Journal of Korea Multimedia Society Vol. 10, No. 11, November 2007(pp. 1367-1383)

센서 네트워크에서 스트림 데이터 질의의 효율적인 처리를 위한 다중 질의 색인 기법

이민수[†], 김연정[††], 윤혜정[†††]

요 약

센서 네트워크는 스스로 감지하고 계산하고 무선으로 서로 통신할 수 있는 기능을 갖춘 센서들로 이루어진 네트워크이다. 센서 네트워크의 특징들로는 네트워크가 자체적으로 관리가 되어야 한다는 것과 배터리 전원이여서 전력의 효율성을 크게 고려해야 한다는 것이 있다. 센서 네트워크에서 생성되는 많은 양의 연속적인 데이터에 대하여 여러 개의 질의들을 동시에 처리해야 하는 경우에 전력의 효율성을 극대화시켜야 한다. 본 연구에서는 센서 네트워크에서 감시 목적의 미리 정의된 다중 질의들에 대해 색인을 두어 다중 질의 처리 성능을 높이고 메모리와 전력을 효율적으로 사용할 수 있는 기법을 제안한다. 공간 색인 기법 중에서 이진 탐색트리에 기반한 데이터 구조로서 각 레벨별로 차원이 반복되어 각 차원을 분할시키는 k-d 트리와, 공간을 계층적 구조로 자르며 겹침 관계를 줄인 R-트리의 변형인 R+-트리를 기반으로 하여 이들의 응용 및 융합을 통해 다중 질의를 색인하는 새로운 트리인 SMILE 트리를 제안한다. 질의들에 대한 SMILE 트리를 구성하여 센서 네트워크에서 생성되는 스트림 데이터에 대하여 관련된 질의를 탐색하도록 하면 질의를 순차 탐색하는 것과 비교하여 경우에 따라서는 평균 탐색시간을 약 50% 정도로 줄일 수 있다.

Multi-query Indexing Technique for Efficient Query Processing on Stream Data in Sensor Networks

Lee Minsoo[†], Kim Yearn Jeong[††], Yoon Hyejung[†††]

ABSTRACT

A sensor network consists of a network of sensors that can perform computation and also communicate with each other through wireless communication. Some important characteristics of sensor networks are that the network should be self administered and the power efficiency should be greatly considered due to the fact that it uses battery power. In sensor networks, when large amounts of various stream data is produced and multiple queries need to be processed simultaneously, the power efficiency should be maximized. This work proposes a technique to create an index on multiple monitoring queries so that the multi-query processing performance could be increased and the memory and power could be efficiently used. The proposed SMILE tree modifies and combines the ideas of spatial indexing techniques such as k-d trees and R+-trees. The k-d tree can divide the dimensions at each level, while the R+-tree improves the R-tree by dividing the space into a hierarchical manner and reduces the overlapping areas. By applying the SMILE tree on multiple queries and using it on stream data in sensor networks, the response time for finding an indexed query takes in some cases 50% of the time taken for a linear search to find the query.

Key words: Sensor Network(센서 네트워크), Stream Data(스트림 데이터), Spatial Indexing(공간 색인 기법), k-d tree(k-d 트리), R+-tree(R+-트리)

1. 서 론

최근에 고정되고 한정된 데이터를 처리하는 방법을 벗어나 새롭게 지속적인 데이터 스트림을 처리하는 연구가 널리 진행되고 있다. 기존 데이터를 처리하던 데이터베이스 관리 시스템(DBMS: database management system)은 급격하고 지속적인 방대한 양의 데이터를 처리할 수 있게 설계되어 있지 않으며 연속 질의(continuous queries)를 지원하지 못하므로 방대한 양의 스트림 데이터를 처리하는데 적합한 시스템으로 데이터 스트림 관리 시스템(DSMS: data stream management system)이 개발되고 있다.

데이터 스트림 관리 시스템은 기존의 데이터가 디스크나 메모리에 있어서 질의를 통해 원하는 데이터를 찾고 관리하던 데이터베이스 관리 시스템과는 다르게 데이터가 하나 또는 하나 이상의 지속적인 스트림 데이터 형태로 들어오면 이 데이터에 대해 질의를 수행하고 관리하는 시스템이다. 데이터 스트림 관리 시스템에서는 데이터의 정확성이 약간 떨어지더라도 정답에 근접한 결과를 제공하여 상황에 따라 적합하게 질의 처리가 적응적으로 이루어진다는 특징이 있다. 데이터 스트림 관리 시스템에서는 사용자 또는 응용이 먼저 질의를 등록하게 되고 그 이후 지속적으로 들어오는 하나 이상의 스트림 데이터를 입력으로 등록된 질의를 수행하게 된다. 이러한 데이터 스트림 관리 시스템에 해당되는 응용들로는 증권 데이터 응용, 네트워크 감시 시스템, 센서 네트워크 데이터베이스 등이 있다.

데이터 스트림 관리 시스템 중에서 가장 대표적인 센서 네트워크 데이터베이스는 스스로 감지하고 계산하고 무선으로 서로 통신할 수 있는 모듈을 갖춘 조그만 센서들의 네트워크로부터 획득한 데이터를 관리하는 시스템이다. 센서 네트워크 데이터베이스에서 센서 네트워크를 구성하고 있는 센서들은 단순히 정보를 수집할 뿐만 아니라 센서에서 생성된 정보를 필터링하고 공유하며 여러 센서 값을 융합하기도 한다. 하지만 센서들은 무선 네트워크로 서로 지속적인 통신을 하기에는 제한된 전력을 가지고 있으므로 시스템을 설계할 때 에너지 소비를 고려해야 한다. 또한 제한된 메모리와 대역폭을 지녔으므로 제한된 자원을 더욱 효율적으로 사용해야 할 필요가 있다.

센서 네트워크 데이터베이스에서 사용자는 자신들이 원하는 정보를 요구하는 질의를 미리 등록하여 원하는 결과를 얻게 된다. 이때 등록된 질의는 연속 질의(continuous query)로서 지속적으로 들어오는 스트림 데이터에 적용되어진다. 센서 네트워크 데이터베이스에 여러 개의 질의가 등록되면 센서 네트워크 데이터베이스에서는 많은 양의 스트림 데이터를 이들 등록된 여러 개의 질의들에 대하여 반복적으로 수행해야 하며 다중 질의 처리 방법에 따라서 에너지 및 메모리의 효율성이 달라질 수 있다. 따라서 센서 네트워크 스트림 데이터에 대해서 다중 질의를 효율적으로 처리하는 방법에 대해서 연구할 필요가 있다.

본 연구에서는 감시 목적의 미리 등록되어진 다중 질의를 효율적으로 처리하는 방법에 주안점을 두고 있다. 다중 질의들에 색인을 두어 센서 네트워크에서 오는 스트림 데이터와 관련되는 질의를 보다 빠르게 찾을 수 있게 하고자 한다. 즉, 기존의 데이터베이스에서는 질의가 들어오면 조건에 맞는 데이터를 찾기 위하여 데이터에 대하여 미리 색인을 만들어 두었으나 스트림 데이터를 다루는 센서 네트워크 데이터베이스에서는 이와는 반대로 계속적으로 들어오는 데이터에 대하여 관련되는 질의를 빠르게 찾기 위하여 오히려 질의에 색인을 만들게 된다. 질의에 색인을 만들지 않은 경우에 스트림 데이터가 하나씩 들어올 때마다 모든 질의들이 관련이 있는지를 다 검토해보아야 하므로 상당한 시간적인 낭비 및 비효율성이 발생한다. 본 연구에서 제안하는 색인 기법은 두 단계의 과정으로 해당하는 질의를 찾을 수 있게 되어있어 매우 빠른 검색이 가능하게 한다. 이를 위해서 공간 색인 기법들 중 속성별로 순차적으로 공간을 잘라서 색인하는 k-d 트리와 공간을 계층적 구조로 자르

※ 교신저자(Corresponding Author) : 이민수, 주소 : 서울시 서대문구 대현동 11-1 이화여자대학교 컴퓨터학과 (120-750). 전화 : 02)3277-3401, FAX : 02)3277-2306, E-mail : mlee@ewha.ac.kr
접수일 : 2007년 6월 13일, 완료일 : 2007년 10월 2일
† 종신회원, 이화여자대학교 컴퓨터학과
†† 준회원, LIG 넥스원 연구원
(E-mail : inverno@ewhain.net)
††† 준회원, 이화여자대학교 컴퓨터학과
(E-mail : auroree@ewhain.net)
※ 본 연구는 교육인적자원부의 2단계 BK21사업의 지원 및 이화여자대학교 디지털 컨텐츠 특성화 사업의 지원을 받아 수행하였음.

고 겹침 관계를 없애는 색인 기법인 R+-트리의 개념을 적용하여 다중 질의를 색인하는 새로운 트리인 SMILE (Stream data Multi-query Indexing with Level dimension nodes & Extended range nodes) 트리를 제안하고 있다. 질의들에 대한 SMILE 트리를 구성하여 센서 네트워크에서 생성되는 스트림 데이터에 대하여 관련된 질의를 탐색하도록 하면 질의를 순차 탐색하는 것과 비교하여 경우에 따라서는 평균 탐색시간을 약 50% 정도로 줄일 수 있다.

본 논문의 구성은 다음과 같다. 2장에서는 관련연구로 데이터 스트림 관리 시스템과 스트림 데이터를 처리하는 시스템 중 하나인 STREAM(Stanford Stream Data Manager) 및 CEI 기반의 질의 인덱싱 기법을 소개하고 데이터 스트림 관리 시스템의 대표적 시스템인 센서 네트워크 데이터베이스로 TinyDB를 설명하며 공간 색인 기법인 k-d 트리, R+-트리에 대해 설명한다. 3장에서는 다중 질의 처리방법과 다중 질의 색인 트리인 SMILE 트리를 제안하고 있다. 4장에서는 센서 네트워크 데이터베이스인 TinyDB를 기반으로 한 시스템 구현의 개발환경과 구현결과를 설명하고 성능을 평가한다. 마지막으로 5장에서는 결론 및 향후 연구 방향을 기술하고 있다.

2. 관련 연구

2.1 데이터 스트림 관리 시스템(Data Stream Management System)

데이터 스트림 관리 시스템[1]은 하나 또는 그 이상의 지속적인 스트림 데이터 형태로 들어오는 데이터들에 대해 질의를 수행하고 관리하는 시스템이다. 데이터 스트림 관리 시스템에서는 데이터에 대한 질의 결과의 정확성에 집중하기 보다는 질의 결과의 정확성은 약간 떨어지나 정답에 근접하며 상황에 따라 적합한 질의 결과를 제공하는 것에 주안점을 둔다. 데이터 스트림 관리 시스템에서 데이터는 실시간으로 도착하게 되며 시스템은 데이터가 도착하는 순서를 조정할 능력이 없고 무한으로 들어오는 스트림 데이터에서 이미 처리된 데이터 스트림 요소들을 보관하거나 버린다.

데이터 스트림 관리 시스템의 구조는 그림 1과 같다.

데이터 스트림 관리 시스템에서 사용되어지는 질

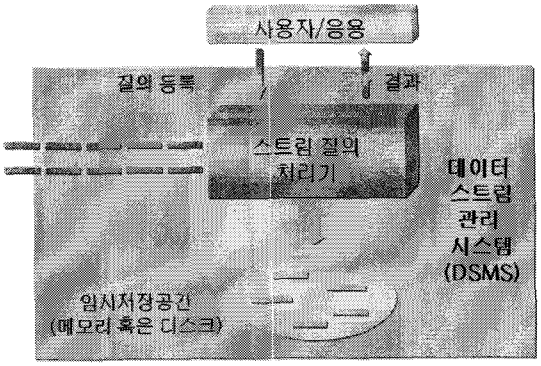


그림 1. 데이터 스트림 관리 시스템 구조

의는 크게 두 가지 방법으로 분류된다. 첫째는 일회성 질의와 연속 질의로 분류하는 방법으로서 일회성 질의는 데이터 집합의 특정 시간의 상태인 스냅샷(snapshot)을 보고 싶을 때 사용된다. 연속 질의는 지속적으로 감시 및 평가하고자 할 때 사용되며 같은 질의를 데이터 스트림에 계속 수행하는 방법이다. 두 번째 분류 방법은 미리 정의된 질의와 애드혹(ad hoc) 질의로 분류하는 것으로서 미리 정의된 질의는 스트림 데이터가 들어오기 전에 미리 사용자가 질의를 정의하여 등록하는 것으로 일반적으로 연속질의이다. 애드혹 질의는 데이터 스트림 관리 시스템이 실행 중에 갑자기 질의가 들어오는 것으로 주로 과거나 미래와 관련된 값을 물어보며 일회성 질의나 연속 질의일 수 있다. 하지만 데이터 스트림 관리 시스템 설계에 적용하기에는 복잡한 면이 있다.

데이터 스트림 관리 시스템에서 사용자 또는 응용은 질의를 등록하게 되고 등록된 질의들에 대해서 지속적으로 들어오는 스트림 데이터를 입력으로 하여 해당 데이터와 관련된 질의를 찾아냄으로써 질의를 수행하게 된다. 질의를 만족시키는 데이터가 들어오면 해당 질의를 등록한 사용자 또는 응용에게 해당 센서 데이터가 결과 값으로 보내어진다. 해당되는 질의를 찾을 때에는 메모리 또는 디스크의 특별한 자료구조를 이용하게 된다.

2.2 스트림 데이터 필터링과 질의 인덱싱 기법

STREAM[2]은 Stanford Stream Data Manager의 약자로 스트림 데이터를 처리하는 시스템이다. STREAM 시스템에서는 분산된 환경의 여러 데이터 소스들로부터 들어오는 스트림 데이터를 미리 등록

된 스트림 프로세서에서 다중 질의를 처리하게 된다. 이러한 환경에서는 급격하게 갱신된 스트림들로 인해 통신 오버헤드가 현저히 많을 수밖에 없다. 이에 STREAM 시스템에서는 통신 오버헤드를 줄이기 위해서 여러 원격 데이터 소스들에 필터를 설치하여 스트림 데이터의 양을 줄이는 방법으로 스트림 데이터를 처리하게 된다. 질의들에 대한 결과 값은 스트림 데이터를 처리하는 과정에서 집계 연산을 사용하므로 정확도가 떨어지게 된다. 이에 사용자는 질의 결과 값이 정확도면에서 약간 떨어진다는 것을 알고 있으며 상황에 따라서는 정확도가 더 떨어져도 크게 영향을 미치지 않는 범위가 있음을 고려한다. STREAM 시스템에서 사용자는 정확도를 미리 지정할 수 있으며 정확도가 낮을수록 원격 소스에서 필터링 하는 데이터의 양이 많아지고 스트림 프로세서에 전달되는 스트림 데이터의 양이 감소하여 통신 오버헤드를 줄이게 된다.

스트림 데이터에 대한 질의들은 연속 질의로서 들어오는 데이터에 대하여 계속적으로 적용되어 스트림 데이터가 해당 질의의 조건에 맞는지를 확인하게 된다. 이때 스트림 데이터 관리 시스템에서는 일반적인 데이터베이스에서의 질의와 데이터의 역할이 바뀌게 되어 인덱스를 구성할 때 데이터에 인덱스를 만드는 것이 아니고 질의에 인덱스를 구성하게 된다. 즉 데이터가 들어오면 즉시 해당되는 질의를 찾을 수 있도록 질의 인덱스를 사용하게 된다. 이러한 질의 인덱스 중에서 CEI(Containment-Encoded Interval) 기반의 질의 인덱싱 기법[3]이 있다. 이 기법은 간접적인 인덱싱 기법으로서 몇 개의 미리 정의된 가상적인 CEI를 중심으로 이루어진다. CEI들은 먼저 질의 구간들을 분해할 때뿐만 아니라 탐색을 하는 데에도 사용된다. CEI들은 서로간의 포함관계가 각자의 아이디 속에 내포가 되도록 정의되고 이름이 지어진다. 이러한 내포 관계가 아이디 속에 포함됨으로써 질의 분해와 탐색이 매우 용이해진다. 그리하여 효율적으로 원하는 스트림 데이터와 관련된 질의를 찾을 수 있다.

2.3 센서 네트워크 데이터베이스

센서 네트워크는 온도나 습도 등의 센싱 기능을 갖는 센서 모듈과 무선 송수신 모듈, 센서들의 정보를 모아 호스트 컴퓨터로 전송하는 게이트웨이, 그리고 센서의 데이터 값을 저장하고 모니터링하는 호스트 서버로 이루어져 있다.

센서 네트워크의 데이터를 관리하고 질의하기 위한 대표적인 시스템으로 버클리 대학에서 만든 TinyDB[4-6]가 있다. TinyDB는 센서 네트워크용 데이터베이스로 크게 TinyOS를 운영체제로 갖는 센서들에 탑재되는 부분과 질의를 던지고 센서 네트워크로부터 받은 결과 값을 처리하여 사용자에게 보여주는 PC에 탑재되는 부분으로 구성되어 있다. 사용자가 질의를 보내면 게이트웨이를 통해 센서 네트워크의 루트 노드에 전달되며 루트 노드는 질의를 자식 노드에게 전달한다. 각 노드들은 감지한 값을 처리하면서 질의를 만족시키는 값을 부모 노드에게 전달하며 최종적으로 사용자에게 도착된 결과 값들은 실시간으로 그래프에 표시되어 보여주게 된다. 그림 2에 TinyDB의 구조와 TinySQL로 작성된 질의를 보인다.

2.4 공간 색인 기법

공간 색인 기법[7]은 위치를 기준으로 하여 공간을 어떻게 분할하여 색인을 두느냐에 따라 크게 두 가지로 나뉠 수 있다. 공간을 구성하는 축(좌표축 혹은 차원)을 기준으로 공간을 잘라서 각 공간에 색인을 두는 방법과 임의의 영역들로 데이터를 잘라서 영역 데이터들에 대해 색인을 두는 방법이 있다. 전자의 대표적인 공간 색인 기법으로는 grid file, k-d 트리가 있다. 후자의 경우는 R-트리와 R-트리 변형 트리들이 있다.

k-d 트리[8]는 그림 3에 보인 것처럼 다차원 공간에 정의된 점들을 이용하여 공간을 분할하는 이진 탐색 트리를 기반으로 하는 데이터 구조이다. k-d

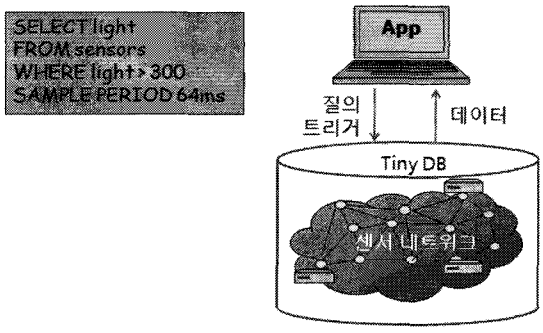


그림 2. TinyDB의 구조

트리의 구조는 이진 트리 형태이며 트리의 각 노드들은 자식 노드 또는 null 값을 가리키는 두 개의 포인터 값을 가지고 있으며 단말 노드에 데이터를 저장하고 있다. 루트 노드부터 단말 노드까지 노드들은 식별자를 가지고 있으며 트리에서 같은 레벨에 있는 노드들은 같은 식별자를 지니게 된다. 예를 들어 루트 노드가 식별자 0이라면 두 자식 노드들은 식별자 1을, k번째 레벨의 식별자는 k-1을 가지게 되며 k+1번째 레벨의 식별자는 0이 되어 이러한 사이클은 반복되게 된다. 각 노드에서 식별자 값보다 작은 값을 지닌 노드는 왼쪽 자식 노드에 위치하게 되며 식별자 값보다 큰 경우는 오른쪽 자식 노드에 위치하게 된다. 이차원에 데이터가 저장된 2-d 트리의 예는 그림 3과 같다.

R-트리[9]는 데이터-분할 방법의 대표적인 공간 색인 기법으로서 공간 객체를 최소 경계 사각형(Minimum Bounding Rectangle)을 사용하여 복잡한 공간 객체를 단순하게 저장하는 B-트리와 유사한 높이 균형 트리이며 R+-트리[10]는 R-트리의 최소 경계 사각형의 중복이 허용됨으로써 각 레벨에서 많은 엔트리들이 검색 윈도우와 겹쳐져서 색인 성능이 떨어지는 점을 보완한 트리이다. 절단-기반(clipping-based) 방법을 사용하여 노드간의 중복을 허용하지 않는 색인 구조이며 최소 경계 사각형의 중복을 없앰으로써 검색의 성능을 높였다. 그림 4에 R+-트리의 예를 보인다.

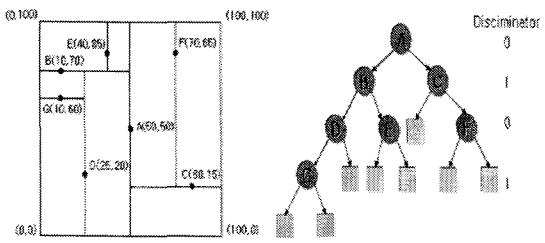


그림 3. k-d 트리 구조

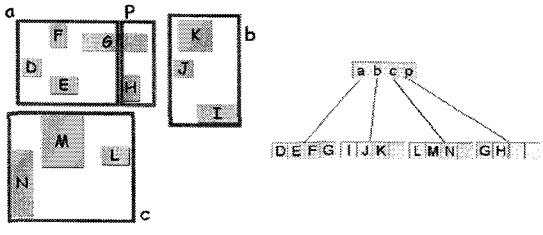


그림 4. R+-트리 예

3. 센서 네트워크에서의 스트림 데이터 처리를 위한 다중 질의 색인 기법

본 장에서는 기존 센서 네트워크에서 다중 질의에 관해 스트림 데이터를 처리하는 방법에 대하여 설명하고 관련 연구를 바탕으로 하여 센서 네트워크에서의 스트림 데이터 처리를 위한 미리 정의된 다중 질의를 색인하는 SMILE(Stream data Multi-query Indexing with Level dimension nodes & Extended range nodes) 트리를 제안하고 있다.

3.1 기존 센서 네트워크의 스트림 데이터 처리

센서 네트워크에서 사용자는 질의를 보내고 센서가 작동되는 동안 지속적으로 그 결과를 얻음으로써 원하는 정보를 얻게 된다. 사용자가 질의를 보내면 질의는 센서들에 전달되고 센서들은 빛, 온도, 습도 등을 감지하고 계산하여 질의를 만족하는 결과 값을 사용자에게 지속적으로 보내게 된다.

센서 네트워크에서 한 사용자가 질의를 여러 개 보내거나 여러 사용자로부터 다양한 질의가 센서 네트워크에 보내어질 때 이를 처리하는 간단한 방법으로는 감지한 센서 데이터를 각각의 질의 하나 하나에 관련이 되는지를 순차적으로 비교하는 것이다. 그러나 이 방법의 경우 모든 질의들에 대해 모든 센서 데이터를 하나하나 다 비교해야 하므로 센서 데이터 하나당 질의의 수만큼 비교를 해야 한다. 이를 보다 효율적으로 할 수 있는 방법으로 미리 정의된 다중 질의들에 대해 트리 형태의 색인을 두어 데이터로 하여금 해당 질의를 보다 빠르게 검색할 수 있도록 하고자 한다. 그림 6에서 순차탐색을 사용한 방법과 질의 색인을 두어 데이터 스트림을 처리하는 방법을 비교하여 보여준다. 기존의 데이터베이

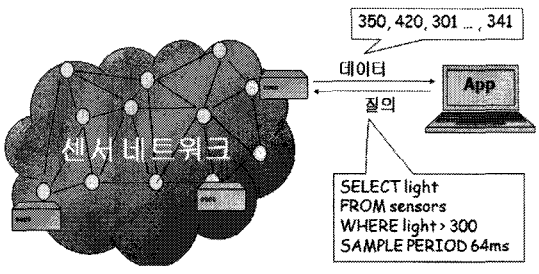


그림 5. 센서 네트워크에서의 질의와 질의 결과 전달

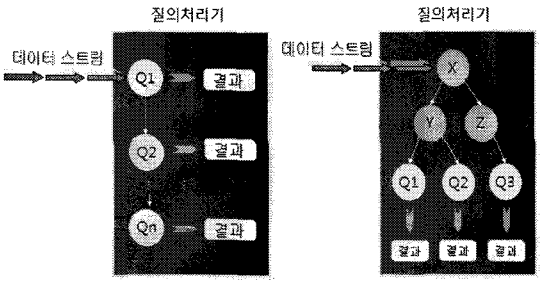


그림 6. 순차 탐색 방법과 질의에 트리 형태의 색인을 두어 탐색하는 방법

스에서는 데이터에 대하여 색인을 만들고 질의가 데이터를 검색하는 수행속도를 향상시켰으나 스트림 데이터 환경에서는 반대로 질의에 색인을 만들고 데이터가 관련된 질의를 효율적으로 검색토록 한다는 차이점이 있다.

3.2 SMILE 트리의 개념과 구조

SMILE (Stream data Multi-query Indexing with Level dimension nodes & Extended range nodes) 트리는 속성(차원)별로 순차적으로 공간을 잘라서 색인하는 k-d 트리와 공간을 계층적 구조로 자르고 겹침 관계를 없애는 색인 기법인 R+-트리의 개념을 적용하여 만든 트리이다. 이러한 SMILE 트리는 미리 정의된 연속 질의만 색인하는 기법으로 일회성 질의나 애드혹(Ad hoc) 질의의 처리는 고려하고 있지 않다.

SMILE 트리는 두 부분으로 이루어져 있으며 첫 번째 부분은 k-d 트리와 유사한 부분으로 미리 정의된 질의에서 센서의 속성의 종류와 속성의 값을 기준으로 순차적으로 공간을 분할한 트리 부분으로 Level dimension 트리라 불리고, 두 번째 부분은 R+-트리와 유사한 부분으로 질의 조건들에 따라 질의의 포함 여부를 트리 형태로 나타낸 부분으로 Extended range 트리로 불린다. 앞서 순차적으로 분할한 Level dimension 트리에 후자의 Extended range 트리가 연결되어 있는 형태이다.

센서 네트워크의 스트림 데이터는 그 양이 많으므로 하나하나 다 모든 질의에 데이터를 비교하여 데이터가 해당하는 질의를 찾기에는 탐색 시간이 오래 걸린다. SMILE 트리의 Level dimension 트리는 이러한 점을 해결하고자 SMILE 트리의 상위 부분에 위치하여 차원을 나누어 주며 탐색 공간을 줄이는 역할을 한다. Level dimension 트리를 통해서 데이터들은 각각의 분할된 영역 안에 속성이 유사한 값들끼리 모이게 되며 이 안에서 더 구체적으로 해당되는 질의를 찾을 때 사용되는 트리가 SMILE 트리의 하위 부분에 해당하는 Extended range 트리이다. 그림 7에서 Level dimension 트리 노드와 Extended range 트리 노드를 포함한 SMILE 트리의 구조를 보여준다.

본 연구에서는 기본적인 AND 연산으로 이루어진 질의들을 인덱스하는 기법을 제공하고 이를 질의 분해 개념과 연관지어 확장하면 OR 기능 등을 갖는 질의어로 쉽게 확장될 수 있다. 즉 X OR Y라는 질의가 있으면 X라는 질의와 Y라는 질의로 분해하여 SMILE 트리에 두 개의 하위 질의로 QX 및 QY를 등록시킬 수 있다. 그리고 SMILE 트리에서 검색결과로 질의 리스트가 나오면 이 리스트에서 QX OR QY를 적용시켜서 참이면 원래의 질의가 만족되는 것으로 볼 수 있다. 이런 방식으로 인덱스 결과 이후에 간단한 "후처리" 과정을 두게 되면 다양한 질의 언어에 대한 지원으로 확장 가능하다.

3.2.1 Level dimension 트리의 구조

SMILE 트리의 상위에 위치한 Level dimension 트리의 노드 구조는 그림 8과 같다.

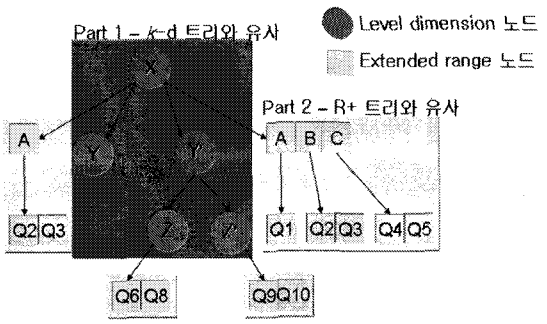


그림 7. SMILE 트리 구조

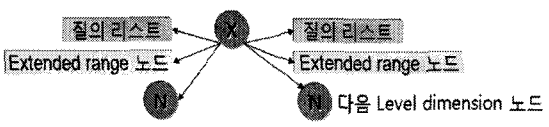


그림 8. Level dimension 트리의 노드 구조

Level dimension 트리에서는 질의에 포함된 데이터의 각 속성별로 각 레벨의 노드가 생성된다. 센서 네트워크에서 각각의 센서의 속성(예로 온도, 빛, 파워 등)별로 하나의 레벨의 노드들이 만들어진다. Level dimension 트리의 노드 구조는 질의 리스트(QueryList), Extended range 트리 노드와 다음 속성의 Level dimension 노드 이렇게 3개의 포인터를 양쪽으로 갖는 구조로 되어 있다. 질의 리스트는 현재 노드의 값에 대하여 조건이 만족되는 질의들만을 저장하는 리스트이다. 예를 들어서 현재 노드의 속성이 빛이고 노드의 값이 30일 때 질의 중 '빛 < 30' 이 있다면 이 질의는 질의리스트에 저장되며 그중에서도 왼쪽 질의리스트에 저장된다. Extended range 트리 노드는 R+-트리와 유사한 Extended range 트리를 가리키는 포인터이다. Extended range 트리는 질의의 조건의 포함관계에 따라 질의의 계층적인 관계를 나타낸 트리이다. Level dimension 노드에서 Extended range 트리로 연결되며 질의에 조건이 많을수록 Extended range 트리의 크기가 커진다. 그림 8에서 N으로 표시된 다음 Level dimension 노드는 센서의 속성 중 현재 Level dimension 노드에서 사용된 속성 다음에 고려해야 할 속성과 값을 저장하고 있는 하위의 Level dimension 노드를 가리킨다.

Level dimension 노드들은 질의에 나타난 센서의 속성 수와 같은 계층 수를 가진 트리를 이루게 되며 각각의 노드는 필요에 따라 노드의 값을 기준으로 왼쪽과 오른쪽에 Extended range 트리를 하나씩 가지고 있을 수 있다. 예를 들어 미리 정해진 세 개의 단순한 질의들(Q1, Q2, Q3)이 그림 9와 같다면 속성은 빛(light), 온도(temp), 파워(power) 총 3개로서 속성 수와 같은 깊이의 트리를 형성하게 되며 속성별로 노드의 값이 정해진다. 그림 9의 예에서 세 개의 질의는 설명의 편의상 하나씩의 속성에 대해서만 정의되었지만 실제로는 세 가지의 센서의 속성 모두에 대해서도 각각의 질의 조건을 정의할 수 있다.

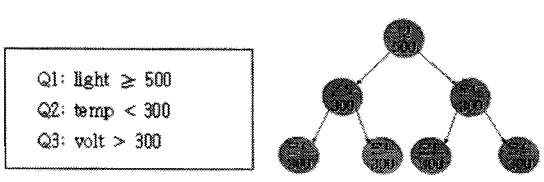


그림 9. 질의와 Level dimension 트리 형성의 예

3.2.2 Extended range 트리의 구조

Extended range 트리는 Level dimension 트리의 각 노드에 연결되는 트리로서 구조는 그림 10와 같다.

Level dimension 노드의 값을 기준으로 질의 조건 범위 값이 Level dimension 노드 값보다 작으면 왼쪽 Extended range 트리에 질의가 속하게 되고, 크거나 같으면 오른쪽 Extended range 트리에 질의가 속하게 된다. 이 때 질의 조건이 Level dimension 노드의 값과 일치하면 Extended range 트리를 생성하지 않고 Level dimension 노드의 해당하는 왼쪽 혹은 오른쪽 질의 리스트에 질의 번호를 삽입하게 된다. Extended range 트리의 각 노드는 {질의 식별자, 속성들 범위, 자식 포인터}의 쌍들로 구성된 하나 이상의 항목들을 갖는다. Extended range 트리의 노드에는 속성이 서로 겹치지 않는 범위의 항목들이 형제로 포함되며 속성 범위가 포함관계가 형성되면 자식 노드로 연결된다. 이에 대한 상세한 내용은 Extended range 트리 생성 방법에서 예를 보이며 설명한다.

3.3 SMILE 트리의 생성 방법

SMILE 트리는 크게 두 부분으로 이루어져 있으며 첫 번째 부분인 Level dimension 트리 부분은 스트림 데이터를 속성별(차원별)로 단계적으로 필터링하는 역할을 하며 두 번째 부분인 Extended range 트리 부분은 Level dimension 트리에 붙는 확장 트리로 질의 영역의 포함 관계를 트리로 구성하여 빠른 질의 검색 역할을 한다. SMILE 트리가 k-d 트리와 R+-트리를 합친 유사한 트리이기에 k-d 트리의 필터링 특징과 R+-트리의 빠른 검색 속도를 특징으로 갖는다.

SMILE 트리는 센서 네트워크 데이터베이스에 미리 등록되어진 연속 질의들을 질의 수행 전에 분석한 후 이 정보를 이용하여 Level dimension 트리를 생성하고 그 후에 Extended range 트리를 생성하는 과정을 거쳐서 SMILE 트리의 생성이 완료된다.

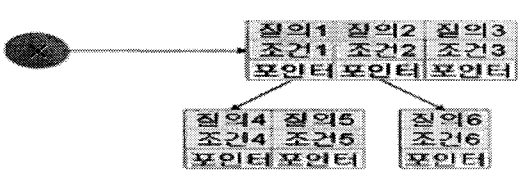


그림 10. Extended range 트리의 구조

SMILE 트리는 감시 목적의 센서 네트워크에서 미리 정의된 다중 질의를 주요 대상으로 하고 있다. 감시 목적에서는 주된 질의들이 미리 정의되는 경우가 많으므로 이러한 환경을 대상으로 하였고 SMILE 트리의 생성과정에서 이러한 정보를 활용하여 최적의 구조를 생성하게 된다. 그러나 경우에 따라서는 매우 발생 빈도는 적으나 새로운 질의를 추가해야 하는 경우도 발생할 수 있다. 이러한 경우에는 삽입 알고리즘이 따로 제공되지 않고 현재의 SMILE 트리를 재생성하는 작업을 진행하게 된다. SMILE 트리에 대한 삽입 알고리즘을 제공하지 않고 재생성해야 하는 이유는 SMILE 트리를 생성할 때에는 전체 질의들을 대상으로 하여 최적의 구조를 결정하기 위하여 SMILE 트리의 구축 과정에서 전체 질의를 보고 판단하는 부분들이 포함되어 있기 때문이다. 실험 결과 재생성 시간이 매우 적기 때문에 검색시간에 대해 미치는 영향은 적으며 재생성을 하게 됨으로써 항상 최적의 구조를 유지할 수 있다.

3.3.1 Level dimension 트리의 생성 방법

Level dimension 트리를 생성하기 위해서는 일단 질의 정보를 알아야 한다. 질의를 분석하여 총 몇 개의 센서 속성이 있는지 확인하고 속성 수와 동일한 높이의 Level dimension 트리를 생성하게 된다. 다중 질의 리스트에서 각각의 속성이 사용되는 회수를 계산하여 가장 많이 나오는 속성 순으로 Level dimension 트리의 레벨들을 형성하게 된다. 즉 가장 많이 나오는 속성이 맨 처음 트리의 루트 노드가 되며 그 다음에 많이 나오는 속성이 다음 레벨이 되고 이러한 방법으로 모든 속성을 차례대로 트리에 레벨별 기준 속성이 되게 한다. 이 때 각각의 노드에서 속성의 값을 설정할 때에는 사용자 질의들에서 속성에 대한 조건으로 사용된 모든 값들 중 중간 값이 선택되게 된다.

예를 들어 그림 11과 같은 사용자 질의들이 있다면 속성은 빛(light), 온도(temp), 볼트(volt) 총 3개이며, 모든 질의들에서 빛은 4회, 볼트는 3회 그리고 온도는 1회 사용되었다. 따라서 생성되는 Level dimension 트리의 루트에서 사용되는 속성은 빛이고 다음 레벨은 볼트, 온도 순이 된다. 각 노드의 기준 값은 질의 값의 중간값이므로 빛은 20, 25, 30, 40 중에서 중간인 30이 된다. 볼트의 기준 값의 경우 빛의 하위 노드인 왼쪽과 오른쪽에 각각 사용될 값들을 정해야 한다. 각각의 질의들에 대하여 부모 노드의 속성 값과 비교하여 질의의 범위의 값이 속성 값보다 작으면 왼쪽 하위 노드에, 크거나 같으면 오른쪽 하위 노드에 질의들이 위치하게 된다. 이때 각각의 하위 노드들에 위치한 질의들의 속성 값에서의 중간값을 취해야 한다. 예를 들어 질의 Q2는 빛이 30이상이므로 부모 노드의 조건과 비교하여 질의의 범위가 크거나 같으므로 오른쪽 하위 노드에 위치하게 된다. 이러한 방식으로 질의들을 위치시킨 후 각각의 하위 노드들 위치에서의 중간값을 취하게 되므로 루트 왼쪽 하위 노드의 볼트는 10을 오른쪽 하위 노드는 15를 갖게 된다. 마지막으로 온도는 빛과 볼트의 값을 비교한 결과 맨 오른쪽 하위 노드에 26값을 가지게 되며 최종적으로 생성된 Level dimension 트리는 그림 11에서 보여주고 있다. 그림 12는 Level dimension 트리에 노드를 삽입하는 알고리즘을 보여준다.

3.3.2 Extended range 트리 생성 방법

Extended range 트리는 Level dimension 트리의 노드에 연결되어 있으며 Extended range 트리의 각 노드에는 질의의 식별자와 질의의 조건 정보를 가지고 있다. Level dimension 트리의 노드에 Extended range 트리와 질의 리스트를 삽입하는 알고리즘을 그림 13에 기술하였다.

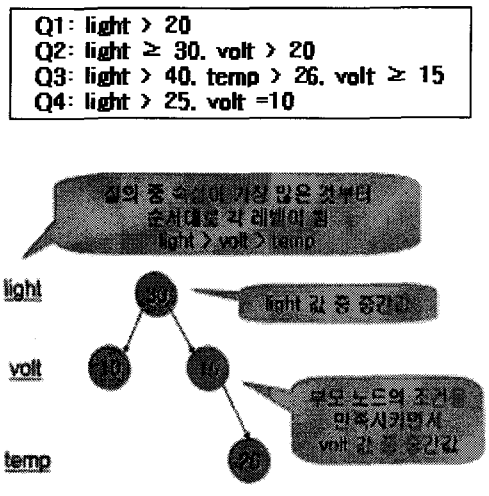


그림 11. 질의와 생성된 Level dimension

LevelDimensionTree 알고리즘

```
CreateLevelDimensionTree()
input : value, query, current_node, value_size, current_pos, attr_num
/* value: 속성별 후보값이 저장된 벡터, query: 질의, current_node : 현재 노드, value_size : 벡터 사이즈,
curret_pos : 현재 확인 중인 벡터 후보값 수 attr_num : 질의에 나타난 센서의 속성 수*/

vector left_values;  // 왼쪽 자식 노드 후보 값 임시 저장
vector right_values;  // 오른쪽 자식 노드 후보 값 임시 저장
Node temp // 임시 노드
/* 각 Level 의 속성을 정하고 각 Level 후보 값을 저장 */
LevelOrder();
/* Level 트리 생성 */
if tree_size = 0 then  // 현재 level dimension 트리의 높이
   temp.value ← temp.getMedian(value);      //중간값을 노드 값으로 저장
   root ← temp;
   tree_size ← tree_size + 1;
end
else                                        // 노드를 삽입할 위치 찾음
   if compare_value(query, current_node.value) <= 0 then  //현재 노드와 질의 비교
      if current_node.left = NULL then //현재 노드의 왼쪽 Level dimension 자식노드가 없는 경우
         left_values.add(value);     //현재 노드의 왼쪽 Level dimension 자식노드 후보 저장
         end
         if current_pos = value_size then  //벡터 안의 모든 후보 값 확인여부
            current_node.set_left(left_values); //leftvalues 값 중 중간 값을 왼쪽 노드로 삽입
         end
      else
         left_values.add(value);     //왼쪽 자식 노드 후보 저장
         if current_pos = value_size then //벡터 안의 모든 후보 값 확인여부
            current_node ← current_node.left;     //한 레벨 내려감
            find_location(current_node, query, value, tree_size, attr_num);// 노드를 삽입할 위치 찾음
         end
      end
   end
   if compare_value(query, current_node.value) >= 0 then //현재 노드와 질의 비교
      if current_node.right = NULL then      //오른쪽 Level dimension 자식노드가 없는 경우
         right_values.add(value);  //오른쪽 Level dimension 자식노드 후보 저장
      end
         if current_pos = value_size then //벡터 안의 모든 후보 값 확인여부
            current_node.set_right(right_values);        //rightvalue 값 중 중간값을 오른쪽 노드로 삽입
         end
      else
         right_values.add(value);  //오른쪽 Level dimension 자식노드 후보 저장
         if current_pos = value_size then //벡터 안의 모든 후보 값 확인여부
            current_node ← current_node.right;  //한 레벨 내려감
            find_location(current_node, query, value, tree_size, attr_num);// 노드를 삽입할 위치 찾음
         end
      end
   end
end
return LevelDimensionTree;
```

그림 12. Level dimension 트리에 노드 삽입 알고리즘

InsertExtendedRangeTree 알고리즘

```
InsertExtendedRangeTree()
input : attr, query, root, query_id
//attr : 센서 데이터 속성, query : 질의, root : Level Tree의 루트, query_id : 질의 번호

current_node ← root;
while search = FALSE
   if query.contains(attr) then  // 질의가 현재 노드의 속성을 가지고 있을 때
      int type ← compare(query, attr, current_node.value); // 질의 삽입 위치 결정
      switch(type)
         case LEFT_EXTENDEDTREE_RIGHT_QUERYLIST :
            if leaf then   //왼쪽 Extended 트리 노드와 오른쪽 질의 리스트에 질의 삽입
               left_Extended(attr, query_id); //왼쪽 Extended 트리 노드에 질의 번호 삽입
               current_node.add_right_QueryList(query_id); //오른쪽 질의 리스트에 질의 번호 삽입
               search ← TRUE;
            end
            else
               current_node ← current_node.left;
               current_node ← current_node.right;
            end
            break;
         case RIGHT_EXTENDEDTREE:
            if leaf then   //오른쪽 Extended 트리 노드에 질의 삽입
               right_Extended(attr, query_id);      //오른쪽 Extended 트리 노드에 질의 query_id 삽입
            end
            else
               current_node ← current_node.right;
            end
            break;
         case LEFT_QUERYLIST_RIGHT_EXTENDEDTREE :
            if leaf then      //왼쪽 질의 리스트와 오른쪽 Extended 트리 노드에 질의 삽입
               current_node.add_left_QueryList(query_id); //왼쪽 질의 리스트에 질의 query_id 삽입
               search ← TRUE;
               right_Extended(attr, query_id);      //오른쪽 Extended 트리 노드에 질의 query_id 삽입
            end
            else
               current_node ← current_node.left;
               current_node ← current_node.right;
            end
            break;
         case LEFT_EXTENDEDTREE :
            if leaf then        //왼쪽 Extended 트리 노드에 질의 삽입
               left_Extended(attr, query_id);        //왼쪽 Extended 트리 노드에 질의 query_id 삽입
            end
            else
               current_node ← current_node.left;
            end
            break;
         case LEFT_QUERYLIST :
            if leaf then        //왼쪽 질의 리스트에 질의 삽입
               current_node.add_left_QueryList(query_id);  //왼쪽 질의 리스트에 질의 query_id 삽입
               search ← TRUE;
            end
```

```
            else
               current_node ← current_node.left;
            end
            break;
         case BOTH_EXTENDEDTREE:
            if leaf then      //양쪽 extended 트리에 질의 삽입
               left_Extended(attr, query_id);      //왼쪽 Extended 트리 노드에 질의 query_id 삽입
               right_Extended(attr, query_id);     //오른쪽 Extended 트리 노드에 질의 query_id 삽입
            end
            else
               current_node ← current_node.left;
               current_node ← current_node.right;
            end
            break;
         case RIGHT_QUERYLIST:
            if leaf then      //오른쪽 질의 리스트에 질의 삽입
               current_node.add_right_QueryList(query_id); //오른쪽 질의 리스트에 질의 query_id 삽입
               search ← TRUE;
            end
            else
               current_node ← current_node.right;
            end
            break;
      end
   end
   else              //질의가 가지고 있는 속성이 현재 노드 속성에 없을 경우
      current_node ← current_node.left;
      current_node ← current_node.right;
   end
end
return SMILETree
```

그림 13. 질의 리스트 및 Extended range 트리 노드 삽입 알고리즘

Level dimension 트리의 루트로부터 하향식(top-down)으로 Level dimension 트리의 노드 값과 질의를 비교하며 따라가다가 질의의 속성의 범위 값이 Level dimension 트리 노드의 속성 값보다 작으면 Level dimension 트리 노드의 왼쪽에 Extended range 트리가 생성되며, 크거나 같으면 오른쪽에 Extended range 트리가 생성된다. 해당 Level dimension 트리의 노드에 Extended range 트리가 없으면 Extended range 트리를 생성하고 Level dimension 트리에 연결하게 된다. 하지만 Extended range 트리가 있을 경우에는 Extended range 트리 노드 안에 있는 조건과 현재 삽입하려는 질의와 비교하여 질의 조건의 포함관계에 따라 크게 3가지 방법으로 삽입이 된다. 그림 14에 이들 방법들을 보여준다.

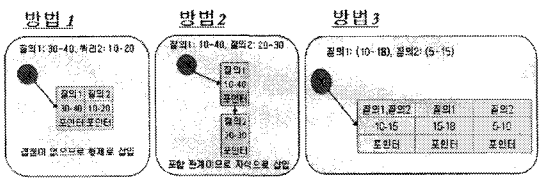


그림 14. Extended range 트리에서의 노드 삽입 방법

첫 번째 방법은 삽입하려는 질의 조건과 기존의 이미 삽입된 조건 사이에 아무런 포함관계가 없으면 형제 노드로 질의 조건과 질의 정보가 삽입된다. 두 번째 방법은 삽입하려는 질의 조건을 비교해 본 결과 기존의 노드에 있는 질의 조건에 포함이 되면 자식 노드로 질의와 질의 조건이 삽입된다. 마지막으로 둘의 조건이 부분적으로 포함관계이면 조건이 포함되는 부분과 포함되지 않는 부분으로 나누어져서 형제

노드로 삽입이 된다. 그림 15는 6개의 질의들에 대하여 SMILE 트리를 생성하는 과정을 보여준다.

3.4 SMILE 트리 탐색 방법

SMILE 트리의 탐색은 비교적 간단하며 탐색 성능이 빠르도록 트리가 구성되었다. 질의를 루트에서부터 비교하며 노드의 값보다 센서 데이터 값이 크거나 같으면 오른쪽 자식 노드를, 작으면 왼쪽 자식 노드를 따라 트리를 내려가며 Level dimension 트리 노드에서 비어있지 않은 질의 리스트를 만나면 해당 질의 식별자를 출력하고 Extended range 트리를 만나면 Extended range 트리를 따라가 센서 데이터 값이 노드에 포함된 질의들의 범위 조건을 만족시키는 질의 식별자만 출력하면서 트리를 탐색하게 된다. 그림 16에 SMILE 트리의 탐색 알고리즘을 기술하였다.

탐색 방법은, 예를 들어 그림 17과 같이 센서 데이터가 빛은 36, 볼트는 20 그리고 온도는 12라고 할 때 루트에서 Level dimension 노드와 값을 비교하여 센서 데이터가 빛이 36으로 루트보다 크므로 오른쪽 Level dimension 자식 노드로 탐색을 계속하게 된다. 이때 오른쪽 Level dimension 자식 노드로 가기 전에 오른쪽 질의 리스트가 있으면 질의 식별자들을 출력해주고 오른쪽 Extended range 트리가 있으면 따라가게 된다. 예에서 오른쪽 질의 리스트에 Q1이 있으므로 Q1을 출력한다. Extended range 트리에서 센서 데이터와 질의 조건을 비교하여 범위에 속하는 경우에만 질의 식별자를 출력해 주며 더 이상 형제 Extended range 트리 노드는 비교하지 않는다. 예에서 Q5를 출력한다. 자식노드로 Extended range 노드가 있는 경우에는 계속적으로 탐색을 진행하지만 더 이상 탐색해야 할 노드가 없으면 Extended range 트리 노드의 탐색은 완료되고 Level dimension 트리의 그 다음 자식 노드를 따라가게 된다. 다음 레벨은 속성이 볼트이므로 센서 데이터가 20이고 현재 노드 값인 15보다 크므로 오른쪽 자식 노드로 따라가게 된다. 또다시 Level dimension 노드에 연결된 질의 리스트와 Extended range 노드를 처리하게 된다. 예에서는 아무 질의도 해당되지 않아 아무것도 출력되지 않는다. 이러한 방법으로 마지막 Level dimension노드까지 탐색을 하면 탐색이 완료된다. 그리하여 결국 예에서는 Q1과 Q5의 관련된 질의들이 찾아진다.

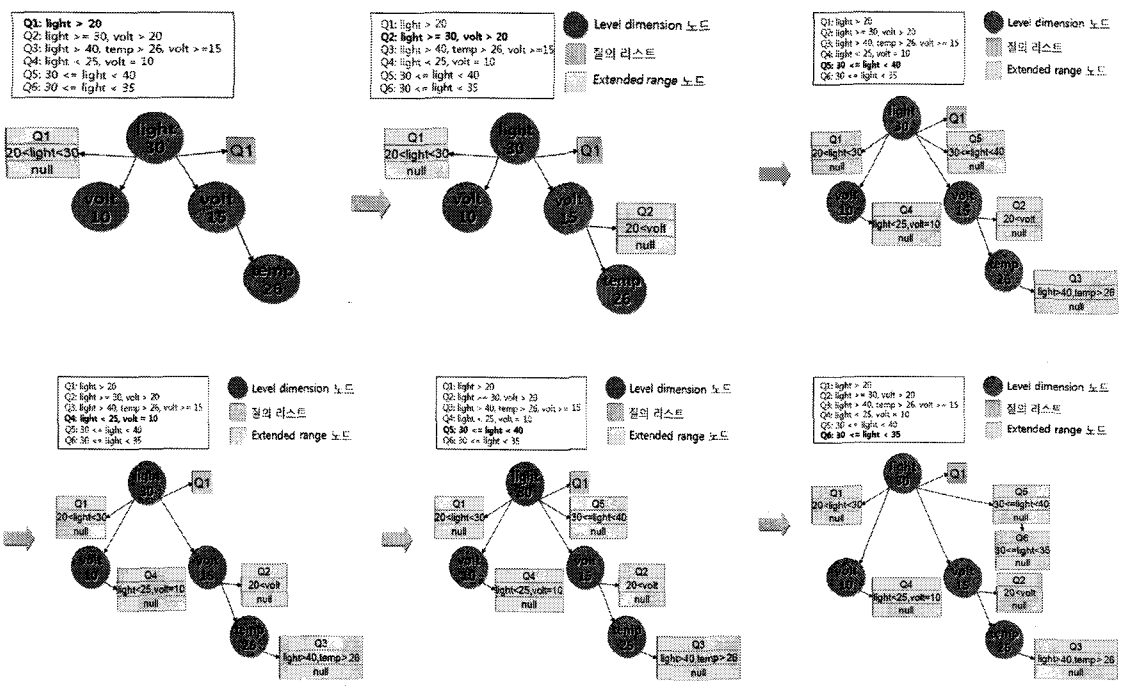


그림 15. Extended range 트리 생성 과정과 전체 SMILE 트리 생성 과정

SMILETreeSearch 알고리즘

```
SMILETreeSearch()
input : data, attr, current_node
// data : 센서 스트림 데이터, attr : 현재 Level 트리의 속성, current_node : SMILE 트리의 루트
//변수 선언
search ← FALSE;
while search = FALSE
   if compare_value(data, attr, current_node.value) < 0 then  //왼쪽 자식 노드 탐색
      if current_node.left_QueryList.size() > 0 then          // 왼쪽 질의 리스트 출력
         System.out.println("result query: + current_node.left_QueryList);
         if current_node.left_Extended != NULL then           //왼쪽 extended 트리 탐색
          Extended_search(data, current_node.left_Extended);
         end
      end
      if current_node.left = NULL then
         search ← TRUE;
      end
      else
         current_node ← currend_node.left;
      end
   end
   else if compare_value(data, attr, currend_node.value) > 0 then  //오른쪽 노드 탐색
      if current_node.right_QueryList.size() > 0 then          // 오른쪽 질의 리스트 출력
          System.out.println("result query: + current_node.right_QueryList);
      end
      if current_node.right_Extended != NULL then
          Extended_search(data, current_node.right_Extended); //오른쪽 extended 트리 탐색
      end
         if current_node.right = NULL then
            search ← TRUE;
         end
      end
      else
          current_node ← current_node.right;
      end
   end
end
```

그림 16. SMILE 트리 탐색 알고리즘

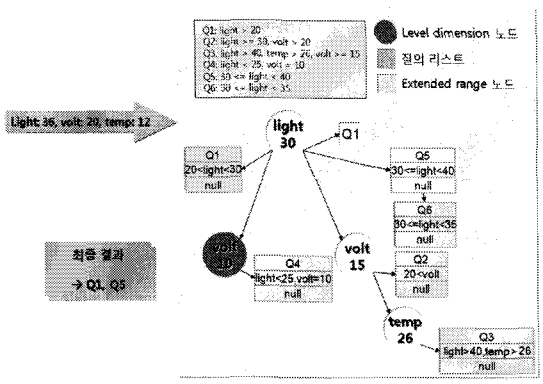


그림 17. SMILE 트리 탐색 과정 예

4. 구현 및 성능 평가

본 논문에서 제안한 SMILE 트리를 구현하고 버클리에서 개발한 센서 장치 및 TinyOS, TinyDB를 이용해 센서 네트워크의 스트림 데이터를 SMILE 트리에 입력으로 하여 실험을 수행하였다. 또한 SMILE 트리를 이용한 탐색과 순차 탐색 및 CEI 기법[3]의 성능을 비교해 보았다.

4.1 시스템 구현 환경

자바를 사용하여 이클립스 자바 통합 개발 환경에

서 SMILE 트리를 구현하였다. 센서 네트워크 시스템은 버클리 대학에서 개발한 TinyOS를 운영체제로 동작하는 휴인스 회사의 센서 키트인 U-Star 2400을 사용하였고 TinyOS의 최신버전은 tinyos-1.x로써 TinyDB및cygwin등을 사용할 수 있다. TinyDB는 Host PC 에서 GUI 혹은 명령창으로 센서 네트워크와 질의를 주고받을 수 있는 구조로 베이스 스테이션 센서로부터 값을 받아 처리하고 보여준다.

TinyDB에 SMILE 트리를 추가한 시스템의 구조는 그림 18과 같다. SMILE 트리는 미리 정의되고 등록된 질의를 입력으로 받아서 센서 데이터를 스트림 데이터로 받기 전에 생성되며 한 번 생성된 후에는 애드혹(Ad hoc) 질의를 반영하지 않으므로 계속 미리 정의된 질의를 연속 질의로 적용하게 된다. TinyDB로 들어오는 센서 네트워크의 스트림 데이터는 전처리로 생성되어진 SMILE 트리를 따라가며 탐색을 하고 센서 데이터가 질의 조건을 만족시키는 경우 해당 질의 식별자를 결과 값으로 출력해준다.

4.2 다중 질의에 대한 SMILE 트리 생성 및 스트림 데이터 탐색 결과

TinyDB안에서 미리 정의된 질의들에 대하여 생성된 SMILE 트리와, 스트림 데이터가 만족하는 다중 질의를 출력해 주는 결과 화면은 다음과 같다. 본 실험에서 사용된 센서는 총 8개이며 각각의 센서들은 빛, 온도, 볼트, 자신의 ID, 노이즈 값을 스트림 데이터로 SMILE 트리의 입력으로 사용되었다. 각각의 질의들은 모든 센서를 대상으로 하는 공통적인 질의들이다.

그림 19의 질의들은 그림 20과 같은 모양의 SMILE 트리로 생성이 되며, 스트림 데이터인 센서 데이터는 이 SMILE 트리를 통하여 센서 데이터를 만족시키는 질의를 탐색하게 되고 그 찾아진 질의들에 대한 결과는 그림 21과 같이 출력된다. 그림 21에서 첫 줄은 센서 데이터의 각 속성별로 값을 나타내며 nodeID는 센서의 ID로 해당 센서가 감지한 빛, 온도, 볼트, 노이즈 값을 나타낸다. 그리고 해당 센서가 감지한 모든 값을 만족시키는 질의 번호를 결과 값으로 밑에 출력해준다. 금회 탐색 소요 시간은 센서 데이터 하나를 탐색하는 과정에 걸린 시간으로 단위는ms이다. 본 실험에서 빛, 온도, 볼트, 센서 노드 ID, 노이즈 이렇게 다섯 개의 속성이 하나의 센서 데이터를 구성한다. 총 누적 소요시간은 지속적으로 센서 데이터를 입력받아 탐색하였을 때 걸린 총 누적 시간으로 단위는 ms이다.

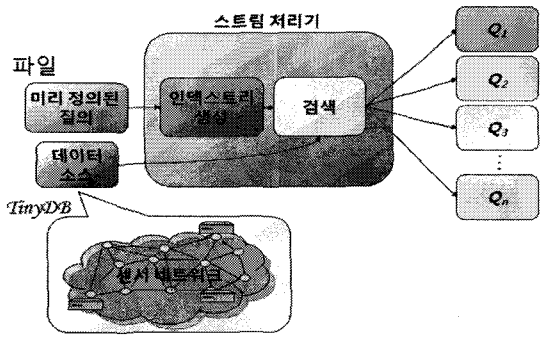
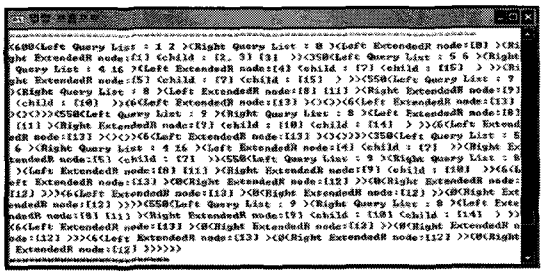


그림 18. SMILE 트리가 포함된 시스템의 전체 구조

```
Q1 : light > 500
Q2 : light < 1000
Q3 : light <= 750
Q4 : 600 <= light <= 800
Q5 : temp > 200
Q6 : temp < 500
Q7 : temp < 350
Q8 : 250 <= temp < 450
Q9 : volt >= 500
Q10 : volt < 700
Q11 : 550 <= volt <= 600
Q12 : volt < 450
Q13 : light >= 600, 1 < nodeID < 8, noise = 0
Q14 : light <= 800, 2 < nodeID < 6, temp >= 300
Q15 : 500 < light <= 700, 350 < temp < 400, 550 < volt < 600
Q16 : 500 < light < 600, 300 < temp < 400
Q17 : temp >= 350
```

그림 19. 질의 예



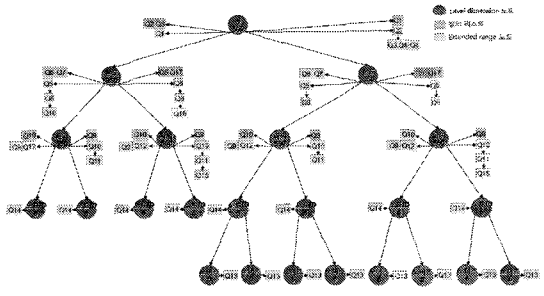


그림 20. 생성된 SMILE 트리

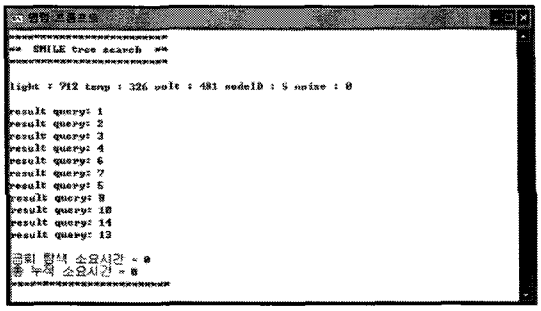


그림 21. SMILE 트리 탐색 결과

4.3 성능 평가

TinyDB에서 수집한 센서 데이터를 가지고 순차 탐색 및 CEI 기반의 탐색과 SMILE 탐색을 하여 성능을 평가하였다. 그림 22는 센서 데이터 하나하나를 탐색할 때 소요된 시간을 누적한 것으로 센서 데이터를 2000, 4000, 6000, 8000개 단위로 순차 탐색 및 CEI 기반의 탐색과 SMILE 트리를 이용한 탐색을 하였을 때 소요된 총 누적시간을 나타낸 결과이다. 순차탐색 및 CEI 기반의 탐색과 SMILE 트리에 사용된 질의의 개수는 20개이다. 소요시간은 센서 데이터 하나가 해당하는 모든 질의를 탐색할 때 소요된 시간으로 단위는 ms이다. 8000개의 스트림 데이터를 받는데 소요된 시간은 약 6시간이었고 순차 탐색 및 CEI 기반의 탐색과 SMILE 트리를 이용하여 탐색을 할 경우 탐색 속도가 빨라서 0ms 으로 나올 때가 있어 총 누적시간은 이보다 적게 나왔다. 그러나 총 소요 시간에서 볼 수 있듯이 센서 데이터를 탐색할 때 걸리는 시간을 장시간 누적하여 보면 점차 탐색하는데 소요되는 시간이 증가되는 것을 볼 수 있다. 총 소요 시간 또한 단위는 ms이다. 누적 소요 시간을 보면 SMILE 트리 탐색이 경우에 따라서 순차 탐색 소요시간의 약 50% 정도의 시간만 드는 것을 볼 수 있다.

그림 23은 질의 수를 다르게 하였을 때 2000개의 센서 데이터를 탐색하는데 소요된 시간을 나타낸 결과로 소요시간은 센서 데이터 2000개를 탐색할 때 소요된 시간을 누적한 것으로 단위는 ms이다. 질의 수를 10, 20, 40개로 점차 늘렸을 때 소요되는 누적시간을 보면 순차 탐색에 비해 SMILE 트리를 이용하여 탐색하는 경우가 소요되는 시간이 적게 드는 것을 볼 수 있으며 약 77-88% 정도의 소요시간만이 드는 것을 볼 수 있다. SMILE 트리 탐색은 CEI 기반의 탐색에 비하여 전반적으로 소요시간이 적어 조금 더 우수한 것을 볼 수 있다. 이는 질의를 색인하여 유사한 센서 스트림 데이터에 연속 질의를 수행하면서 반복되는 중복 계산을 줄여서 검색에 소요되는 시간을 줄인 것으로 볼 수 있다.

미리 정의된 질의를 전처리로 색인한 SMILE 트리는 트리 생성을 위한 시간 비용이 초기에 있으나

센서 데이터 수	소요시간		
	순차탐색	CEI 탐색	SMILE 트리 탐색
2000	3774	2883	2792
4000	17080	11279	10184
6000	39171	24021	21962
8000	84601	41172	37282

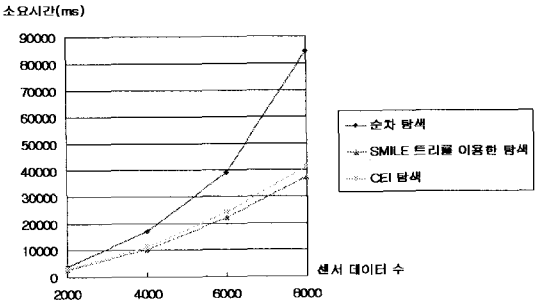


그림 22. 성능 평가 결과 I

질의 수	소요시간(2000개의 센서데이터)		
	순차탐색	CEI 탐색	SMILE 트리 탐색
10	2595	2311	2281
20	3471	2956	2877
30	6554	5347	5028
40	10069	8202	7669

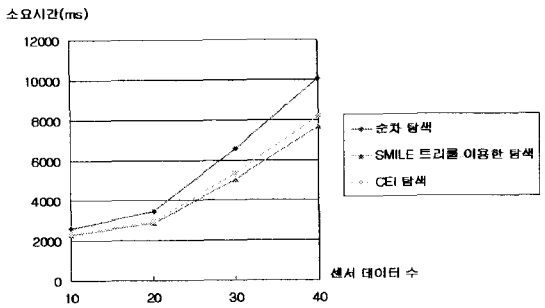


그림 23. 성능 평가 결과 II

그 비용이 본 실험의 40개 질의인 경우 10ms와 같이 대부분 몇 ms로 매우 적다. 처음에 질의를 등록할 때 외에도 질의가 새로 추가되거나 변경, 혹은 삭제될 때에 추가적인 인덱스 생성 및 관리와 관련한 비용이 든다. 사실 감시 목적의 센서 네트워크에서는 주로 초기에 질의가 많이 설정되며 주로 스트림 데이터 처리 시간동안은 추가나 변경, 삭제의 발생 빈도가 매우 적다. 그리하여 기본적으로 인덱스를 재생성하는 발생 빈도가 적으므로 이러한 비용이 검색 속도를 저해하게 되는 상황은 적다고 볼 수 있다. 결국 SMILE 트리를 생성한 후에는 유지비용이 들지 않아 많은 양의 스트림 데이터를 장시간 받아서 검색할 때에는 검색 속도 및 시간 면에서 그 비용이 매우 적다고 볼 수 있다.

메모리 사용 측면에서는 초기에 색인을 생성하여 유지하면서 계속적으로 색인을 위한 메모리 오버헤드가 있으나 이는 매우 적은 메모리만을 필요로 한다. 본 실험의 40개 질의의 경우에 1K 바이트 이하의 메모리만을 필요로 하여 TinyDB에서 충분히 수용할 수 있는 오버헤드이다. 그리고 이러한 메모리 오버헤드는 존재하지만 검색 시간이 줄어들면서 검색에 드는 메모리 사용이 보다 짧은 시간동안만 효율적으로 이루어지므로 일부 메모리 오버헤드를 보상받을 수 있게 된다. 즉, 적은 양의 추가 메모리를 사용하면서 검색 시간은 상대적으로 매우 많이 줄어든다는 것이다.

4. 결론 및 향후 연구

센서 네트워크는 무수한 센서들의 통신으로 이루어진 네트워크로 주로 감시하는 시스템, 즉 자연재해 방지, 교통 텔레매틱스, 동물 서식지 관찰 등에 사용된다. 이러한 센서 네트워크에서는 배터리 기반의 전원과 소형 장치 제약성과 제한된 메모리를 고려하여야 한다. 본 논문에서는 센서 네트워크의 특성을 고려하여 다중 질의를 효율적으로 처리할 수 있는 방법으로 다중 질의 색인 기법인 SMILE 트리를 제안하고 TinyDB를 기반으로 구현하였다. 제안한 새로운 트리인 SMILE (Stream data Multi-query Indexing with Level dimension nodes & Extended range nodes) 트리는 감시 목적의 시스템에 미리 정의된 다중 질의를 색인하는 기법으로 공간 색인하는 기법들 중에서 k-d 트리와 R+-트리의 개념을 적용 및 융합하였다. TinyDB에 SMILE 트리를 실질적으로 구현하고 적용하여 센서 네트워크에서 실제 수집되는 센서 데이터를 입력으로 SMILE 트리의 탐색 성능을 평가해본 결과 순차적으로 탐색하는 경우와 비교하여 경우에 따라서는 약 2배 정도 빠른 성능을 나타내고 기존의 CEI 기반의 기법보다 우수한 성능을 낸다.

향후에는 SMILE 트리에 애드혹(Ad hoc) 질의를 적용할 수 있는 방법을 연구할 필요가 있다. 현재 SMILE 트리는 미리 정의된 질의이며 연속 질의인 경우만을 고려하고 있어 애드혹 질의나 일회성 질의가 오게 되면 시스템을 종료하고 색인을 재생성하게 되고, OR 연산을 할 경우 질의를 여러 질의로 나눠서 시스템에 등록해야 하는 번거로움이 있어 이와 관련된 연구도 필요하다.

참 고 문 헌

[1] B. Bobcock, S. Babu, M. Datar, R. Motwani, and J. Widom, "Models and Issues in Data Stream Systems," *Proceedings of 21st Symposium on Principles of Database Systems*, pp. 1-16, 2002.

[2] STREAM project, http://infolab.stanford.edu/stream/.

[3] K. Wu, S, Chen, and S. Yu, "Query indexing with containment-encoded intervals for efficient stream processing," *Knowledge and Information Systems*, Vol.9, No.1, pp. 62-90, 2006.

[4] TinyDB, http://berkeley.intel-research.net/tinydb/.

[5] J. Gehrke and S. Madden, "Query Processing in Sensor Networks," *IEEE ComSoc*, 2004.

[6] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "TinyDB: An Acquisitional Query Processing System for Sensor Networks," *ACM transactions on Database Systems*, Vol.30, No.1, pp. 122-173, March 2005.

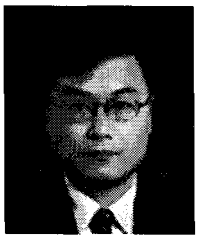
[7] C. Bohm, S. Berchtold, and D. A. Keim, "Searching in High-Dimensional Spaces -

Index Structures for Improving the Performance of Multimedia Databases," *ACM Computing Surveys*, Vol.33, No.3, pp. 323-373, September 2001.

[8] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Communications of the ACM*, Vol.18, No.9, pp. 509-517, 1975.

[9] A. Guttman, "R-Trees-A Dynamic Index Structure for Spatial Searching," Proc. of 1984 *ACM SIGMOD Conference*, pp. 47-57, 1984.

[10] T. Sellis, N. Roussopoulos, and C. Faloutsos, "The R+-Tree : A Dynamic Index for Multi-Dimensional Objects," Proc. *13th VLDB Conference*, pp. 507-518, 1987.



이 민 수

1992년 서울대학교 컴퓨터공학과 학사
1995년 서울대 대학원 컴퓨터공학과 공학석사
1995년~1996년 LG전자 미디어통신연구소 연구원
2000년 University of Florida 컴퓨터공학과 공학박사
2000년~2002년 미국 Oracle Corporation, Senior Member of Technical Staff
2002년~현재 이화여자대학교 컴퓨터학과 조교수
관심분야 : 데이터웨어하우스, 지식기반 시스템, 웹 데이터베이스

김 연 정

2005년 이화여자대학교 컴퓨터학과 졸업(학사)
2007년 이화여자대학교 컴퓨터학과 졸업(공학석사)
2007년~현재 LIG 넥스원 연구원
관심분야 : 스트림데이터처리, 센서 네트워크, 바이오 데이터마이닝

윤 혜 정

2002년 덕성여자대학교 전산학과 졸업(학사)
2005년 이화여자대학교 대학원 컴퓨터학과 졸업(공학석사)
2007년 이화여자대학교 대학원 박사과정
관심분야 : 워크플로우, 시맨틱 웹 서비스, 스트림데이터처리, 바이오 데이터마이닝