DOI: 10.3745/KIPSTD.2008.15-D.2.163

WT-Heuristics: 스트림 데이터 환경에서의 효율적인 필터 연산자 순서화 기법

민 준 기†

요 약

인터넷과 인트라넷의 확산에 따라, 스트림 데이터 처리 (stream data processing) 와 같은 새로운 분야가 등장하게 되었다. 스트림 데이터는 실시간적이고 연속적으로 생성된다. 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 기법인 WT-Heuristics를 제안한다. WT-Heuristics 기법은 연산 실행 순서에서 인접한 두 연산자들의 연산 순서만을 고려함으로써 효율적으로 연산자 순서를 결정할 수 있다. 또한 본 논문에서 제안하는 방법은 시스템의 부담을 적게 주면서도 데이터의 변화에 따라 수행 순서를 변화시킨다.

키워드 : 스트림 데이터, 연산자 순서

WT-Heuristics: An Efficient Filter Operator Ordering Technology in Stream Data Environments

Jun-ki Min

ABSTRACT

Due to the proliferation of the Internet and intranet, a new application domain called stream data processing has emerged. Stream data is real-timely and continuously generated. In this paper, we focus on the processing of stream data whose characteristics vary unpredictably by over time. Particularly, we suggest a method which generates an efficient operator execution order called WT-Heuristics. WT-Heuristics efficiently determines the operator execution order since it considers only two adjacent operators in the operator execution order. Also, our method changes the execution order with respect to the change of data characteristics with minimum overheads.

Key Words : Stream Data, Operator Order

1. 서 론

최근 데이터베이스 분야에서 연속 스트림 데이타를 지원하기 위한 많은 연구들이 진행되었다 [1, 2, 3]. DBMS와 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[1]. 스트림 데이터 처리 분야에서는 정적인 데이터를 여러 번 반복하여 처리할 수 있는 기존의 분야와는 달리 데이터가 연속적으로 끊임 없이 입력으로 주어지며, 안정적이고 지속적인 데이터 이미지의 다중 검색의 이점 없이 입력되는 데이터를 연속적으로 처리하여야 한다. 따라서, 스트림 데이터 환경에서는, 검색하기 위한 질의를 사전에 시스템에 등록해 놓고 실시간에 등록된 질의를 연속적으로 처리하는 연속 질의 (continuous query) 형태를 지닌다.

이러한 형태의 데이터와 질의를 지원하기 위하여 Aurora [2], Niagara [3], Hancock [4], STREAM [5], Telegraph [6] 등의 시스템이 개발 되었다. 이러한 시스템의 주요 관심은 스트림 환경에 적합한 새로운 시스템 구조의 고안, 질의 언어의 정의, 공간-시간 복잡도가 낮은 알고리즘의 설계 등이다. 이중에서도, 스트림 데이터 환경에서의 질의들은 장기간, 연속적으로 수행되기 때문에, 효율적인 질의 계획을 생성하는 방법은 매우 중요한 성능 요소가 된다.

† 종신회원 : 한국기술교육대학교 인터넷미디어공학부 조교수
논문접수 : 2007년 5월 8일, 심사완료 : 2007년 7월 9일

특히 스트림 데이터 환경에서는 시간의 변화에 따라서 데이터의 분포나 입력 빈도가 변경됨으로 이를 반영하여 효율적인 질의 계획을 수립할 수 있는 적응성(adaptiveness)이 필요하게 된다. 따라서, 본 논문에서는 스트림 데이터에 대한 질의문을 구성하는 연산자들 간의 효율적인 수행 순서 생성 기법에 대하여 다룬다. 여기서 문제의 간결성을 위하여 기존의 연구 [7]에서와 같이 질의문은 교환 가능한 필터들(commutative filters)의 집합이라고 가정한다. 이러한 교환 가능 필터는 스트림 환경에서는 일반적이다 [8,9]. 임의의 튜플 t가 필터에 입력되었을 때, 필터는 사전에 정의된 조건에 따라서 입력 튜플 t를 탈락(drop) 시키거나 다음 단계의 연산자로 넘기게 된다. 전체 질의 성능은 서로 다른 필터 연산자 순서에 따라서 폭넓게 변동된다. 예를 들어, 연산자 O1은 값이 1, 3, 5 인 튜플을 단위 시간 내에 탈락시키고 연산자 O2는 값이 2, 4,6인 튜플을 단위 시간에 탈락시킨다고 하자. 만약 입력 스트림이 2, 4, 6이라고 할 경우, 연산자 순서가 O1, O2 일 경우 전체 비용은 6 단위 시간이다. 이에 반하여 연산자 순서가 O2, O1 일 경우에는 3 단위 시간만 걸릴 뿐이다. [7]의 연구에서 교환 기능 필터들의 집합을 일반적인 다중 조인으로 확장하는 방안을 제시하였다. 따라서, 이 필터들의 순서를 결정함으로써 가장 효율적인 질의 처리 순서를 결정할 수 있게 된다.

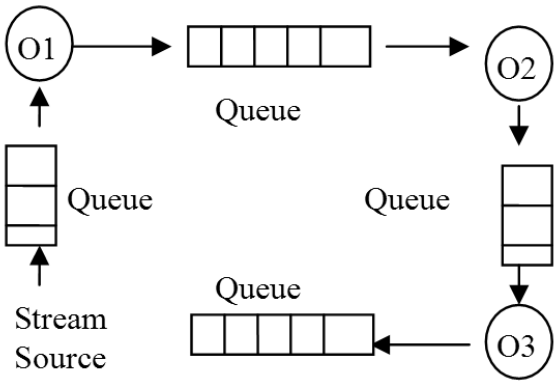
본 논문의 나머지 부분의 구성은 다음과 같다. 2장에서는 스트림 데이터 분야에서 각광 받고 있는 중요한 연구 이슈들을 살펴 보고 3장에서는 본 논문에서 제안하는 효율적인 연산자 순서화 기법을 대하여 자세히 기술하고 4장에서 실험을 통하여 제안된 기법의 성능을 평가하고 마지막으로 5장에서 결론을 맺는다.

2. 관련연구

최근 스트림 데이터 분야에서 질의 처리와 관련된 다양한 주제에 대하여 많은 연구들이 진행되고 있다.

그 중 한 분야가 질의 색인(query index) 분야이다. 1장에서 언급한 바와 같이 스트림 데이터 환경에서는 질의를 미리 등록하고 추후 데이터가 전송될 때 질의를 수행하게 된다. 여기서 등록된 질의 개수가 아주 많은 경우 입력된 데이터를 처리 할 수 있는 질의들을 찾아내는 것 자체가 스트림 데이터 처리 시스템의 성능에 큰 부담이 된다. 따라서 질의들의 조건(predicate)을 색인화 하여 입력된 데이터를 처리할 수 있는 질의를 빠르게 찾을 수 있다 [10]. Niagara CQ [3]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 [12]나 Interval SkipList [13] 이용한 질의 색인 기법들이 제안되었으며 최근에 [14]에서는 MLGF를 이용한 질의 색인 기법을 제안하였다.

또 다른 스트림 질의 처리 관련 분야는 연산자 스케쥴링(operator scheduling) [16, 17] 이다 다음의 (그림 1)과 같이



(그림 1) 간단한 질의 구조

질의는 여러 개의 연산자들로 구성되고 각 연산자 사이에는 큐(Queue)가 존재한다. 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 [15] 등을 활용할 수 있다. 그러나, 스트림 데이터 처리 환경에서는 가용 메모리 양이 입력되는 데이터 량에 비하여 상당히 작으므로 Babcock 등[16]은 메모리 사용량을 최소하는 근접 최적(near optimal)인 Chain이라는 스케쥴링 기법을 제안하였다. 개념적으로, 이 기법은 현재 실행 가능 연산자들 중 단위 시간당 가장 많은 데이터를 탈락시키는 연산자를 선정하여 수행하도록 한다. 또한 Carney 등은 [17]에서는 최소-비용(min-cost), 최소-대기(min - latency), 최소 메모리(min-memory) 등 다양한 목적에 따른 연산자 스케쥴링 기법들을 제안 하였다.

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, O3으로 구성되어 질 때, O1-O2-O3 순서로 할지, O2-O1-O3로 할지 등을 정하는 것이다. 이에 반하여 연산자 스케줄링은 이러한 연산자 적용 순서가 정해졌을 때, 스트림으로 들어오는 데이타 특성 때문에 복수개의 연산자가 실행 가능해질 경우 어떠한 연산자를 수행시킬 지를 실시간에 정하는 문제이다.

가장 좋은 연산자 순서는 필터의 특성(예, 선택율, 처리시간)과 입력 스트림 데이터의 특성에 따라서 변화된다. 앞에서 언급한 바와 같이, 스트림 데이터의 특성은 시간에 따라서 변화된다. 따라서 연산자 순서 문제를 처리하기 위하여서는 적응형 접근방법이 필요하다.

연산자 순서 문제와 관련되어, Avnur과 Hellersterin [18]은 Eddy라는 튜플 라우팅 기법을 제안하였다. Eddy 시스템에서는 많은 튜플을 탈락시킨 연산자가 높은 우선권을 유지

하도록 하여 튜플을 먼저 처리하도록 하는 티켓 기반 처리 기법을 제안하였다. 이러한 튜플 라우팅 기법을 위하여 입력되는 각 튜플들은 어떠한 연산자가 적용되었는지를 유지하고 있어야 하는 부담이 존재한다.

[7]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 C는 다음과 같다라고 정의 하였다.

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

수식 (1)에서, d(i|j)는 순서가 $O_{f(1)}$, $O_{f(2)}$, ... ,$O_{f(j)}$ 까지 입력 튜플 e가 탈락되지 않다가 $O_{f(i)}$에 탈락될 조건부 확률이고 t_i 는 연산자 $O_{f(i)}$에서 하나의 튜플을 처리하는데 필요한 시간이다. 따라서, C는 하나의 입력 튜플이 처리 (또는 탈락) 되는데 걸리는 평균 시간이다. C를 최소화하는 연산자 순서를 찾는 것이 A-Greedy 기법의 목적으로 하였다.

이러한 목적을 달성하기 위하여, A-Greedy는 다음의 수식을 만족하도록 연산자 순서를 재배열하는 욕심꾸러기 경험론 (Greedy Heuristics) 규칙을 사용하였다.

$$\frac{d(i \mid i-1)}{t_i} \geq \frac{d(j \mid i-1)}{t_j} \quad , 1 \leq i < j \leq n \tag{2}$$

다시 말하면, A-Greedy 기법은 단위 시간당 가장 많은 튜플을 탈락시키는 연산자를 최초 연산자로 이용한다. 그리고, A-Greedy기법은 처음 연산자로부터 전송된 튜플들 중 가장 많은 튜플을 탈락시키는 연산자를 두 번째 연산자로 설정한다. 나머지 연산들의 순서들도 비슷한 방식으로 결정된다.

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

A-Greedy 기법에서는 프로파일들을 관리하기 위하여 프로파일 윈도우를 이용한다. 프로파일 윈도우는 질의 처리 시에 탈락된 입력 스트림에서 표본으로 추출한 프로파일 튜플들의 슬라이딩 윈도우(sliding window) 이다. 하나의 프로파일 튜플은 m개의 부울리안(Boolean) 에트리뷰트 $b_1,\cdots,b_m$들로 구성되어 있으며 이 부울리안 변수는 연산자 $O_1,\cdots,O_m$에 대응된다.

A-Greedy 프로파일링 기법은 질의 처리에 의하여 탈락된 튜플들 중 하나의 튜플 e를 탈락-프로파일 확률(drop-profile probability) p 에 따라서 선택을 한다. 그리고, A-Greedy 프로파일러(Profiler)는 튜플 e를 모든 연산자에 적용하여 프로파일 튜플을 생성한다. 여기서, 만약 연산자 O_i가 튜플 e를 탈락시키면 애트리뷰트 b_i의 값은 1이고 그렇지 않다면 b_i의 값은 0이다 (그림 2-(a) 를 보시오.).

b_1	b_2	b_3	b_4
1	0	0	0
0	0	1	1
0	1	1	1
1	0	1	1
0	0	0	1

(a) 프로파일 윈도우

O_4	O_1	O_3	O_2
4	2	3	1
	1	0	0
		0	0
			0

(b) 매트릭스 뷰

(그림 2) A-Greedy 프로파일

A-Greedy 재최적화기(reoptimizer)는 욕심꾸러기 경험론 규칙을 따르는 연산자 순서를 유지한다. 이를 위하여 A-Greedy 재최적화기는 프로파일 윈도우를 이용하여 (그림 2-(b))와 같은 특별한 뷰(View)를 유지한다. (그림 2-(b))의 매트릭스 뷰의 첫 번째 행에서 보듯이 O_4가 가장 많은 튜플을 탈락시킨다. 따라서, 각 연산자의 처리 시간이 같다고 가정 하면O_4를 최초 연산자로 선정한다. 두 번째 라인은 O_4를 통과한 튜플에 대한 O_1, O_3, O_2의 튜플 탈락 개수이다. O_1은 하나의 튜플을 탈락시키고 O_3, O_2는 하나도 탈락시키지 못함으로 O_1을 두 번째 연산자로 선정한다. A-Greedy에서는 이러한 방식으로 연산자 순서를 정한다.

A-Greedy기법의 문제점은 프로파일링 부담이 크다는 것이다. 실제 튜플은 임의의 연산자에서 탈락 될 수 있지만 프로파일을 위한 튜플은 모든 연산자에 다 적용시킨다. 따라서, 일반적인 입력 튜플의 비용은 수식 (1)과 같이다. 그러나, 프로파일에 사용된 튜플의 비용은 $\sum_{i=1}^{n} t_i$ 로써, 수식 (1)보다 크다. 즉, 실제 튜플 처리 비용보다 프로파일 튜플 처리 비용이 훨씬 크게 된다. 다시 말해, 전체 입력 데이타의 10%를 프로파일링 한다고 할 때 시스템 부담이 10% 보다 크게 증가한다는 것이다.

3. WT-Heuristics

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

3.1 WT-Heuristics의 목적

위에서 언급한 바와 같이 각 연산자는 전달 받은 튜플을 탈락시키거나 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.

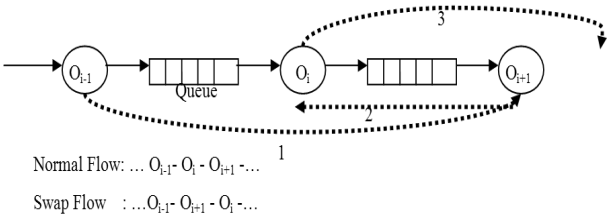
하나의 튜플이 입력되어 처음 연산자 O_1을 거칠 때, O_1의 선택율 s_1라고 하면, O_1의 전체 처리 시간 t_1중 $(1-s_1)t_1$ 만큼의 시간은 데이터 탈락을 위하여 수행된 시간이다. 두 번째 연산자 O_2에 있어서, 연산자 O_1에서 탈락되지 않은 튜플들이 연산자 O_2에서도 탈락하지 않을 조건부 선택율을 s_2이라고

할 경우에는 $(1-s_2)(t_1+t_2)$ 시간이 탈락 데이터를 위하여 소비된 시간이다. 따라서, 각 연산자들에 대한 총 낭비 시간(Total Waste-Time) W는 다음과 같이 정의 할 수 있다. 이 낭비 시간은 데이터의 특성이 변화됨에 따라서 변동된다.

$$W = \sum_{i=1}^{n}((1-s(i\mid 0,1,...,i-1))\sum_{j=1}^{i}t_j) \quad (3)$$

수식 (3)에서, s(i|0,1,…,i-1)은 하나의 튜플 e가 연산자 O_1,…, O_{i-1} 에서 탈락되지 않았을 때, 연산자 O_i에서도 탈락되지 않을 조건부 선택율(selectivity)이다. t_j는 연산자 O_j에서 하나의 튜플을 처리하는 데 드는 비용이다.

각 연산자의 조건부 확률 s(i|0,1.,…, i-1)는 데이터의 특성에 따라서 변동됨으로 총 낭비 시간 또한 데이터의 특성의 변동에 따라서 변화된다. 따라서, 본 논문의 목적은 총 낭비 시간 W를 최소화시키고 적응적으로 연산자 순서를 변화시키는 것이다. 그러나, 질의가 n개의 연산자들로 구성되어 있을 때, 모든 가능한 순서화 방법은 n! 개이다. 이 모든 가능한 순서들에 대하여 낭비 시간을 계산하는 것은 시스템에 많은 부담을 주게 된다. 또한 A-Greedy 방식과 같은 프로파일 기법을 사용하는 것도 2절에서 언급한 바와 같이 시스템에 추가적인 부담을 유발하게 된다. 따라서, 본 논문에서는 시스템의 부담을 최소화한 WT-Heuristics 기법은 고안하였다. WT-Heuristics 기법은 연산자 수행 순서 중 인접한 두 연산자 간의 순서 만을 고려함으로써 효율적으로 연산자 순서를 결정할 수 있다.



(그림 3) 연산자 교환

3.2 WT-Heuristics 기법

연속 질의문이 n개의 교환 가능 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_i ($1 \le i \le n$)는 자신의 데이터 처리 시간 t_i와 조건부 선택율 s(i|0,1,…i-1)을 유지한다. 여기서 조건부 선택률 s(i|0,1…,i-1)은 O_0, …, O_{i-1}까지 탈락되지 않은 튜플이 O_i에서도 탈락되지 않을 확률이므로 간단히 O_{i-1} 연산자로부터 전달되어진 튜플들 중 몇 개의 튜플이 선택되었는지를 유지하고 있으면 된다.

3.1절에서 언급한 바와 같이 모든 가능한 연산자 순서들에 대한 총 낭비시간을 계산하는 것은 많은 부담이 따른다. 예를 들어, A-Greedy 기법에서는, $O(n^2)$개의 탈락율을 매트릭스 뷰에서 유지하여야 한다. 이에 반하여, 우리가 제안하는 기법인 WT-Heuristics는 비교적 적은 수의 선택율 만을 유지한다. WT-Heuristics에서는 각 연산자가 3종류의 조건부 선택율을 유지한다. 이러한 선택율 중 하나가 위에서 언급한s(i|0,1,…,i-1)로써 이는 쉽게 구할 수 있다.

또한 조건부 확률 s(i+1|0,1,…,i-1)을 예측한다. WT-Heuristics기법에서 이 조건부 확률을 얻기 위하여 임의의 연산자 O_{i-1} ($1 \le i \le n$) 에서 탈락되지 튜플들 중 튜플 e를 교환 확률 (swap probability) p를 이용하여 선택한다. 만약 튜플 e가 선택되었다면, 이를O_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|0,1,…,i-1)를 추정할 수 있다.

마지막으로, 튜플e가 O_{i+1} 에서 탈락되지 않았다면 이를 O_i 연산자에게 전송하여 처리토록 한다. 이를 통하여 우리는 조건부 확률s(i|0,1,…,i-1,i+1)을 추정할 수 있게 된다.

(그림 3)에서 보이듯이, 일반적인 튜플은 현재의 연산자 순서 …,O_{i-1},O_i,O_{i+1},… 에 따라서 처리된다. 이에 반하여 O_{i-1}에서 탈락되지 않은 튜플들 중 교환 확률 p에 의하여 선택된 튜플 e는 …, O_{i-1},O_{i+1}, O_i,…순서로 처리되게 된다. 이를 위하여 연산자 O_{i-1}는 입력받은 튜플이 탈락되지 않았을 경우 [0,1) 사이의 균등 분포(Uniform distribution)을 따르는 난수(random number)를 발생시켜 이 값이 교환 확률 p보다 작으면 연산자 O_{i+1} 에게 전송하고 p보다 클 경우에서는 연산자 O_i에게 전송한다. 이러한 난수 생성 부담은 A-Greedy 알고리즘에서도 탈락-프로파일 확률(drop-profile probability)에 의하여 동일하게 발생한다.

WT-Heuristics에서는 위에서 언급한 3가지 조건부 확률을 이용하여 하나의 튜플이 O_i, O_{i+1} 연산자 순서 처리 했을 때의 부분 낭비 시간 $WT_{i,i+1}$과 O_{i+1}, O_i 순서로 처리 되었을 때의 부분 낭비 시간 $WT_{i+1,i}$을 수식 (4)와 같이 추정 할 수 있다. 수식 (4) 의 $WT_{i+1,i}$는 연산자 O_{i-1} 다음에 O_{i+1}수행될 때 연산자 O_{i+1}의 낭비 시간과 연산자 O_{i+1} 다음에 O_i가 수행될 때 연산자 O_i의 낭비 시간의 합으로 표현되며, $WT_{i,i+1}$ 은 연산자 O_{i-1} 다음에 O_i와 O_{i+1}가 수행될 때 O_i의 낭비 시간과 O_{i+1}의 낭비 시간의 합으로 표현된다.

$$\begin{aligned} WT_{i+1,i} &= (1-s(i+1\mid 0,...,i-1))(\sum_{j=1}^{i-1}t_j + t_{i+1}) \\ &\quad +(1-(s(i\mid 0,...,i-1,i+1))(\sum_{j=1}^{i+1}t_j) \\ WT_{i,i+1} &= (1-s(i\mid 0,...,i-1))(\sum_{j=1}^{i}t_j) \\ &\quad +(1-s(i+1\mid 0,...,i-1,i))(\sum_{j=1}^{i+1}t_j) \end{aligned} \quad (4)$$

즉, $WT_{i,i+1}$은 현재 연산자 순서의 부분 낭비 시간을 나타내고 $WT_{i+1,i}$는 연산자 O_i와 O_{i+1}이 서로 교환되었을 때의 추

정 부분 낭비 시간을 나타냄으로, 만약 $W_{i,i+1}$이 $W_{i+1,i}$보다 크다면 WT-Heuristics는 O_{i-1} 연산자 다음에 O_{i+1} 연산자가 수행되고 다음에 O_i 연산자가 수행되도록 연산자 순서를 재 배열한다.

입력 스트림의 특성이 변화되면, 추정 선택율도 영향을 받는다. 따라서, 각 연산자는 지속적으로 특별한 튜플 e를 선택하고 위에 언급한 세가지 선택율을 지속적으로 추정한다. 따라서, $WT_{i,i+1}$과 $WT_{i+1,i}$를 실시간으로 계산할 수 있다. 결론적으로 WT-Heuristics 기법은 연산자 순서를 적응적으로 변화시킬 수 있다.

또한, 만약 교환 확률 p에 의하여 연산자 O_{i-1}에서 선택된 튜플 e가O_{i+1}와 O_i에서 탈락되지 않으면 O_{i+2} 연산자(만약 존재한다면)에게 전송하여 처리하도록 한다.

여기서 주목할 점은, 교환 확률 p에 의하여 선택된 튜플 e가 O_i나 O_{i+1}에 의하여 탈락된다면 그 연산자 적용 순서에 상관 없이 결국 튜플e는 탈락될 것이다. 또한 튜플 e 가 O_i와 O_{i+1}에 의하여 탈락되지 않는다면 어떠한 순서로 적용한다고 하더라도 탈락되지 않고 연산자 O_{i+2}로 보내어질 것이다. 따라서, WT-Heuristics 기법은 A-Greedy 기법과 달리 프로파일링 부담이 존재하지 않는다. 즉, A-Greedy 기법에서는 임의의 연산자에서 탈락된 튜플을 모든 연산자에 적용하는 프로파일링 기법을 적용함으로써 더 이상 처리 하지 않아도 될 튜플에 대한 추가 처리 비용이 발생되게 되나, WT-Heuristics기법에서는 추가 처리가 필요한 튜플(즉, 연산자 O_{i-1}에서 탈락되지 않은 튜플)에 대하여 추후 연산자 순서를 바꾸어 적용하는 방식을 적용함으로써 A-Greedy기법과 같은 추가 처리 비용이 발생하지 않는다. 더욱이, A-Greedy 기법은 $O(n^2)$개의 조건부 확률을 유지하는데 반하여, WT-Heuristics는 O(n)개의 조건부 확률(연산자마다 세 개의 선택율)을 유지한다. 따라서 우리가 제안한 기법인 WT-Heuristics기법이 보다 적은 메모리를 사용한다.

A-Greedy 기법에서는 새로운 프로파일 튜플이 생성될 때 마다, 현재의 연산자 순서가 욕심꾸러기 경험론 규칙에 위배되는 지를 검사한다. 그리고 매트릭스 뷰를 통해서n-i+1개의 연산자들 중에서 가장 좋은 i 번째 연산자를 선정한다. 이에 반하여 WT-Heuristics는 현재 연산자 순서에서 인접된 두 연산자의 순서를 교환할 경우 성능이 향상된다면 해당 두 연산자의 순서를 변경한다. 따라서, WT-Heuristics는 한 순간에 두 연산자 간의 연산 순서만을 고려함으로 A-Greedy에서 생성되는 최적의 연산자 순서로 도달하는데 많은 시간이 걸릴 수 있다. 또한 WT-Heuristics는 지역적 최적 연산자 순서(local optimal operator order)에 머무를 수도 있다. 그러나 WT-Heuristics는 추가적인 프로파일 부담이 없으므로 A-Greedy 기법에 비하여 낮은 처리 부담을 가진다. 4장의 실험 결과에서 다양한 실험 환경 상에서 WT-Heuristics의 우수성을 보였다.

결론적으로 WT-Heuristics은 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. 예를 들어, 인접한 두 연산자 O_A,O_B에 대하여 특정 시점 t에서 $WT_{A,B}$의 값이 10이고, $W_{B,A}$의 값이 9.8일 경우, $WT_{B,A}$의 값이 $W_{A,B}$보다 작으므로 연산자 순서를 O_B-O_A로 할 것이다. 그리고 t+1 시점에서 $WT_{A,B}$의 값이 9.7 이 되고, $WT_{B,A}$의 값이 9.9가 되면 다시 연산자 순서를 O_A-O_B로 할 것이다. 즉, 인접된 두 연산자의 낭비시간의 차이가 크지 않을 경우, 지속적으로 두 연산자의 순서를 바꾸려고 하는 변동(trashing)이 발생할 수 있다. 따라서, WT-Heuristics에서도A-Greedy 기법에서 사용한 변동 회피 변수(trashing-avoidance parameter) α (0 < α <= 1)를 이용하여 $WT_{i+1,i} < \alpha\ WT_{i,i+1}$ 일 때 두 연산자 O_i, O_{i+1}의 순서를 교환하도록 하였다. 다시 말하면, $WT_{i+1,i}$와 $WT_{i,i+1}$의 비율(= $WT_{i+1,i}$/ $WT_{i,i+1}$)이 α 보다 작을 경우에만 두 연산자의 순서를 교환하도록 하였다.

4. 실 험

본 논문에서, 시뮬레이션을 통하여 본 논문에서 제안하는 WT-Heuristics의 성능을 분석하였다. 합성 데이터 세트를 이용하여 WT-Heuristics의 성능을 고정 연산자 순서 방식 및 A-Greedy 기법과 비교하였다. 우리의 실험에서 WT-Heuristics는 우리가 예상한 대로 뛰어난 성능을 보여주었다.

4.1 실험 환경

실험은 윈도우 XP에 탑재된 펜티엄 IV-1.7GHz에서 수행되었으며 메인 메모리는 1GByte이다. WT-Heuristics의 우수성을 보이기 위하여 우리는 3가지 연산자 순서 알고리즘-WT-Heuristics, A-Greedy, 고정 연산자 순서(static) 방식을 직접 구현하였다. 고정 연산자 순서 방식은 초기 연산자 순서를 변형하지 않고 그대로 유지하는 방법이고 나머지 두 방법은 적응적으로 연산자 순서를 변경한다.

우선, 합성 데이터 세트와 본 실험에서 사용한 설정값에 대하여 설명하도록 한다. A-Greedy 기법과의 공정한 비교를 위하여 본 실험에서는 A-Greedy 기법 [7] 의 실험 환경을 활용하였다. 합성 데이터 세트는 값의 영역이 [0,10000]이 되도록 하는 균등 분포를 이용하였다. 실험을 위한 다른 설정값은 <표 1>에 정리하였다.

〈표 1〉 설정값

설정변수	기본값
연산자 수	6
입력 데이터 수	100,000
연산자 처리 비용	1
비조건 선택율	50%
상관 계수(T)	T=2
변동 회피변수(α)	α=0.9
탈락-프로파일 확률 (For A-Greedy)	0.01
프로파일 윈도우 크기 (For A-Greedy)	500
교환 확률 (for WT-Heuristics)	0.01

우리는 연산자의 수를 기본값으로 6으로 하였다. 그리고 100,000개의 스트림 데이터를 이용하여 실험을 수행하였다. 각 연산자 처리 비용은 1로 설정하였다. 나중에, 4.2절에서 연산자 수, 스트림 데이터 개수, 연산자 처리 비용을 변화시킨 실험 결과에 대해서도 보인다.

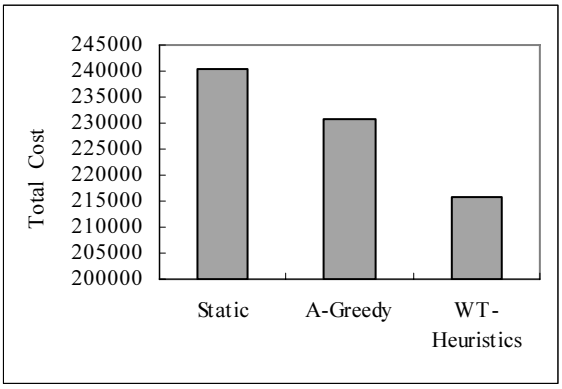
이 실험에서 각 연산자의 비조건 선택율은 50%이다. 이는 각 연산자 O_i가 맨 처음 연산자이면 50%의 데이터를 탈락시킨다는 것을 의미한다.

성능에 영향을 주는 하나의 요소는 연산자들 간을 상관관계이다. 연산자들 간의 상관 관계를 나타내기 위하여 [7]에서 소개된 상관 계수를 이용하였다. n개의 연산자는 T개의 연산자들로 구성된 ⌈n/T⌉ 개의 그룹으로 나누어진다. 여기서 T를 상관 계수라 한다. 만약 두 연산자가 서로 다른 그룹에 속해 있다면 이 두 연산자는 서로 독립이다. 그러나 만약 같은 그룹에 속해 있다면 입력 데이터에의 80%에 대하여 동일한 결과를 생성하는 상관관계를 지닌다. 다시 말하면, 연산자 O_i와 O_j 가 동일한 그룹에 속해 있다면, 연산자 O_i에 의하여 서로 다른 100개의 데이터가 탈락되었다면 이중 80개는 O_j에 의하여서도 탈락된다는 것이다.

4.2 실험 결과

첫 번째로, <표 1>에서 보인 바와 같이 연산자 개수가 6일 때, 3가지 연산자 순서화 방법에 대한 성능을 측정하였다. 기본 설정 값을 이용한 실험 결과는 (그림 4)에 나타나 있다.

(그림 4)에서 보듯이, 본 논문에서 제안한 WT-Heuristics가 가장 좋은 질의 수행 성능을 보여주고 있다. 고정 연산자 순서(static) 방식은 연산자 순서를 변동시키지 않음으로 가장 나쁜 성능을 보여주고 있다. 이에 비하여 A-Greedy 방식과 WT-Heuristics는 고정 연산자 방식보다 좋은 성능을 보여주고 있다. 고정 연산자 순서 방식과 비교하여, A-Greedy 기법은 약 4%의 성능 이득을 보여주고 있으며 WT-Heuristics는 약 10%의 성능 향상율을 보여주고 있다. 이는 WT-Heuristics 기법이 프로파일 생성과 같은 추가 부담이 없기 때문에 다른 기법들 보다 뛰어난 성능을 보인 것이다.

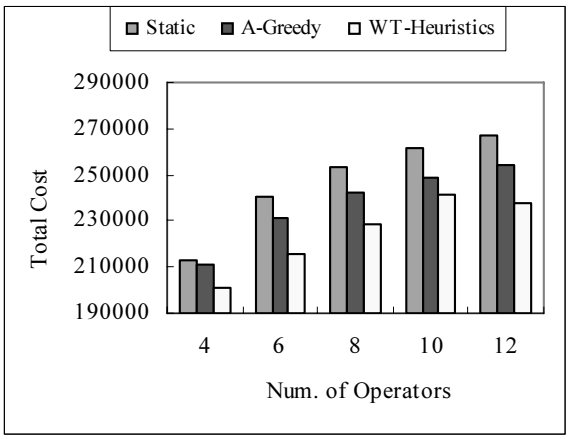


(그림 4) 기본 설정값을 이용한 성능

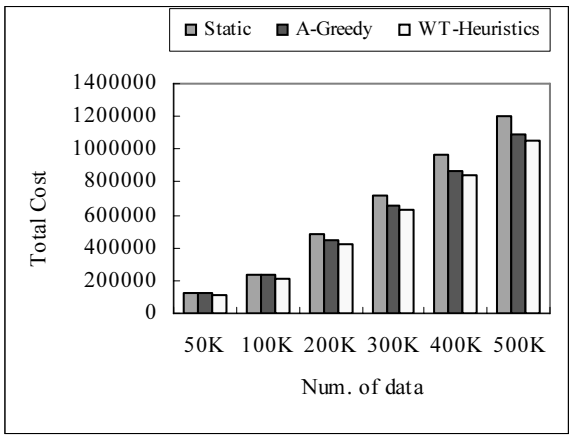
(그림 5)는 연산자 개수에 의한 영향을 보여준다. (그림 5)에서 보듯이 연산자 개수가 증가할 수록 각 기법들의 처리 비용 또한 증가한다. 그럼에도 불구하고 각 기법들의 성능 비율은 크게 바뀌지 않는다.

(그림 6)은 스트림 데이터의 개수를 50,000, 100,000, 200,000, 300,000, 400,000, 500,000개로 변경시키면 측정한 각 기법들의 질의 처리 성능을 보여주고 있다. (그림 6)에서 보이는 바와 같이 데이터의 개수가 작을 경우 (즉, 스크림 데이터의 개수가 50,000 일 때), A-Greedy가 가장 나쁜 성능을 보여준다. A-Greedy 기법은 프로파일링을 필요로 한다. 따라서, 데이터의 개수가 적을 경우, 프로파일링의 부담만 나타나고 연산자 순서의 재 배치로 인한 이점은 나타나지 않는다. 그러나, 데이터의 개수가 증가함에 따라서, A-Greedy의 프로파일링 부담은 연산자 순서화에 따른 성능 향상과 상쇄된다.

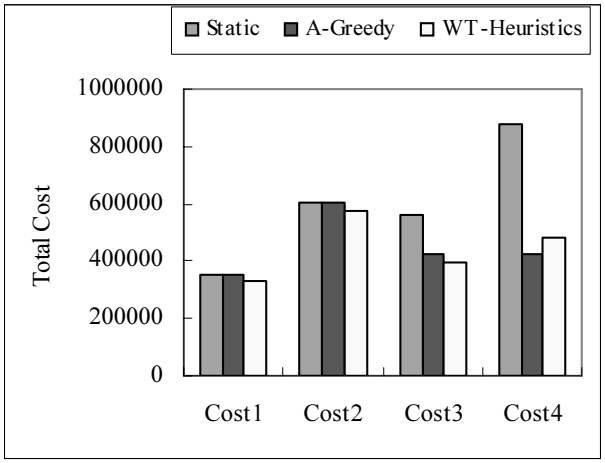
이에 비하여 WT-Heuristics는 프로파일링 부담이 존재하지 않기 때문에 데이터 개수가 작을 때에도 고정 연산자 순서 기법에 비하여 좋은 성능을 보여주고 있다. 평균적으로 A-Greedy와 WT-Heuristics의 성능 향상율은 각각 약 9%와 12%이다.



(그림 5) 다양한 연산자 개수를 이용한 성능



(그림 6) 스트림 데이터 개수의 변화에 따른 성능



(그림 7) 연산자 비용 변화에 따른 성능

마지막으로, 각 연산자의 비용을 변화시켜서 각 방법들의 성능을 측정하였다. 연산자 비용 변화를 위한 실험 사례로서 네 종류의 연산자 비용 사례-Cost1, Cost2, Cost3, Cost4-를 생성하였다. Cost1 사례로써, 우리는 연산자 비용 리스트<1, 1, 1, 1, 5, 5> 를 여섯 개의 연산자에 순서대로 할당하였다. Cost2로써는 연산자 비용 리스트<1, 1, 5, 5, 5, 5>를 사용하였으며, Cost3로는 <1, 5, 1, 5, 1, 5>를 Cost4로는 <5, 1, 5, 1, 5, 1>을 사용하였다.

각 기법들에 대한 성능은 (그림 7)에 나타나 있다. Cost1, Cost2, Cost3 사례들에 있어서는 WT-Heuristics가 가장 좋은 성능을 보인다. Cost1 사례에 있어서는 A-Greedy가 다른 기법들에 비하여 약간 나쁜 성능을 보인다. Cost 4 사례에서는 WT-Heuristics기법이 A-Greedy 기법보다 나쁜 성능을 보여주고 있다. 이 실험에서는 우리는 WT-Heuristics의 최종 연산자 순서와 A-Greedy의 연산자 순서가 동일함을 확인하였다. 앞에서 언급한 바와 같이 WT-Heuristics기법은 한 번에 인접한 두 연산자의 순서만을 고려하기 때문에 최적의 연산자 순서에 도달하는데 많은 시간이 걸릴 수 있다. Cost4 사례는 이러한 경우를 보여주고 있다. 그러나 다른 실험 결과들에서 보듯이 WT-Heuristics는 실시간 처리 부담을 최소화 하면서 연산자 순서를 결정하는 특징을 가지고 있다. 따라서, Cost4에서 보인 성능 차이는 시간이 지남에 따라서 경감될 수 있다.

결론적으로 WT-Heuristics는 대부분의 경우에 가장 좋은 성능을 보여 준다.

5. 결 론

최근에 들어와서 지속적으로 생성되는 스트림 데이터를 연속적으로 처리하는 스트림 데이터 처리를 위한 다양한 기법들에 대한 연구가 활발히 진행되고 있다. 본 논문에서는 다양한 스트림 데이터 처리에 관한 연구 주제들 중에서 연산자 순서화 문제에 대하여 다루었다. 스트림 데이터는 시간에 따라서 데이터의 평균값과 같은 특성이 변화되는 특징을 지니고 있다. 따라서, 시간에 따라서 변화되는 데이터의 특성에 맞추어 효율적인 연산자 순서를 생성하는 것이 필요하다.

따라서 본 논문에서는 WT-Heuristics 기법을 제안하였다. WT-Heuristics기법의 목적은 낭비 시간을 최소화하는 것이다. WT-Heuristics 기법은 교환 확률에 따라서 입력 튜플을 인접된 두 연산자중 뒤에 오는 연산자에 적용함으로써 현재 연산자 순서상의 낭비 시간과 두 연산자의 순서를 바꾸었을 때의 추정 낭비 시간을 유지하도록 하였다. 그리고 현재의 낭비 시간과 추정 낭비 시간을 비교하여 두 연산자의 순서를 교환하는 것이 더 좋을 경우 두 연산자의 순서를 바꾸도록 한다. 이를 통하여 WT-Heuristics기법은 추가적인 비용 부담 없이 효율적으로 연산자 순서를 결정할 수 있다. 또한 본 연구에서는 제안하는 WT-Heuristics 기법의 효율성을 보이기 위하여 WT-Heuristics를 구현하였으며 비교 대상인 A-Greedy 기법 및 고정 연산자 순서 기법을 구현하여 성능 평가를 수행하였다. 다양한 환경에서는 실험 결과에서 보듯이 WT-Heuristics 기법은 시스템의 부담 없이 필요한 정보를 획득함으로써, 다른 기법들에 비하여 보다 효율적으로 스트림 데이터 처리한다.

참 고 문 헌

[1] D. Terry, D. Goldberg, D. Nichols, B. Oki, “Continuously Queries over Append-Only Databases,” In Proceedings of ACM SIGMOD Conference, 1992.

[2] D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, G. Seidman, M. Stonebraker, N. Tatbul, S. B. Zdonik, “Monitoring streams - a new class of data management applications,” In Proceedings of VLDB Conference, pp. 215-226, 2002.

[3] Niagara Project (http://www.cs.wis.edu/niagara)

[4] C. Cortes, K. Fisher, D. Pregibon, A. Rogers, “Hancock: a language for extracting signatures from data streams,” In Proceedings in ACM SIGKDD Conference, pp.9-17, 2000.

[5] A. Arasu, B. Babcock, S. Babu, M. Datar, K. Ito, R. Motwani, I. Nishizawa, U. Srivastava, D. Thomas, R. Varma, J. Widom, J., “Stream: The stanford stream data manager,” IEEE Data Engineering Bulletin, Vol.26, No.1, pp.19-26, 2003.

[6] J. M. Hellerstein, M. J. Franklin, S. Chandrasekaran, A. Deshpande, K. Hildrum, S. Madden, V. Raman, V., M. A. Shah, “Adaptive query processing: Technology in evolution,” IEEE Data Engineering Bulletin, Vol.23, No.2, pp.7-18, 2000.

[7] S. Babu, R. Motwani, K. Munagala, I. Nishizawa, J. Widom, “Adaptive ordering of pipelined stream filters,” In Proceedings of ACM SIGMOD Conference, pp.407-418, 2004.

[8] F. Fabret, H. A. Jacobsen, F. Llirbat, J. Pereira, K. A. Ross, D. Shasha, “Filtering algorithms and implementation for very fast publish/subscribe,” In Proceedings of ACM

SIGMOD Conference, pp.115-126, 2001.
[9] K. A. Ross, “Conjunctive selection conditions in main memory,” In Proceedings of PODS Conference, pp.109-120, 2002.
[10] J. Chen, D. J. DeWitt, F. Tian, Y. Wang, “Niagaracq: A scalable continuous query system for internet databases,” In Proceedings of ACM SIGMOD Conference, pp.379-390, 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,” In Proceedings of ACM SIGMOD Conference, pp.668, 2003.
[12] T. Brinkhoff, H. Kriegel, R. Scheneider, B. Seeger, “The R*-tree: An Effcient and Robust Access Method for Points and Rectangles,” In Proceedings of ACM SIGMOD Conference, pp.322-331, 1990.
[13] W. Pugh, “Skip lists: A probabilistic alternative to balanced trees,” Communication of ACM, Vol.33, No.6, pp.668-676, 1990.
[14] H. S. Lim, J. G. Lee, M. J. Lee, K. Y. Whang, I. Y. Song, “Continuous query processing in data streams using duality of data and queries,” In Proceedings of ACM SIGMOD Conference, pp.313-324, 2006.
[15] H. M. Deitel, “An Introduction to Operating Systems,” Addison-Wesley, 1990.
[16] B. Babcock, S. Babu, M. Datar, R. Motwani, “Chain : Operator scheduling for memory minimization in data stream systems,” In Proceedings of ACM SIGMOD Conference, pp.253-264, 2003.
[17] D. Carney, U. Cetintemel, A. Rasin, S. B. Zdonik, M. Cherniack, M. Stonebraker, “Operator scheduling in a data stream manager,” In Proceedings of VLDB Conference, pp.838-849, 2003.
[18] R. Avnur, J. M. Hellerstein, “Eddies: Continuously adaptive query processing,” In Proceedings of ACM SIGMOD Conference, pp.261-272, 2000.

민 준 기
e-mail : jkmin@kut.ac.kr
1995년 숭실대학교 전자계산학과(공학사)
1997년 한국과학기술원 전자전산학과 (공학석사)
2002년 한국과학기술원 전자전산학과 (공학박사)
2002년~2003년 한국과학기술원 연수연구원(Post Doc.)
2003년~2004년 한국과학기술원 초빙교수
2004년~2005년 전자통신연구원 선임연구원
2005년~현 재 한국기술교육대학교 인터넷미디어공학부 조교수
관심분야 : Query Processing, XML, Stream Data, Sensor Network