제 26 회 한국정보처리학회 추계학술발표대회 논문집 제 13 권 제 2 호 (2006. 11)

스트림 데이터 환경에서의 효율적인 필터 연산자 순서화

민준기
*한국기술교육대학교 인터넷 미디어 공학부
e-mail : jkmin@kut.ac.kr

Efficient Filter Operator Ordering On Stream Data Environments

Jun-Ki Min
*Korea University of Technology and Education

요 약

인터넷과 인트라넷의 확산에 따라, 스트림 데이터 처리 (stream data processing) 와 같은 새로운 분야가 등장하게 되었다. 스트림 데이터의 특징은 실 시간적이고 연속적으로 생성된다는 것이다. 따라서 기존의 질의 처리와는 달리 질의 또한 연속적으로 처리된다. 본 논문에서는 시간에 따라서 예측할 수 없게 특성이 바뀌는 데이터 스트림에 대한 처리에 대하여 다룬다. 특별히, 본 논문에서는 스트림 데이터에 대한 질의문을 구성하는 연산자들 간의 효율적인 수행 순서 생성 기법을 제안한다. 본 논문에서 제안하는 방법은 시스템의 부담을 적게 주면서도 데이터의 변화에 따라 수행 순서를 변화시킨다. 또한 본 논문에서는 고정 연산자 순서와 비교하여 제안한 기법의 우수성을 보였다.

1. 서론

최근 데이터베이스 분야에서 연속 스트림 데이타를 지원하기 위한 많은 연구들이 진행되었다[1,2,3].

DBMS 와 같은 전형적인 데이터 처리 시스템들은 안정적인 저장소를 통해 데이터를 관리하고 데이터의 수명(lifetime)동안 데이터에 대한 질의를 처리한다. 그러나, 인터넷과 인트라넷의 확산에 따라, 스트림 데이터 처리 (stream data processing) 와 같은 새로운 분야가 등장하게 되었다.

스트림 데이터 처리 분야에서는 정적인 데이터를 여러 번 반복하여 처리할 수 있는 기존의 분야와는 달리 데이터가 연속적으로 끊임 없이 입력으로 주어지며, 입력되어지는 데이터를 검색하기 위한 질의는 사전에 시스템에 등록해 놓고 실시간에 등록된 질의를 연속적으로 처리하는 연속 질의 (continuous query) 형태를 지닌다.[1]

이러한 형태의 데이터와 질의를 지원하기 위하여 Aurora[4], Niagara[5], Hancock[6], STREAM[7], Telegraph[8] 등의 시스템이 개발 되었다. 이러한 시스템의 주요 관심은 스트림 환경에 적합한 새로운 시스템 구조의 고안, 질의 언어의 정의, 공간-복잡도가 낮은 알고리즘의 설계 등이다. 이중에서도, 스트림 데이터 환경에서의 질의들은 장기간, 연속적으로 수행되기 때문에, 효율적은 질의 계획을 생성하는 방법은 매우 중요한 성능 요소가 된다.

특히 스트림 데이터 환경에서는 시간의 변화에 따라서 데이터의 입력 분포나 입력 빈도가 변경되어짐으로 이를 반영하여 효율적인 질의 계획을 수립할 수 있는 적응성 (adaptiveness) 이 필요하게 된다. 따라서, 본 논문에서는 시간에 따라서 예측할 수 없게 특성이 바뀌는 데이터 스트림에 대한 처리에 대하여 다룬다. 특별히, 본 논문에서는 스트림 데이터에 대한 질의문을 구성하는 연산자들 간의 효율적인 수행 순서 생성 기법에 대하여 다룬다.

여기서 문제의 간결성을 위하여 기존의 연구[9]에서와 같이 질의문은 교환 가능한 필터들의 집합이라고 가정한다. 필터는 임의의 튜플을 입력 받아 필터 조건에 따라서 다음 단계 연산자로 넘기거나 탈락 (drop) 시킨다. [9] 에서는 교환 기능 필터들의 집합을 일반

적인 다중 조인으로 확장하는 방안을 포함하고 있다. 따라서, 이 필터들의 순서를 결정함으로써 가장 효율적인 질의 처리 순서를 결정할 수 있게 된다.

2. 관련 연구

스트림 데이터에서 질의 처리와 관련된 다양한 분야에서 많은 연구들이 진행되고 있다.

그 중 한 분야가 질의 색인 (query index) 분야[이다. 1 장에서 언급한 바와 같이 스트림 데이터 환경에서는 질의를 미리 등록하고 추후 데이터가 전송될 때 질의를 수행하게 된다. 여기서 등록된 질의 개수가 수 백만 개에 달할 경우 입력된 데이터를 처리 할 수 있는 질의들을 찾아내는 것도 스트림 데이터 처리 시스템의 성능에 큰 부담이 된다. 따라서 질의들의 조건을 색인화 하여 입력된 데이터를 처리할 수 있는 질의를 빠르게 찾는 방법이다[10]. Niagara CQ 와 Telegraph CQ [11] 에서는 질의 색인 기법으로 이진 탐색 트리 (binary search tree)을 변형한 IBS (interval binary search) 를 이용하여 사용한다. 이외에도 R-Tree 나 Interval SkipList[12] 이용한 질의 색인 기법들이 제안되었으며 최근에 [13]에서는 MLGF 를 이용한 질의 색인 기법을 제안하였다.

또 다른 스트림 질의 처리 관련 분야는 연산자 스케쥴링 (operator scheduling) [14, 15] 이다 다음의 그림 1 과 같이 질의는 여러 개의 연산자들로 구성되고 각 연산자 사이에는 큐가 존재한다.

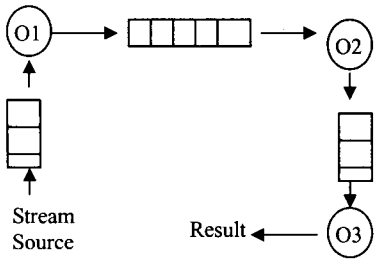

그림 1. 질의 연산자 구성

여기서 각 연산자는 입력 큐에 데이터가 존재할 때 수행이 가능하다. 따라서 연산자 스케쥴러는 실시간에 실행 가능 연산자들 중 어떠한 연산자를 수행시킬 것인가를 결정한다. 이 연산자 스케쥴링 기법으로는 기존의 운영체제의 프로세스 스케쥴링 기법에 사용되었던 FIFO(First-In-First-Out), Round robin, Minimum Cost First 등을 활용할 수 있다. 그러나, 스트림 데이터 처리 환경에서는 가용 메모리 양이 입력되는 데이터 량에 비하여 상당히 작으므로 [14]에서는 메모리 사용량을 최소하는 Chain 기법을 제안하였다. 이 기법은 현재 실행 가능 연산자들 중 단위 시간당 가장 많은 데이터를 탈락 시키는 연산자를 선정하여 수행하도록 한다.

또한 Carney 등은 [15]에서는 최소-비용 (min-cost), 최소-대기 (min –latency), 최소 메모리(min-memory) 등 다양한 목적에 따른 연산자 스케쥴링 기법들을 제안하였다.

스트림 데이터 질의와 관련된 또 다른 연구 분야는 본 논문에서 다루고자 하는 연산자 순서 문제 (operator ordering problem) 이다. 연산자 순서 문제는 하나의 입력 튜플에 대하여 여러 개의 연산자들을 적용할 수 있을 때 어떠한 순서로 적용해야 가장 효율적인가를 정하는 문제로 그림 1 과 같이 하나의 질의가 O1,O2,O3 으로 구성되어 질 때, O1-O2-O3 순서로 할지, O2-O1-O3 로 할지 등을 정하는 것이다. 이에 반하여 연산자 스케줄링은 이러한 연산자 적용 순서가 정해 졌을 때, 스트림으로 들어오는 데이타 특성 때문에 복수개의 연산자가 실행 가능해질 경우 어떠한 연산자를 수행 시킬지를 실시간에 정하는 문제이다.

연산자 순서 문제와 관련되어, Avnur 과 Hellersterin [16]은 Eddy 라는 튜플 라우팅 기법을 제안하였다. Eddy 시스템에서는 많은 튜플을 탈락 시킨 연산자가 높은 우선권을 유지하도록 하여 튜플을 먼저 처리하도록 하는 티켓 기반 처리 기법을 제안하였다. 이러한 튜플 라우팅 기법을 위하여 입력 되는 각 튜플들은 어떠한 연산자가 적용되었는지를 유지하고 있어야 하는 부담이 존재한다.

[9]에서는 연산자 순서 문제를 위하여 A-Greedy 기법을 제안하였다. A-Greedy 기법에서 연산자 순서가 $O_{f(1)}$, $O_{f(2)}$, ... ,$O_{f(n)}$이라고 할 때, 질의 처리 비용 C 는 다음과 같다라고 정의 하였다.

$$C = \sum_{i=1}^{n} t_i D_i \qquad where D_i = \begin{cases} 1 & (i==1) \\ \prod_{j=1}^{i-1}(1 - d(j \mid j-1) & (i>1) \end{cases}$$

여기서, d(i|j)는 순서가 $O_{f(1)}$, $O_{f(2)}$, ... ,$O_{f(j)}$ 까지 입력 튜플이 탈락되지 않다가 Of(i)에 탈락될 조건부 확률이고 t_i 는 연산자 $O_{f(i)}$에서 하나의 튜플을 처리하는데 필요한 시간이다. 따라서, C 는 하나의 입력 튜플이 처리 (또는 탈락) 되는 평균 시간으로 C 를 최소화하는 것을 A-Greedy 기법의 목적으로 하였다.

A-Greedy 기법은 최초 연산자로 단위 시간당 가장 많은 튜플을 탈락 시키는 연산자를 이용하고, 그 다음으로 최초 연산자로부터 출력된 튜플들을 가장 많이 탈락 시키는 연산자를 두 번째 연산자로 설정한다. 즉 A-Greedy 에서는 $d(i|i-1)/t_{f(i)} > d(j|i-1)/t_{f(j)}$ 가 되는 $O_{f(i)}$ 를 i 번째 연산자로 결정하는 욕심쟁이 경험론 규칙 (greedy heuristic rule)를 이용하였다.

이러한 욕심쟁이 경험론을 적용하기 위해서는 임의의 조건부 확률 d(j|i-1)를 구해야 하나 연산자 순서가 결정되고 나면 d(i|i-1)만을 구할 수 있을 뿐이다. 따라서 A-Greedy 에서 임의의 d(j|i-1)를 구하기 위하여 profiling 기법을 제안하였다.

A-Greedy profiling 기법은 질의 처리에 의하여 탈락된 튜플들 중 하나의 튜플 e 를 탈락-프로파일 확률 (drop-profile probability) p 에 따라서 선택을 하고 튜플 e 를 모든 연산자에 적용하여 그림 2-(a)와 같이 어떤 연산자들에서 탈락되는지 (1 로 표시) 를 파악하고 이

를 기반으로 그림 2-(b)와 같은 매트릭스 뷰를 생성한다.

O1	O2	O3	O4
1	0	0	0
0	0	1	1
0	1	1	1
1	0	1	1
0	0	0	1

(a) 프로파일

O4	O1	O3	O2
4	2	3	1
	1	0	0
		0	0
			0

(b) 매트릭스 뷰

그림 2. A-Greedy 프로파일

매트릭스 뷰 2-(b)의 첫 번째 행에서 보듯이 O4 가 가장 많은 튜플을 탈락 시킴으로, 각 연산자의 처리 시간이 같다고 가정 하면 O4 를 최초 연산자로 선정한다. 두 번째 라인은 O4 를 통과한 튜플에 대한 O1, O3, O2 의 튜플 탈락 개수이다. O1 은 하나의 튜플을 탈락 시키고 O3, O2 는 하나도 탈락 시키지 못함으로 O1 을 두 번째 연산자로 선정한다. A-Greedy 에서는 이러한 방식으로 연산자 순서를 정한다.

A-Greedy 기법의 문제점은 프로파일일 부담이 크다는 것이다. 실제 튜플은 임의의 연산자에서 탈락 될 수 있지만 프로파일 튜플은 모든 연산자에 다 적용시킴으로써 실제 튜플 처리 비용보다 프로파일 튜플 처리 비용이 훨씬 크게 된다. 즉 전체 입력 데이타의 10% 를 프로파일링 한다고 할 때 시스템 부담이 10% 보다 크게 증가한다는 것이다.

3. 제안 기법

이 절에서는 스트림 데이터의 특성이 변화함에 따라서 적응적으로 연산자 순서를 변경하는 WT-Heuristic 기법에 대하여 살펴 본다.

3.1 제안 기법의 목적

위에서 언급한 바와 같이 각 연산자는 전달 받은 튜플을 탈락 시키거나 다음 연산자로 전달하게 된다. 질의 처리의 목적은 빠른 시간에 입력 데이터를 처리하여 그 결과를 출력하는 것이다. 따라서, 질의 처리 결과에 포함되지 못하고 탈락되는 데이터를 처리 하는 시간을 최소화시켜 한다.

하나의 튜플이 입력되어 처음 연산자 O_1 을 거치면 O_1의 선택율 s_1라고 할 때, O_1의 전체 처리 시간 t_1 중 $(1-s_1)t_1$ 만큼의 시간은 데이터 탈락을 위하여 수행된 시간이다. 두 번째 연산자의 선택율이 s_2/s_1 이라고 할 경우에는 $(1-s_2/s_1)(t_1+t_2)$ 시간이 탈락 데이터를 위하여 소비된 시간이다. 따라서, 낭비 시간 (Waste-Time)은 다음과 같이 정의 할 수 있다. 이 낭비 시간은 데이터의 특성이 변화됨에 따라서 변동된다.

$$\text{Waste-Time} = \sum_{i=1}^{n}((1-s(i\mid i-1))\sum_{j=1}^{i}t_j)$$

여기서, s(i|i-1)은 하나의 튜플이 연산자 $O_{f(1)}$ ... $O_{f(i-1)}$에서 탈락되지 않았을 때, 연산자 $O_{f(i)}$에서도 탈락되지 않을 조건부 선택율(selectivity)이라고 하고, t_j는 연산자 $O_{f(j)}$에서 하나의 튜플을 처리하는 데 드는 비용이다. 따라서, 본 논문의 목적은 이 낭비 시간을 최소화 시키는 것이다.

질의가 n 개의 연산자들로 구성되어 있을 때, 모든 가능한 순서화 방법은 n! 개로 이 모든 가능한 순서들에 대하여 낭비 시간을 계산하는 것은 시스템에 많은 부담을 주게 된다. 또한 A-Greedy 방식과 같은 프로파일 기법을 사용하는 것도 절에서 언급한 바와 같이 시스템에 추가적인 부담을 유발하게 된다. 따라서, 시스템의 부담을 최소화한 WT-Heuristic 기법은 제안한다.

3.2 WT-Heuristic 기법

연속 질의문이 n 개의 교환가능 필터 연산자로 구성되어 있다고 가정하고 현재의 연산자 순서가 O_0, O_1, O_2, ..., O_n이라고 하자. 여기서 O_0는 스트림 데이터를 전송하는 데이터 소스를 지칭한다. 따라서, O_0 는 데이터의 탈락이 전혀 없는 필터 연산자라고 간주할 수 있다. 각 연산자 O_x (x>= 1)는 자신의 데이터 처리 시간 t_x와 조건부 선택율 s(x|x-1)을 유지한다. 여기서 조건부 선택률 s(x|x-1)은 O_0, ..., O_{x-1}까지 탈락되지 않은 튜플이 O_x 에서도 탈락되지 않을 확률이므로 간단히 O_{x-1} 연산자로부터 전달되어진 튜플들 중 몇 개의 튜플이 선택되었는지를 유지하고 있으면 된다.

WT-Heuristic 기법에서는 임의의 연산자 Oi-1 (i>0) 로부터 선택된 튜플들 중 튜플 e 를 교환 확률 (swap probability) p 를 이용하여 선택하고 이를 O_i 연산자 대신 O_{i+1} 연산자에 전송한다. O_{i+1} 연산자는 전송 받은 튜플 e 처리하여 선택되었는지, 탈락 되었는지의 정보를 유지한다. 이를 통하여 우리는 s(i+1|i-1)를 추정할 수 있다.

또한, 튜플 e 가 O_{i+1} 에서 탈락되지 않았다면 이를 O_i 연산자에게 전송하여 처리토록 한다. 이를 통하여 우리는 s(i|i+1)을 추정할 수 있게 된다. 즉, 일반적인 튜플은 O_i, O_{i+1} 순서로 처리되는데 반하여 교환 확률 p 에 의한 튜플 e 는 O_{i+1}, O_i 순서로 처리되게 된다.

이를 통하여 하나의 튜플이 O_i, O_{i+1} 연산자 순서 처리 했을 때의 낭비 시간 $W_{i,i+1}$`과 O_{i+1}, O_i 순서로 처리 되었을 때의 낭비 시간 $W_{i+1,i}$ 을 다음과 같이 추정할 수 있다.

$$WT_{i+1,i} = (1-s(i+1\mid i-1))(\sum_{j=1}^{i-1}t_j + t_{i+1}) + (1-(s(i\mid i+1))(\sum_{j=1}^{i+1}t_j)$$

$$WT_{i,i+1} = (1-s(i\mid i-1))(\sum_{j=1}^{i}t_j) + (1-s(i+1\mid i))(\sum_{j=1}^{i+1}t_j)$$

WT-Heuristic 기법에서 데이터의 특성이 변화되어 $WT_{i+1,i} < WT_{i,i+1}$ 이 된다면, Oi-1 연산자 다음에 Oi+1 연산자가 수행되도록 연산자 수행 순서를 수정한다.

또한 튜플 e 가 O_i 연산자는 이를 처리하고 탈락 되지 않았다면 O_{i+2} 연산자에게 전송하여 처리하도록 한다. 여기서 튜플 e 가 O_i 나 O_{i+1} 에 의하여 탈락된다면 그

연산자 적용 순서를 바꾸어도 탈락될 것이며, e 가 O_i 와 O_{i+1} 에 의하여 탈락되지 않는다면 어떠한 순서로 적용한다고 하더라도 탈락되지 않을 것이다. 따라서, WT-Heuristic 기법은 A-Greedy 기법과 달리 프로파일링 부담이 존재하지 않는다.

결론적으로 WT-Heuristic 은 스트림 데이터의 특성의 변화에 따라서 연산자 순서를 바꾸도록 한다. 그러나, 스트림 데이터 특성의 변화에 너무 민감하게 반응한다면 연산자 순서 재 정의를 위한 부담이 커지게 된다. 따라서, WT-Heuristic 에서도 A-Greedy 기법에서 사용한 변동 회피 변수(trashing-avoidance parameter) α ($0 < \alpha <= 1$)를 이용하여 $WT_{i+1,i} < \alpha WT_{i,i+1}$ 일 때 두 연산자 O_i, O_{i+1}의 순서를 변동하도록 하였다.

4. 실험

본 논문에서 제안하는 WT-Heuristic 기법의 우수성을 보이기 위하여, 시뮬레이션을 통하여 고정 연산자 순서 방식과 성능 비교를 하였다. 실험을 위한 각 설정 값은 표 1 에 정리되어 있다

연산자수	4, 6, 8
스트림데이타 수	20000, 200000
교환 확률	0.01
변동 회피 변수	0.9
무조건 선택율	0.5
연관계수(r)	2

<표 1> 실험 설정치

표 1 의 무조건 선택율이란 각 연산자가 임의의 입력 데이타에 대한 선택율을 지칭하는 것으로 이는 데이터 소스로부터 직접 데이터를 전송 받을 때의 선택율과 같다.

또한 연산자들 간의 의 연관 관계를 표현하기 위하여 연관 계수 r 을 이용하였다. n 개의 연산자는 n/r 개의 그룹으로 나누어 진다. 서로 다른 그룹에 속한 연산자는 서로 독립이며, 같은 그룹에 속한 연산자는 입력의 80%에 대하여 같은 결과를 생성한다. 다시 말하면 서로 다른 100 개의 데이터가 연산자 A 에서 탈락되었다면 같은 그룹에 속한 연산자 B 에서는 그 중 80 개가 탈락된다는 것이다.

그림 3 은 데이터 수가 200000 일때, 각 연산자 수에 따른 WT-Heuristic 기법과 고정 연산자 순서 방식과의 성능을 비교한 것으로 본 논문에서 제안하는 방식이 보다 효율적임을 알 수 있다.

5. 결론

본 논문에서는 스트림 데이터의 특성이 변화 됨에 따라 연산자 수행 순서를 변동하는 WT-Heuristic 기법을 제안하였다.

추후에, 적응형 연산자 순서 결정 알고리즘임 A-Greedy 기법과 본 논문에서 제안하는 WT-Heuristic 기법의 성능을 비교할 예정이다.

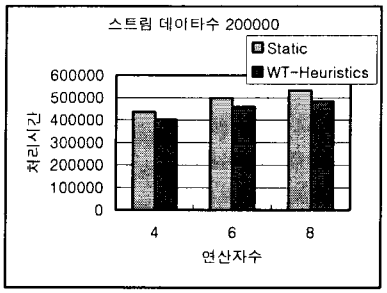

그림 3. 성능 비교

참고문헌

[1] D. Terry, D. Goldberg, D. Nichols, and B. Oki, "Continuously Queries over Append-Only Databases," ACM SIGMOD Conf., 1992.

[2] S. R. Madden, M. A. Shah, J. M. Hellerstein and V. Raman, "Continuously Adaptive Continuous Queries over Streams," ACM SIGMOD Conf.,2002

[3] S. Babu and J. Widom, "Continuous Query over Data Streams," ACM SIGMOD Record, 2001

[4] D. Carney, U. Cetintemel. M. Cherniack, C. Convey, S. Lee, G. Seidman, M, Stonebraker, N. Tatbul, and S. Zdonik, "Monitering streams-a new class of data management applications," VLDB Conf. 2002

[5] Niagara Project, http://www.cs.wisc.edu/niagara

[6] C. Cortes, J. Fisher, D. Pregibon, A. Rogers, and F. Smith, "Hancock: a language for extracting signatures from data streams," ACM SIGKDD Conf. 2000

[7] Stanford Stream Data Management (STREAM) Project. http://www-db-stanford.edu/stream

[8] J. Hellerstein, M. Franklin, S. Chandrasekaran, A. Deshpande, K. Hildrum, S. Madden, V. Raman, and M. A. Shah, " Adaptive query processing: Technology in evolution," IEEE Data Engineering Bulletin, 23(2), 2000

[9] S. Babu, R. Motwani, K. Munagala, " Adaptive Ordering of Pipelines Stream Filters," ACM SIGMOD Conf., 2004.

[10]J. Chen et al., "NiagaraCQ: A Scalable Continuos Query System for Internet Databases," ACM SIGMOD Conf., 2000.

[11] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. Madden, F. Reiss, M. A. Shah, "TelegraphCQ: Continuous Dataflow Processing," ACM SIGMOD Conf., 2003

[12]W. Pugh, "Skip List: A Probabilistic alternative to balanced trees," Communication of the ACM, 33(6), 1990

[13]H. Lim, J. Lee, M. Lee, K. Whang, I. Song, "Continuous query processing in data streams using duality of data and queries," ACM SIGMOD Conf., 2006

[14] B. Babcock, S. Babu, M. Datar, R. Motwani, "Chain: Operator Sceduling for Memory Minimization in Data Stream Systems," ACM SIGMOD Conf.,2003

[15] D. Carney, U. Cetinteme, A. Rasin, S. Zdonik, M. Cherniaack, M. Stonebraker, "Operator Scheduling in a Data Stream Manager," VLDB Conf., 2003

[16] R. Avnur and J. Hellerstein, " Eddies: Continuously adaptive query processing, " ACM SIGMOD Conf., 2000