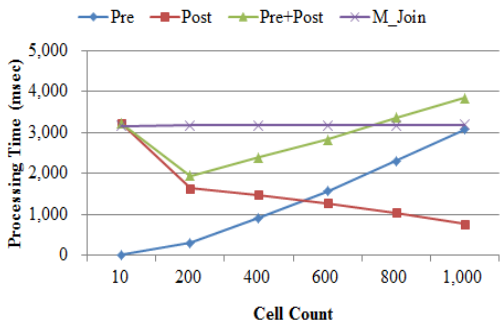
(a) $J_{cnt}=4$



(b) $J_{cnt}=7$



(c) $D_{max}=500$



(d) $D_{max}=1,000$

(그림 9) C_{cnt} 값의 변화에 따른 연산 속도 비교

의 전처리 과정의 비용은 조인 수가 증가함에 따라 매우 적은 양으로 증가하는 반면에 단순 조인 비용은 매우 큰 폭으로 증가 한다. 단순 조인 비용이 큰 폭으로 증가하는 이유는 조인에 참여하는 스트림의 수가 많아지면서 그만큼 비교해야 하는 투플의 수도 많아지기 때문이다. 그러나 제안한 알고리즘을 이용해 조인이 가능하지 않은 투플을 필터링 하게 되면 조인 횟수가 많아지는 만큼 필터링을 통해 절약되는 비용도 커지기 때문에 비용의 증가 폭이 크지 않음을 알 수 있다.

5.3 셀 개수 변화에 대한 비교

앞서의 실험을 살펴보면 조인 수가 증가하는 경우를 제외한 모든 경우에서 전처리 시간은 일정함을 알 수 있다. 이는 전처리 시간이 D_{max}나 윈도우 크기 W와 무관함을 알 수 있다. (그림 9(a),(b))의 실험을 통해 전처리 시간에 가장 큰 영향을 주는 변수는 C_{cnt}로서 C_{cnt}가 증가함에 따라 전처리 시간(Pre)이 단순 증가함을 알 수 있다. C_{cnt}만을 증가시켰으므로 M_Join과 전처리를 적용시킨 Pre+Post의 변화는 거의 없다. C_{cnt}가 약 400 근처에서 전처리 시간이 알고리즘 적용 후 조인 시간보다 더 걸리는 것을 볼 수 있는데 이는 앞서의 실험들과 마찬 가지로 δ가 작아져서 셀의 개수가 많아지면 전처리 시간이 증가하기 때문이다. 그러나 (그림 9(b))와 같이 조인 개수가 많은 경우에는 전처리 비용의 증가를 충분히 감당할 만큼 제안한 알고리즘의 성능이 좋음을 알 수 있다. 특히 주어진 실험에서 D_{max} 값이 500이므로 C_{cnt}의 최대값 역시 500이다. 이것은 현재 그래프에서 C_{cnt}를 더 이상 증가시킬 수 없음을 의미 한다.

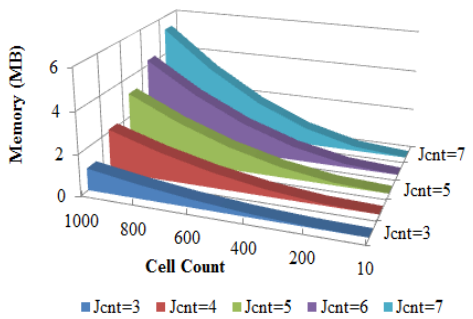
(그림 9(c),(d))는 스트림에 100,000개의 데이터가 들어왔을 때 모든 데이터를 처리 하는데 걸린 시간 결과로써, 역시 C_{cnt}가 증가함에 따라 전처리 시간이 증가한다. 전처리 시간 외에 M_Join과 전처리를 적용한 Pre+Post의 비용은 C_{cnt}의 변화와 관계가 없으며 C_{cnt}가 약 500근처에서 전처리 비용이 후처리(Post) 비용과 같아지지만 C_{cnt}를 더 이상 크게 하는 것은 앞서 설명한 바와 같이 아무

런 의미가 없고, 제안한 알고리즘을 적용하지 않은 M_Join 비용과 큰 차이를 보이기 때문에 실험 결과는 좋은 형태를 보여준다고 할 수 있다.

(그림 9(a))는 도메인 크기가 500인 경우 C_{cnt} 증가에 따른 연산 속도를 비교한 것이다. C_{cnt}가 증가함에 따라 전처리 시간은 늘어나고 전처리를 적용한 Pre+Post의 시간은 감소하는 것을 볼 수 있다. 이는 도메인 크기 대비 C_{cnt}의 크기가 커지면서 전처리의 적중률이 높아지기 때문이다. C_{cnt} 값이 10일 때는 전처리 비용이 매우 작지만 전처리의 효과가 거의 없어 M_Join과 같은 시간이 걸리는 것을 알 수 있다. 전처리를 적용한 Pre+Post 비용이 전처리 비용에 비해 매우 크기 때문에 전처리 비용의 증가가 상대적으로 완만하게 이루어진다. (그림 9(b))는 D_{max}가 1,000인 경우이다. C_{cnt}의 증가에 따라 전처리 비용은 증가하고 전처리를 적용한 Pre+Post 비용은 감소하는 것은 (그림 9(c))와 같지만 C_{cnt}가 200인 지점을 지나면 후처리 비용의 감소에 비해 전처리 비용의 증가가 커져서 전체 조인 비용이 점차 커짐을 알 수 있다. 특히 C_{cnt}가 800 이상에서는 M_Join의 조인 비용보다도 커지게 된다. 즉 C_{cnt}를 크게 하여 전처리의 적중률을 높이더라도 전처리를 적용한 Pre+Post의 비용 감소폭이 점차 둔화되므로 어느 정도 이상으로 C_{cnt}를 키우는 것은 오히려 전처리의 부하를 가중시켜 효율성이 떨어짐을 알 수 있다.

5.5 Memory 사용량

(그림 10)은 본 논문에서 제안한 알고리즘이 사용하는 메모리 사용량을 J_{cnt}와 C_{cnt}에 대해서 비교한 것이다. 셀 개수가 증가할수록 메모리 사용량이 증가하는 것을 볼 수 있다. 특히 조인 수가 증가할수록, 셀 개수 증가에 따른 메모리 사용량의 증가율이 커지는 것을 알 수 있다. 이는 전처리를 위해 항상 2개의 AVS와 조인수-1개의 AMS가 필요하기 때문이다. AMS는 행렬 구조이기 때문에 AMS가 하나 늘어날수록 셀 개수 제곱의 n배로 메모리 사용량이 증가하게 된다. 메모리 사용량은 (6)과 같은 수식으로도 구할 수 있다.



(그림 10) 전체 메모리 사용량

$$Mem_{AMS} = n \times {C_{cnt}}^2 + \frac{2 \cdot {C_{cnt}}^2}{32} \quad \cdots\cdots (6)$$

$$Mem_{AVS} = {C_{cnt}}^2 + \frac{{C_{cnt}}^2}{32}$$

6. 결 론

지금까지 데이터 스트림 환경에서 연속질의의 다중 조인 연산을 처리 하는데 있어 투플의 모니터링과 조인 결과 투플 수를 미리 계산하는 방법을 통해 조인 연산을 최적화하는 방법을 알아보았다. 본 논문에서 제안한 알고리즘은 미리 계산된 값을 이용해 조인의 불필요한 중간 결과 생성을 최소화 하는 방법을 제시하여 가장 효율적인 조인 연산을 수행하도록 한다. 또한 역연산 결과는 최종 조인 결과에 참여 하는 투플들을 각각의 조인 연산마다 선택할 수 있도록 함으로써 그 효율성을 더했다. 특히 4장에서 제안한 비트 연산을 이용한 연산 최적화 방법을 이용하면 벡터와 행렬 연산의 비용을 감소시켜 빠른 속도로 조인 가능 투플을 선택 할 수 있도록 하였다.

조인 결과 투플 수 계산에 걸리는 시간은 오직 셀의 수 C_{cnt}와 조인 개수 J_{cnt}에 의해서만 영향을 받기 때문에 전처리 시간을 두 가지 변수의 조절만으로 효과적으로 제어할 수 있으며, 특히 조인 개수가 많아지는 경우 전처리 시간의 증가보다 조인 비용의 증가가 더욱 크기 때문에 실질적으로 C_{cnt}만으로 전처리 시간을 조절 할 수 있다고 하겠다.

실제 조인 결과가 필요없는 모니터링 질의의 경우 조인을 수행하지 않고 전처리 결과만을 가지고 조인 여부를 판단하여 원하는 결과를 얻을 수 있다. 또한 제한하는 전처리 알고리즘은 실험 결과 조인에 참여하는 스트림의 수가 많아지고 윈도우 크기 또는 도메인 크기의 증가로 조인 성공률이 커질수록, 즉 조인의 결과가 많을수록 좋은 성능을 보여주는데, 이는 시스템에 많은 부하가 걸리는 상황에서 더 좋은 성능을 보여 줌으로 그 활용도가 높다고 하겠다.

참 고 문 헌

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

[2] B. Babcock, S. Babu, M. Datar, R. Motwani, “Chain: Operator Scheduling for Memory Minimization in Data Stream Systems″, SIGMOD ’03, June 2003.

[3] The STREAM groups, “STREAM: The Stanford Stream Data Manager (short overview paper)″, IEEE Data Engineering Bulletin, March 2003.

[4] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. Madden, V. Raman, V. Reiss, and M. Shah, “TelegraphCQ: Continuous Dataflow Processing for an Uncertain World”, CIDR ’03, Jan. 2003.

[5] J.-H. Hwang, U. Cetintemel, and Stan Zdonik, “Fast and Highly-available Stream Processing over Wide Area Networks”, ICDE ’08, April 2008

[6] S. Babu and J. Widom, “Continuous Queries over Data Streams″, SIGMOD Record, vol.30, No.3, pp.109-120, 2001.

[7] S. Madden, M. Shah, J. M. Hellerstein, and V. Raman, “Continuously Adaptive Continuous Queries over Streams″, SIGMOD ’02, June 2002.

[8] R. Avnur and J. M. Hellerstein. “Eddies: Continuously Adaptive Query Processing”, SIGMOD Record, vol.29, No.2, pp.261-272. 2000.

[9] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom, “Models and Issue in Data Stream Systems″, PODS ’02, June 2002.

[10] Y. Yang, D. Papadias, “Just-In Time Processing of Continuous Queries”, ICDE ’08, April 2008.

[11] Y. Zhu, E. A. Rundensteiner, and G. T. Heineman, “Dynamic Plan Migration for Continuous Queries over Data Streams″, SIGMOD ’04, June 2004.

[12] S. Viglas, J. F. Naughton, and J. Burger, “Maximizing the Output Rate of Multi-way Join Queries over Streaming Information Sources”, VLDB ’02, pp.285-296, Aug. 2003.

[13] S. Babu, K. Munagala, J. Widom, and R. Motwani, “Adaptive Caching for Continuous Queries”, ICDE ’05, pp.118-129, April 2005.

[14] L. Golab and M. T. Ozsu, “Processing Sliding Window Multi-Joins in Continuous Queries over Data Streams″, VLDB ’03, Sep. 2003.

[15] T. Urhan and M. J. Franklin. “XJoin: A Reactively-Scheduled Pipelined Join Operator”, IEEE Data Engineering Bulletin, vol.23, No.2, pp.27-33, June 2000.

[16] H.-S. Lim, J.-G. Lee, M.-J. Lee, K.-Y. Whang, “Continuous Query Processing in Data Streams using Duality of Data and Queries″, SIGMOD ’06, June 2006.

[17] P. Bizarro, S. Babu, D. DeWitt, and J. Widom, “Content-based Routing: Different Plans for Different Data″, VLDB ’05, pp.757-768. Aug. 2005.

[18] H.-H. Lee, E.-W. Yun, W.-S. Lee, “Attribute-based Evaluation of Multiple Continuous Queries for Filtering Incoming Tuples of a Data Stream″, Information Sciences, vol.178, No.11 pp.2416-2432, June 2008

◐ 저 자 소 개 ◑

서 기 언

2007년 동국대학교 정보통신공학과 졸업(학사)
2009년 연세대학교 대학원 컴퓨터과학과 졸업(석사)
2009년~현재 삼성전자 DMC 연구소 근무
관심분야 : 데이터베이스, 센서 데이터 스트림 처리
E-mail : skybuilder9@gmail.com

이 주 일

2003년 홍익대학교 컴퓨터공학과 졸업(학사)
2006년 홍익대학교 대학원 컴퓨터공학과 졸업(석사)
2009년~현재 연세대학교 대학원 컴퓨터과학과 재학(박사과정)
관심분야 : 데이터베이스, 센서 데이터 스트림 처리, 데이터 스트림 마이닝, 빅 데이터 분석
E-mail : tad@database.yonsei.ac.kr

이 원 석

1985년 Boston University 컴퓨터과학과 졸업(학사)
1987년 Purdue University 대학원 컴퓨터과학과 졸업(석사)
1990년 Purdue University 대학원 컴퓨터과학과 졸업(박사)
2004년~현재 연세대학교 컴퓨터과학과 정교수
관심분야 : 센서 데이터 스트림 처리, 데이터 스트림 마이닝, OLAP / 데이터 웨어하우스
E-mail : leewo@database.yonsei.ac.kr