DOI: 10.3745/KIPSTD.2008.15-D.3.297

QUISIS: Interval Skip List를 활용한 질의 색인 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. 스트림 데이터 환경에서는 복수 개의 질의들이 미리 등록되고 후에 도착되는 데이터는 등록된 질의들에 의하여 평가된다. 따라서 질의 성능을 향상시키기 위하여, 스트림 데이터 처리 시스템을 위한 다양한 연속성 질의 색인 방법들이 제안되었다. 본 논문에서는 스트림 데이터를 위한 질의 색인에 대하여 다룬다. 일반적으로, 스트림 질의는 간격 조건식을 포함하고 있다. 따라서, 간격 조건식을 이용하여, 질의들을 색인화할 수 있다. 이 논문에서, 탐색 속도를 향상시키기 위하여, Interval Skip List를 수정한 효율적인 질의 색인 방법, QUISIS를 제안한다. QUISIS는 최근 데이터 값이 근 미래에 도착하는 값과 비슷하다는 지역성을 활용한다. 성능 평가를 통하여, 본 논문에서 제안하는 기법의 효율성을 보인다.

키워드 : 스트림 데이터, 색인, 질의 색인

QUISIS: A Query Index Method Using Interval Skip List

Min Jun-Ki†

ABSTRACT

Due to the proliferation of the Internet and intranet, new application domains such as stream data processing have emerged. Stream data is real-timely and continuously generated. In stream data environments, a lot of queries are registered, and then, the arrived data item is evaluated by registered queries. Thus, to accelerate the query performance, diverse continuous query index schemes have been proposed for stream data processing systems. In this paper, we focus on the query index technique for stream data. In general, a stream query contains the range condition. Thus, by using range conditions, the queries can be indexed. In this paper, we propose an efficient query index scheme, called QUISIS, using a modified Interval Skip Lists to accelerate search time. QUISIS utilizes a locality where a value which will arrive in near future is similar to the current value. Through the experimental study, we show the efficiency of our proposed method.

Key Words : Stream Data, Index, Query Index

1. 서 론

최근 데이터베이스 분야에서 연속 스트림 데이터를 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 12월 4일
수 정 일 : 1차 2008년 2월 25일, 2차 2008년 2월 27일
심사완료 : 2008년 2월 27일

스트림 데이터 처리 시스템(Stream Data Processing System)은 기존에 많이 사용되어온 데이터베이스와 같은 전통적인 데이터 처리 시스템들과는 서로 다른 특징들을 지니고 있다. 전통적인 데이터 처리 시스템들은 데이터를 안정적인 저장소에 저장하고 입력된 질의를 처리하기 위하여 저장소에 존재하는 데이터를 여러 번 검색하도록 한다. 그러나, 스트림 데이터 처리 시스템은 기존의 데이터 처리와는 다른 다음과 같은 특징을 지닌다.

첫째, 대량의 데이터가 지속적으로 실시간에 시스템에 도착된다.
둘째, 데이터의 검색을 위한 질의들은 사전에 등록되어 있다.
셋째, 고확장성을 보장하기 위하여 다수의 질의를 한꺼번에 처리해야 한다.
넷째, 전송된 데이터를 복수 번 검색할 수 없다.

따라서, 이러한 스트림 데이터를 처리하기 위하여서는 기존의 데이터베이스 관리 시스템과는 차별화된 구조의 데이터 관리를 위한 자료 모델 및 정보 처리 기법이 필요하다. 본 연구에서는 대량의 스트림 데이터를 처리하기 위한 관리 구조를 연구하는 것을 목적으로 한다. 특히, 위에서 언급한 바와 같이, 스트림 데이터 환경에서는 입력된 스트림 데이터를 다수의 질의들이 동시에 처리되어야 한다. 여기서 등록되어 있는 질의들이 상당히 많다면, 입력된 하나의 데이터에 대하여 모든 질의들을 수행하는 것은 시스템의 성능 저하를 유발한다 [7].

따라서, 입력된 데이터를 처리할 수 있는 질의 집합을 찾아서 이러한 질의들만을 수행하게 하면 시스템의 성능을 향상 시킬 수 있다. 이러한 상황에서의 문제는 어떻게 입력된 데이터를 처리할 수 있는 질의 집합을 효율적으로 찾아내는가 이다. 이러한 문제를 해결하기 위하여 최근에 질의 인덱스 기법에 대한 연구가 활발히 진행되고 있다[8].

전통적인 데이터 처리 시스템들은 안정된 저장소에서 관리되고 있는 데이터를 효과적으로 검색하기 위하여 B-Tree, R-Tree와 같인 인덱스 구조를 활용하였다. 그러나 스트림 데이터 환경에서는 데이터 대신에 질의들이 사전에 등록되어 있고 데이터들은 지속적으로 시스템에 도착되는 특성을 지닌다.

예를 들어, 인텔리전스 빌딩에는 각 층별, 방 별로 온도 센서들이 부착되어 있어서, 실시간으로 측정된 실내 온도를 스트림 데이터 처리기로 보내게 된다. 스트림 데이터 처리기에는 사전에 측정 온도가 25도 이상일 경우 에어컨을 작동시키라는 상황 인지 (Context Aware) 서비스가 등록되어 있다고 하자. 이 경우 측정된 온도가 25도 이상인지 아닌지를 평가하고 이상일 경우 상황 인지 서비스를 호출하는 질의가 등록 수행되어야 한다.

여기서 만약 측정된 온도를 평가 수행하기 위한 질의가 수백만 개에 달한다면, 입력된 하나의 데이터에 대하여 수백만 개의 질의들이 수행되어야 함으로 시스템의 성능이 저하된다. 따라서, 등록된 질의들을 인덱스화하여 도착되는 데이터를 처리할 수 있는 질의 집합만을 추출하여 이러한 질의들만을 수행하도록 하여 질의 부담을 감소시키는 방법이 필요하다.

일반적인 스트림 데이터 질의들은 간격(Interval)으로 표현되는 질의 조건을 포함하고 있다. 따라서, 이러한 질의 조건들을 활용하여 질의 색인을 생성할 수 있다. 예를 들어, 증권 거래 정보에서 많은 사용자는 어떤 데이터가 특정 값을 가지는가 보다는 그 데이터가 어떤 의미를 가지는 영역 안에 들어가는가 아닌가에 더 관심을 가진다. 즉, 주식 투자자는 자신이 산 주식이 20,000원에서 30,000원 사이에 들어가면 알려 주기를 바랄 것이다.

기존의 스트림 데이터 처리기는 성능 개선 방안으로 간격 질의를 색인화하는 방식에 대한 깊은 연구를 하지 않았다. 따라서, 기존의 데이터 인덱스 기법으로 사용되었던, B-Tree나 R-Tree를 질의 색인으로 활용하였다. 이러한 색인 기법들은 임의의 영역 안에 포함되는 값들을 찾기에는 효과적인 구조로 되어 있다. 그러나 스트림 데이터 처리와 같이 하나의 값이 들어 왔을 때, 그 값을 포함하는 모든 영역들을 찾아내는 데는 비효율적이다.

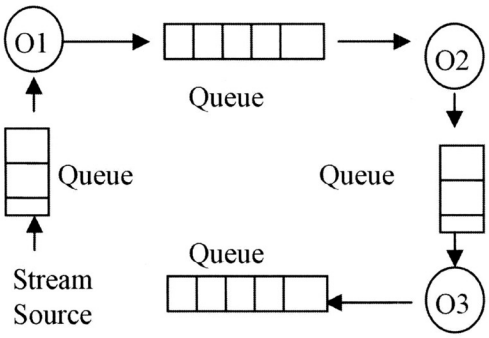
따라서 본 논문에서는 간격 조건으로 나타낼 수 있는 질의들을 위한 효율적인 질의 색인 방안, QUISIS(Query Index for Stream data using Interval Skip Lists)를 제안한다. QUISIS는 Interval Skip List를 기반으로 하고 있으며, 현재 도착한 데이터 값이 미래의 데이터 값과 유사하는 지역성(Locality)를 활용한다.

본 논문의 나머지 부분의 구성은 다음과 같다. 2장에서는 스트림 데이터 분야에서 각광 받고 있는 중요한 연구 이슈들을 살펴 보고 3장에서는 본 논문에서 제안하는 효율적인 질의 색인 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 연산자 스케쥴링 (operator scheduling) [9, 10] 이다. 다음의 (그림 1)과 같이 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 [11] 등을 활용할 수 있다. Babcock 등[9]은 메모리 사용량을 최소하는 근접 최적(near optimal)인 Chain이라는 스케쥴링 기법을 제안하였다. 또한 Carney 등은 [10]에서는 최소-비용(min-cost), 최소-대기(min - latency),



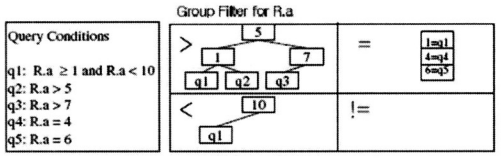
(그림 1) 간단한 질의 구조

최소 메모리(min-memory) 등 다양한 목적에 따른 연산자 스케쥴링 기법들을 제안 하였다.

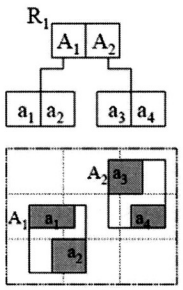
또 다른 연구 분야는 연산자 순서 문제[12, 13]이다. 연산자 순서 문제는 하나의 입력 튜플에 대하여 여러 개의 연산자들을 적용할 수 있을 때 어떠한 순서로 적용해야 가장 효율적인가를 정하는 문제로 (그림 1)과 같이 하나의 질의가 O1, O2, O3으로 구성될 때, O1-O2-O3 순서로 할지, O2-O1-O3 순서로 할지 등을 정하는 것이다. 이에 반하여 연산자 스케줄링은 이러한 연산자 적용 순서가 정해 졌을 때, 스트림으로 들어오는 데이터 특성 때문에 복수개의 연산자가 실행 가능해질 경우 어떠한 연산자를 수행 시킬지를 실시간에 정하는 문제이다.

마지막으로 본 논문의 연구 주제인 질의 색인 분야가 있다. 1장에서 언급한 바와 같이 스트림 데이터 환경에서는 질의를 미리 등록하고 추후 데이터가 전송될 때 질의를 수행하게 된다. 여기서 등록된 질의 개수가 아주 많은 경우 새로 도착된 데이터를 처리할 수 있는 질의들을 찾아내는 것 자체가 스트림 데이터 처리 시스템의 성능에 큰 부담이 된다. 따라서 질의들의 조건 (predicate)을 색인화 하여 새로 도착된 데이터를 처리할 수 있는 질의를 빠르게 찾을 수 있다[14].

Niagara CQ [3]와 Telegraph CQ [15]에서는 질의 색인 기법으로 이진 탐색 트리 (binary search tree)을 변형한 IBS (interval binary search)[14]를 이용하여 사용한다. 이진 탐색 트리에서 노드를 찾기 위하여 트리의 루트에서 시작해서 찾고자 하는 노드를 찾을 때까지 아래 단계로 내려간다. 찾는 노드보다 큰 노드를 만나면 왼쪽 포인터를 따라가고 작은 노드를 만났을 때는 오른쪽 포인터를 찾아가도록 한다. (그림 2)는 IBS의 예제이다. (그림 2)에서 보는 바와 같이 하나의 질의 색인은 네 개의 자료구조로 구성되어 있다: 작다 이진 트리, 크다 이진 트리, 동등 해쉬 테이블, 비동등 해쉬 테이블. 데이터 값이 들어 올 때, 이진 트리들과 해쉬 테이블들은 입력된 값을 이용하여 관련 없는 질의들을 제외시킨다. 이 접근 방식은 두 개의 경계 조건(boundary condition)을 가지는 일반적인 영역 질의에는 적합하지 않다. 이는 각 경계 조건은 독립된 이진 탐색 트리에 등록됨으로, 각 이진 탐색의 결과에는 불필요한 결과들이 포함되어 후처리 작업



(그림 2) 이진 탐색 트리를 이용한 질의 색인 예

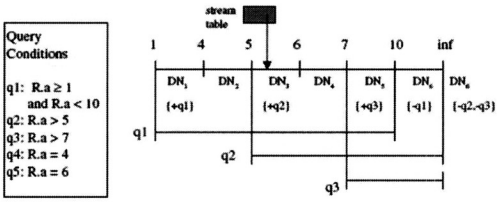


(그림 3) R-tree 예

을 수행해야 하기 때문이다.

질의 색인으로써 R-tree를 이용할 수도 있다. R-tree[16, 17]는 B-Tree로부터 유도되는 n-차원 데이터를 담을 수 있도록 설계된 계층적 인덱스 이다. 트리의 각 노드는 자신의 자식 노드들을 포함하는 가장 작은 n-차원 사각형에 해당한다. 단말 노드는 데이터베이스에 있는 하나의 데이터를 포함하는 MBR(minimum bounded rectangle)을 표현한다. (그림 3)은 R-Tree의 구조를 보여준다. 간격으로 표현되는 질의도 2차원 데이터 형태로 표현할 수 있음으로 R-Tree를 이용하여 색인화할 수 있다. 그러나, 이 R-tree는 색인화된 데이터들의 중첩(overlap)이 심할 경우 복수 개의 자식 노드들을 모두 검색해야 함으로써 그 성능이 좋지 않다고 알려져 있다. 최근에 [18]에서는 MLGF를 이용한 질의 색인 기법을 제안하였다.

최근에, 영역 질의 색인을 위하여 BMQ-Index[19]가 제안되었다. BMQ-Index는 두 개의 데이터 구조로 이루어 있다: DMR 리스트, 스트림 테이블. DMR 리스트는 <DN_1, DN_2, … DN_n, DN_{n+1}> 형태의 DMR 노드들의 리스트 이다. 등록된 질의들의 집합을 Q라고 하고, 하나의 질의 $q_i \in Q$의 조건이 $l_i < x < u_i$인 x에 대한 것이라고 할 경우, q_i를 (l_i, u_i)로 나타낼 수 있다. 이 경우 l_i는 질의 q_i의 질의 조건 중 낮은 경계값을 나타내고, u_i를 높은 경계값을 나타낸다. 하나의 DMR 노드, DN_j는 <DR_j, +$DQSet_j$, -$DQSet_j$> 형태를 지닌다. DR_j는 특정 영역 (b_{j-1}, b_j)를 나타내며, +$DQSet_j$은 낮은 경계값 l_k가 b_{j-1}인 질의들의 집합을 나타내고 -$DQSet_j$은 높은 경계값 u_k 가 b_{j-1}인 질의들의 집합을 나타낸다. 스트림 테이블은 가장 최근에 입력된 데이터 값을 포함하는 DMR 노드를 가리킨다. (그림 4)는 BMQ-Index의 예를 보여준다.



(그림 4) BMQ-Index의 예

QSet(t)를 시간 t에 도착된 스트림 데이터 값 v_t에 대한 질의 집합이라고 하고 v_t는 DN_j의 영역인 DR_j안에 존재한다고 하자. 또한 시간 t+1에 도착하는 데이터 값 v_{t+1}은 DN_h의 영역 DR_h안에 존재한다고 하자. 즉 $b_{j-1}<= v_t < b_j$이고 $b_{h-1}<= v_{t+1}<b_h$이다. 이 경우 QSet(t+1)은 다음과 같이 얻어진다.

if j < h, $QSet(t+1) = QSet(t) \cup [\cup^{h}_{i=j+1}+DQSeti] - [\cup^{h}_{i=j+1} -DQSeti]$

if j > h, $QSet(t+1) = QSet(t) \cup [\cup^{h+1}_{i=j}+DQSeti] - [\cup^{h+1}_{i=j} -DQSeti]$

if j = h, QSet(t+1) = QSet(t)

BMQ-Index는 현재 데이터와 그 다음 도착하는 데이터가 매우 유사할 경우 소수의 DMR 노드들만 검색하여 대응되는 질의 집합을 찾을 수 있다. 그러나 BMQ-Index는 다음과 같은 문제를 가지고 있다. 첫 번째로, 앞으로 도착될 데이터가 현재의 데이터와 상당히 다르면, 많은 DMR 노드들인 선형 탐색 형태(linear search fashion)으로 검색되어야 한다. 두 번째로, BMQ-Index는 (l, u) 형태의 조건들만을 지원하고 일반적인 형태인 [l, u] 또는 (l, u] 형태의 조건들은 지원하지 못한다. 따라서, 그림에서 나타난 바와 같이 질의 q4와 q5는 BMQ-Index에 등록되지 못했다. 더욱이, BMQ-Index는 경계 조건에서 올바로 동작되지 못한다. 예를 들어, 만약 v_t가 5.5 일 때, QSet(t)는 {q1 q2} 이다. 이때 v_{t+1}이 5일 경우, QSet(t+1)은 위의 수식에 따라서 {q1, q2}가 된다. 그러나, 올바른 QSet(t+1)는 {q1} 이다.

3. QUISIS

본 논문에서는 도착된 스트림 데이터에 대하여 효율적으로 질의들을 검색할 수 있는 질의 인덱스 구조에 대하여 연구하였다. 증권 데이터나 온도 데이터와 같은 스트림 데이터는 일반적으로 가장 최근에 들어온 데이터가 앞으로 입력되는 데이터와 유사한 값을 지닌다는 데이터 근접성을 지닌다. 따라서, 이러한 근접성을 활용하여 보다 효율적인 질의 인덱스 방안을 도출하고자 한다.

본 절에서는 본 연구를 통하여 제안하는 질의 색인 구조에서 대하여 설명한다. 제안된 질의 색인 구조는 Interval Skip List를 기반으로 하고 있다. 따라서 우선 Interval Skip List에 대하여 설명한다.

3.1 Interval Skip List

Interval Skip List[20] 는 각 노드가 하나 이상의 앞방향 포인터(forward pointer)를 가지고 있다는 점을 제외하고는 연결 리스트와 유사하다. 각 노드의 앞 방향 포인터의 개수 k는 각 노드의 최대 레벨(level)이라고 하며, 각 노드의 최대 레벨은 다음과 같은 확률식에 의하여 결정된다.

$$P(k) = \begin{cases} 0 & \text{for } k < 1 \\ (1-p)\cdot p^{k-1} & \text{for } k \geq 1 \end{cases}$$

여기서 p가 1/2일 경우 노드 레벨의 분포는 다음과 같이 할당된다: 1/2의 노드들은 하나의 앞방향 포인터를 가지며, 1/4의 노드들은 두 개의 앞방향 포인터를 가지며,... 등. 즉, Interval Skip List에서 노드 생성 시 0에서 1사이의 난수를 발생 시켜 그 값이 1/2보다 작으면 최대 레벨이 1인 노드를 만들고 1/2에서 3/4(=1/2+1/4) 사이면 최대 레벨이 2인 노드를 생성하는 식으로 노드의 최대 레벨을 결정한다.

한 노드의 최대 레벨을 k라 할 때, k개의 앞방향 포인터들 중 i번째 앞방향 포인터 (이를 레벨 i라고 한다.)는 최대 레벨이 i이거나 i보다 큰 최대 레벨을 지니는 다음 노드를 가리킨다. 또한, 노드와 앞방향 포인터는 대응되는 간격(interval)에서 유지되는 정보를 관리하기 위하여 마커(marker)들을 가진다. 즉, 마커는 해당 간격이 나타내는 객체 정보(객체 식별자)들로 구성된다.

간격 I= (A,B)가 인덱스에 추가되어야 한다고 가정하면, 간격 I의 끝 포인트 A,B가 리스트에 존재하지 않으면 포인트 A, B는 리스트에 노드로 삽입되어야 한다. 위에서 언급한 바와 같이 각 노드의 레벨은 위의 확률식에 의하여 결정된다. 여기서, 기존의 인덱스에 값 X를 가지는 노드의 앞방향 포인터가 값 Y를 지니는 노드를 가리키고 있을 때 (즉, X < Y 일 경우), 다음과 같은 조건을 만족할 경우 간격 I를 위한 식별자가 (X,Y)의 간선에 마커로써 추가된다.

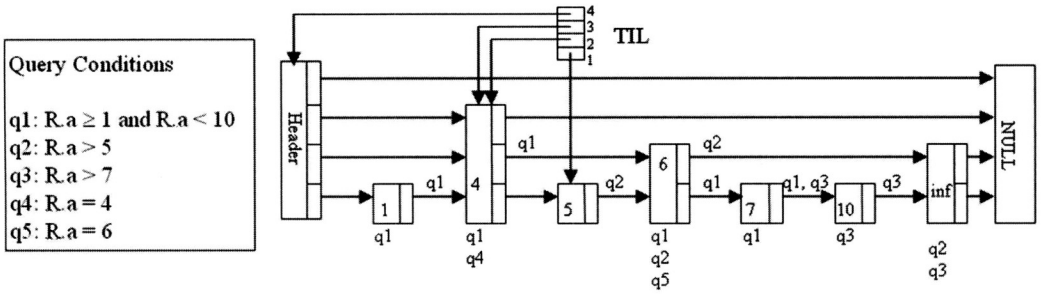
-- 포함: 간격 I는 간격 (X,Y)를 포함한다.
-- 최대성: 간격 (X,Y)를 포함하면서, 간격 I에 포함되는 간격(X′,Y′)에 대응되는 앞방향 포인터는 존재하지 않는다.

추가적으로, 만약, 간격 I를 위한 마커가 간선 위에 존재하고, 그 간선에 연결된 노드의 값이 간격 I에 포함되어 있다면 그 노드도 간격 I를 위한 마커를 지닌다. 이 때, 노드 위에 존재하는 마커는 eqMarker 라고 한다.

Interval Skip List의 시간 복잡도는 O(logN)으로 알려져 있다. 여기서 N은 인덱스된 간격들의 수이다. 또한 Interval Skip List의 공간 복잡도는 O(N logN)으로 B-Tree와 동일하다.

3.2 시간 근접성을 이용한 질의 인덱스 기법

주어진 탐색 값에 대하여 Interval Skip List는 헤더부터 탐색을 시작한다. 스트림 데이터 환경에서는, 미래의 데이터 값이 현재의 데이터 값과 유사하는 지역성을 지닌다. 따라



(그림 5) 제안된 질의 색인 구조

서 Interval Skip List를 기반으로 이러한 지역성을 이용하는 새로운 질의 색인 방안을 제안한다.

(그림 5)은 현재 접근한 데이터 값의 5.5 일 때의 제안된 색인기법의 모습을 보여준다. 현재 데이터 값에 의하여 방문하고 있는 간격들의 정보를 유지하기 위하여 시간 관심 리스트(temporal Interesting List: TIL)를 사용한다.

TIL은 MAX 레벨부터 1 레벨까지의 노드들을 기록하고 있으며 관리되고 있는 노드의 레벨 i의 앞방향 포인터는 현재 데이터 값을 포함하는 간격을 나타낸다. 예를 들어, 그림 5의 TIL의 레벨 1에 의하여 표현되는 간격 [5, 6]은 현재 데이터 값 5.5를 포함하며, TIL의 레벨2에 의하여 표현되는 간격 [4, 6]도 현재 데이터 값 5.5를 포함한다.

(그림 5)에서 Header는 가장 작은 값을 나타내고 NULL은 가장 큰 값을 나타낸다고 해석한다. 따라서, 일반적인 숫자 시스템과 달리, Header는 -inf 보다 작고, NULL은 inf 보다 크다.

이때, TIL에 의하여 표현되는 간격들은 다음과 같은 특성을 만족한다.

특성1. 레벨 i의 TIL 노드에 의하여 표현되는 간격은 레벨 i+1의 TIL 노드에 의하여 표현되는 간격에 포함된다.

예를 들어, 레벨 1의 TIL 노드에 의하여 표현된 간격 [5,6)은 레벨 2의 TIL노드에 의한 간격 [4,6)에 포함된다. 이러한 특성을 이용하여 원래 Interval Skip List에 비하여 효율적으로 탐색 공간을 줄일 수 있다.

TIL을 이용하여 위하여, 특성 1을 이용한 findQuery() 프로시져를 만들었다. findQuery() 프로시져의 개관은 (그림 6)과 같다. 기본적으로 findQuery() 프로시져의 동작은 레벨1의 TIL 노드와 새로 입력된 데이터 값 (즉, key) 사이의 조건에 따라서 달라진다.

만약 key가 레벨 1의 TIL 노드 n의 값 v1과 같다면 (라인 1-3), 기존의 질의 집합 QSet에는 간격 (v1, -)으로 표현되는 질의들을 포함하고 있을 수 있음으로, n에서 시작되는 간선들의 모든 마커들을 기존의 질의 집합 QSet에서 제거한다. 대신, 노드 n의 eqMarker를 QSet에 포함시킨다. 이는 레벨 1의 TIL 노드 n의 eqMarker는 v1을 포함하는 간격들

```
QSet      // a query set for previous key
TIL       // a List points to the nodes of QUISIS
begin
1. if(TIL[1]->value = key){
2.          for(i := TIL[1]->level; i >= 1; i--) QSet := QSet-TIL[1]
            ->markers[i]
3.          QSet := QSet U TIL[1]->eqMarker
4.} else if(TIL[1]->value < key and
            (TIL[1]->forward[1]=NULL or key < TIL[1]->forward[1]
            ->value)){
5.          QSet := QSet-TIL[1]->eqMarker
6.          for(i := TIL[1]->level; i >= 1; i--) QSet := QSet U
            TIL[1]->markers[i]
7. } else{
8.          QSet := QSet - TIL[1]->eqMarker
9.          if(TIL[1]->forward[1] = NULL or key < TIL[i]->
            forward[1]->value){
10.                   for(i := 1; i <= maxLevel; i++)
11.                             if(TIL[i]->forward[i] = NULL
            or key < TIL[i]->forward[i]->value) break
12.                             else QSet = QSet-TIL[i]->
            markers[i]
13.         }else{
14.                   for((i := 1; i <= maxLevel; i++)
15.                             if(TIL[i]= Header and key >=
            TIL[i]->value) break
16.                             else QSet = QSet - TIL[i]->
            makrsers[i]
17.         }
18.         anode := TIL[--i]
19.         while(i >= 1){
20.                   while(anode->forward[i] != NULL and anode
            ->forward[i]->value <= key)
                              anode = anode->forward[i]
21.                   if(anode != Header and anode->value !=
            key) QSet := Qset U anode->marker[i]
22.                   else if(anode != Header) QSet = QSet U
            anode->eqMarker[i]
23.                   TIL[i] := anode;
24.                   i := i-1
25.         }
26.}
27.return QSet
end
```

(그림 6) findQuery() 알고리즘

을 포함하는 질의들을 나타내기 때문이다. 예를 들어, (그림 5)와 같이 현재 데이터 값이 5.5이고 QSet이 {q1, q2}라고 할 때, 새로운 데이터 값 5가 들어오면, Line2에 의하여 {q2}가 제거되고 노드 5의 eqMarker 는 공집합이므로 질의 집합 QSet은 {q1}이 된다.

만약 key 가 레벨 1의 TIL 노드에 의하여 표현되는 간격 (v1, u1) 안에 포함될 경우 (라인 4-6), QSet에서 노드 n의 eqMaker에 속한 질의들은 모두 제거된다. 이는 QSet안에 (-.v1]이 포함되었을 수 있기 때문이다. 대신 노드 n에서 시작되는 모든 간선들의 마커가 추가된다.

만약, key가 간격 (v1, u1) 안에 존재하지 않는다면, findQuery 프로시져는 key를 포함하는 레벨 i의 TIL에 의한 간격 [vi, ui]를 찾는다 (라인 8-17). 이 단계는 key가 u1보다 크거나 같은 경우 (라인 9-12)와 key가 v1보다 작을 경우(라인 13-17)로 나누어진다. 또한 이 단계에서 레벨 1에서 i-1의 TIL에 의하여 표현되는 간선들의 마커들은 QSet에서 제거된다 (라인 12와 16). 그리고 난 후, 프로시져는 값인 vi인 노드 (즉 anode)로부터 시작하여 질의들을 수집한다. 이 경우, 레벨 i의 anode로부터 시작되는 간선의 마커들은 이미 QSet에 포함되어 있으므로 레벨 i를 감소시킨다 (라인 18). 감소된 레벨 i값으로 나타나는 간선으로 표현되는 간격이 key를 포함하고 있지 않을 경우 레벨 i의 anode의 앞방향 포인터가 가리키는 다음 노드로 넘어간다 (라인 21). 만약 anode의 값이 key와 같다면 anode의 eqMaker를 QSet에 추가한다 (라인 22). 그렇지 않다면 anode의 앞방향 포인터가 나타내는 간선의 마커를 QSet에 추가한다(라인 21). 그리고, anode는 TIL의 I레벨 노드로 설정된다 (라인 23). 이후 레벨 i는 i-1로 감소되고 다시 key가 감소된 레벨 i로 가리키는 anode와의 관계를 이용하여 QSet에 질의 정보를 수집한다. 이러한 탐색 프로시져는 레벨 i가 1이 될 때까지 지속된다.

예를 들어, (그림 5)와 같이 현재 데이터가 5.5, QSet이 {q1,q2}이고, 새로운 데이터 값이 13이라고 할 때, TIL[3]에 의하여 표현되는 [4,Null)이 13을 포함한다. 따라서, TIL[1]과 TIL[2]에 의한 마커 {q2}, {q1}은 QSet에서 제거된다. 그리고, 레벨 2를 가지고 노드 4부터 질의들을 수집한다. [4,6]은 13을 포함하지 않음으로 프로시져는 다음 노드 6의 레벨 2에 의하여 표현되는 간격 [6,NULL]을 검사한다. [6,Null]은 13을 포함하므로, 마커 {q2}가 포함된다. 그리고 TIL[2]는 노드 6을 가리킨다. 그리고 프로시져는 노드 6의 레벨 1 앞방향 포인터부터 탐색을 시작한다. [10, inf]는 13을 포함하므로 {q3}인 QSet에 포함된다. 결국, QSet은 {q2,q3}가 된다.

데이터 근접성을 이용한다는 점에서, 우리가 제안한 QUISIS는 BMQ-Index와 유사하다. 그러나, QUSIS는 Interval Skip List에 기반하고 있어서, 일반적인 경우에, BMQ-Index보다 QUISIS가 보다 효율적이다. 우리의 실험 결과는 QUISIS의 효율성을 보여 준다.

4. 실 험

본 절에서는 기존의 질의 인덱스 기법들과 비교하여 본 연구에서 제안한 질의 인덱스 방안의 효율성을 보인다. 비교 대상 질의 인덱스로서 BMQ-Index[8]와 Interval Skip List를 사용하였다. 실험은 윈도우-XP환경의 1GByte 메모리를 갖춘 Pentium-IV-3.0Ghz CPU 머신에서 수행되었다. 또한 실험을 위하여 다양한 설정값을 이용한 합성 데이터를 이용하였다. 비교 대상인 BMQ-Index는 경계 조건에서 올바로 동작하지 않음으로 실험을 위하여 수정된 BMQ-Index를 구현하였다. 위에서 언급한 바와 같이, Interval Skip List의 공간 복잡도는 O(n log n)으로 B-Tree와 동일하며 이는 충분히 시스템에서 저장 관리할 수 있는 복잡도이다. 따라서, 본 실험에서는 제안된 인덱스 방안에 대한 속도만을 비교하였다.

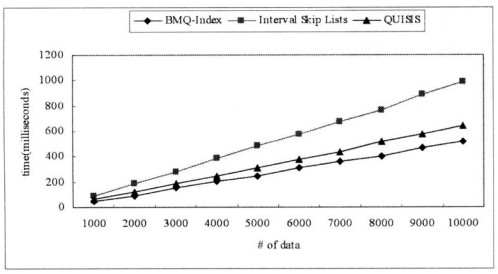
기본 실험 환경은 <표 1>에 나타나 있다.

실험에 사용된 데이터는 <표 1>에 나타난 것처럼 1에서 1,000,000 사이의 값을 지니며, 질의는 100,000개를 미리 사전에 등록시켜 색인화 하였다. <표 1>의 질의 간격은 질의 조건의 간격을 애트리뷰트 영역으로 평준화하여 표시한 것으로 각 질의 간격의 낮은 경계값은 1에서 1,000,000사이의 값이 되도록 했으며 높은 경계값은 낮은 경계값+1,000(0.1%)가 되도록 하였다.

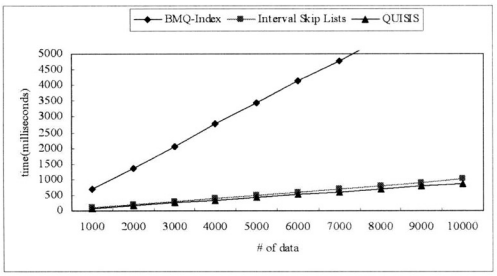
스트림 데이터 환경에서는 질의가 미리 등록되어 있고 데이터가 지속적으로 생성됨으로 본 실험에서도 데이터를 지속적으로 생성시켜 이에 대한 질의 인덱스 성능을 측정하였다. 데이터의 개수는 <표 1>에 나온 것처럼 10,000를 생성하였다. 또한 데이터를 생성시키는 데 있어서 근접성을 만들기 위하여 변동 레벨(FL)을 사용하였다. 변동 레벨(FL)은 연속된 두 스트림 테이타의 평균 차이를 데이터 영역으로 평준화하여 나타낸 것이다. 따라서, FL 값이 작으면, 연속된 스트림 데이터의 차이가 작아짐으로 근접성이 심하게 나타나고 FL값이 커질수록, 연속된 스트림 데이터의 차이가 커짐으로 근접성이 적게 나타나게 된다.

〈표 1〉 기호 테이블

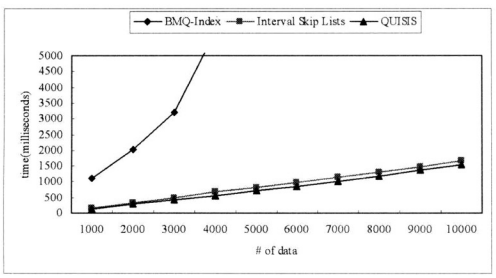
파라메터	값
데이타 영역	1~1,000,000
질의 수	100,000
질의 간격의 크기	0.1% (= 1,000)
데이터 수	10,000
변동 레벨(FL)	0.01% (= 100) ~ 1%(10,000)



(그림 7) FL = 0.01% 일 때의 실험 결과



(그림 8) FL = 0.1% 일 때의 실험 결과



(그림 9) FL = 1% 일 때의 실험 결과

(그림 7)은 변동 레벨(FL)이 0.01%일 때의 실험 결과를 보인다. 이 경우에는 BMQ-Index가 가장 좋은 성능을 보인다. 이는 BMQ-Index가 간단한 자료 구조로 이루어져 있기 때문이다. 즉, 위에서 언급한 바와 같이 BMQ-Index는 현재 데이터가 바로 직전 데이터와 비슷하면 인접된 DMR 노드만을 이용하여 빨리 구할 수 있다. 그러나 (그림 8과 9)처럼 FL이 0.1%일 때와 FL이 1%일 경우에 가장 나쁜 성능을 보여주고 있다. 또한 (그림 7)에서 우리의 QUISIS가 Interval Skip List 보다 성능이 더 좋음을 보이고 있다.

(그림 8과 9)에 나타난 다양한 환경에서의 실험 결과에서 보듯이, 본 연구에서 제안한 질의 색인 기법은 FL값이 0.01%일 때를 제외하고는 가장 좋은 성능을 보여 주고 있다. BMQ-Index에서는 앞으로 올 데이터가 기존의 데이터와 다를 경우 많은 DMR노드들을 검색해야만 한다. 그러나, BMQ-Index와 반대로 본 논문에서 제안한 질의 색인 기법인 QUISIS는 TIL을 이용하여 효율적으로 질의 집합을 검색하며 헤더부터 검색하는 부담을 제거하여 거의 모든 경우에 뛰어난 성능을 보여주고 있다.

5. 결 론

스트림 데이터는 실시간적이고 연속적으로 생성된다. 스트림 데이터 환경에서는 복수 개의 질의들이 미리 등록되고 후에 도착되는 데이터 값은 등록된 질의들에 의하여 평가된다. 일반적인 스트림 데이터 질의들은 간격(Interval)으로 표현되는 질의 조건을 포함하고 있다. 따라서, 이러한 질의 조건들을 활용하여 질의 색인을 생성할 수 있다. 따라서 본 논문에서는 입력된 데이터 값을 처리할 수 있는 질의 집합을 효율적으로 찾아내는 질의 색인 기법인 QUISIS를 제안하였다.

QUISIS는 간격을 효과적으로 색인화하는Interval Skip List를 기반으로 하고 있다. 또한 미래의 데이터와 현재의 데이터가 유사하다는 근접성을 유지하기 위하여 TIL이라는 구조를 포함하였으며 이 유사성을 위한 검색 방법을 제시하였다. 제안된 질의 색인 방안의 성능을 보이기 위하여 합성 데이터를 이용하여 다양한 질의 환경의 실험을 수행하였으며, 실험 결과는 본 연구에서 제안한 질의 색인 구조가 기존의 질의 색인 방법에 비하여 뛰어남을 보여 주었다.

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. Choi, J. Lee, S.M. Kim, S. Jo, J. Song, Y.J. Lee, "Accelerating Database Processing at e-Commerce Sites,"

In Proceedings of International Conference on Electronic e-Commerce and Web Technologies. (2004)

[8] K.A. Ross, "Conjunctive selection conditions in main memory," In Proceedings of PODS, 2002.

[9] 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.

[10] 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.

[11] H. M. Deitel, 'An Introduction to Operating Systems,' Addison-Wesley, 1990.

[12] R. Avnur, J. M. Hellerstein, "Eddies: Continuously adaptive query processing," In Proceedings of ACM SIGMOD Conference, pp.261-272, 2000.

[13] 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.

[14] S. Madden, M.A. Shah, J.M. Hellerstein, V. Raman, "Continuously adaptive continuous queries over streams," In Proceedings of ACM SIGMOD Conference, 2002.

[15] 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.

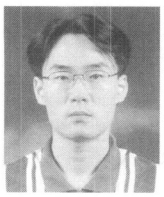
[16] A. Guttman, "R-Trees: A Dynamic Index Structure for Spatial Searching," In Proceedings of ACM SIGMOD Conference, 1984.

[17] T. Brinkhoff, H. Kriegel, R. Scheneider, B. Seeger, "The R*-tree: An Efficient and Robust Access Method for Points and Rectangles," In Proceedings of ACM SIGMOD Conference, 1990.

[18] 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.

[19] J. Lee, Y. Lee, S. Kang, H. Jin., S. Lee, B. Kim, J. Song, "BMQ-Index: Shared and Incremental Processing of Border Monitoring Queries over Data Streams," In Proceedings of International Conference on Mobile Data Management (MDM'06), 2006.

[20] E.N. Hanson, T. Johnson, "Selection Predicate Indexing for Active Databases Using Interval Skip Lists," Information Systems 21,1996.



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