효율적인 센서 네트워크 관리를 위한 다중 연속질의 분할

박 정 업[†] · 조 명 현[††] · 김 학 수[†††] · 이 동 호[††††] · 손 진 현[†††††]

요 약

최근 센서네트워크에 관련된 많은 연구가 진행되고 있다. 특히, 센서의 전력 보전을 위한 많은 기술들이 개발되고 있는데, 본 논문에서는 센서 네트워크의 불필요한 전력 소비를 줄이는 다중 연속질의 최적화에 관련된 방법을 제시한다.

우리는 센서 네트워크에서 전송되는 데이터의 횟수나 전송량의 원천적 문제가 되는 다중 연속 질의의 중복성 문제를 해결하는 분할 알고리즘을 제안한다. 분할 알고리즘은 새롭게 생성된 사용자 질의와 기존의 질의 들 사이에 중첩 질의 영역을 제거하기 위해, 질의 인덱스(QR-tree)를 통해 하나의 질의를 둘 이상의 질의로 분할하는 알고리즘이다. QR-트리는 효율적인 질의 분할을 위해, R*-트리를 본 논문의 구조에 맞게 개량한 것이다. 이러한 처리 결과, 우리는 센서 네트워크의 전체 에너지에서 약 20% 가량의 에너지를 보존할 수 있었다.

키워드 : 센서 네트워크, 연속질의, 다중 연속질의 최적화

The Multiple Continuous Query Fragmentation for the Efficient Sensor Network Management

Jung Up Park[†] · Myung Hyun Jo[††] · Hak Soo Kim[†††] · Dong-Ho Lee[††††] · Jin Hyun Son[†††††]

ABSTRACT

In the past few years, the research of sensor networks is forced dramatically. Specially, while the research for maintaining the power of a sensor is focused, we are also concerned with query processing related with the optimization of multiple continuous queries for decreasing in unnecessary energy consumption of sensor networks.

We present the fragmentation algorithm to solve the redundancy problem in multiple continuous queries that increases in the count or the amount of transmitting data in sensor networks. The fragmentation algorithm splits one query into more than two queries using the query index (QR-tree) in order to reduce the redundant query region between a newly created query and the existing queries. The R*-tree should be reorganized to the QR-tree right to the structure suggested. In the result, we preserve 20 percentage of the total energy in the sensor networks

Key Words : Sensor Networks, Continuous Query, Multiple Query Optimization

1. 서 론

최근 몇 년간, 네트워크와 모바일 컴퓨팅에 관한 관심이 상당한 연구 분야로 다가오고 있다. 특히 특정 지역의 관찰을 위해 많은 센서들로부터 애드-혹(ad-hoc) 데이터를 추출하는 무선 센서 네트워크에 관심이 집중되고 있다. 그래서 MIT, UCLA, 그리고 UC Berkeley[7-9]는 작고, 무선이며, 적은 에너지 소모를 하는 무선 센서 개발에 참여하기 시작했다. 이와 같은 프로젝트의 목적은 컴퓨터와 센서가 일상생활의 모든 측면을 보조해주는 유비쿼터스 컴퓨팅을 구축하는 것이다.

무선 센서 네트워크에 관련된 연구는 크게 두 가지 분야로 나뉘어 진행 중이다. 첫째, 센서 및 센서 네트워크를 구축, 관리하는 연구 분야이다. 이것은 적은 에너지 소모를 위한 저 전력(low-power) 센서를 개발할 뿐만 아니라, 센서 네트워크에 적용되는 토폴로지 및 프로토콜도 개발한다. 특히 센서는 제한된 배터리를 갖기 때문에, 센서 수명 연장에 관련된 연구가 이 분야에서 가장 주목할 만한 논제라고 할 수 있다. 하지만 센서 자체의 에너지를 확장시키기 위한 배터리 개발보다는 센서에 전송되는 데이터를 줄이기 위한 연구가 더욱 활발히 진행되고 있다. 일반적으로 센서는 샘플링 데이터를 계산하는 에너지(Computing cost)보다 샘플링 데이터를 전송하는 에너지(Transmitting cost)가 더 많이 소비된다[10]. 그래서 센서들 간에 전송되는 데이터 횟수 및

※ 본 연구는 정보통신부 및 정보통신연구진흥원의 대학 IT연구센터 육성·지원 사업(IITA-2006-C1090-0603-0031)의 연구결과로 수행되었음
† 준 회 원 : 한양대학교 컴퓨터공학과 석사과정
†† 준 회 원 : 코난테크놀로지 근무
††† 준 회 원 : 한양대학교 컴퓨터공학과 박사과정
†††† 정 회 원 : 한양대학교 컴퓨터공학과 조교수
††††† 종신회원 : 한양대학교 컴퓨터공학과 조교수
논문접수 : 2006년 7월 22일, 심사완료 : 2006년 10월 2일

데이터양을 줄이기 위해 TAG[5], TiNA[11]와 같은 연구가 진행되고 있다.

둘째, 센서로부터 전송된 데이터를 처리하고, 센서로 전송될 연속질의를 처리하기 위한 서버 측면의 연구가 진행되고 있다. 관계형 데이터베이스는 과거(Historical) 질의[2]에 대해서만 처리를 할 수 있고, 질의에 대한 응답은 오직 한번만 일어난다. 하지만 센서 네트워크에 대한 질의는 한 번의 질의에 계속적인 응답을 받을 수 있고, 미래에 대한 질의도 할 수 있어야 한다[2]. 즉, 센서 네트워크는 광역의 온도나 습도 데이터를 연속적으로 전송하기 때문에, 기존 관계형 데이터베이스에 저장되기에 적합하지 않다. 그래서 STREAM [12], Aurora[13], TelegraphCQ[3] 등의 프로젝트에서 센서 네트워크에 대한 연속질의 및 데이터 관리에 대한 기술을 연구하고 있다[1].

본 논문은 센서 네트워크 두 분야의 중간에 위치한 연구로써, 본 연구의 궁극적인 목적은 센서의 수명을 연장시키기 위한 것이다. 하지만 기존 연구처럼 센서 네트워크에 구성된 센서들 사이의 토폴로지나 프로토콜에 관련된 알고리즘을 개발하는 것이 아니라, 사용자에서 센서로 전해지는 다중 연속질의의 중복 영역을 최적화하여 센서의 불필요한 전력 소모량을 최소화하고자 한다. 다중 연속질의들 사이의 중첩 문제를 해결하기 위해, 본 논문에서는 질의 인덱스 QR-트리를 제안한다.

본 논문의 구성은 다음과 같다. 제 2장에서는 기존 연속질의 최적화 방법의 장·단점을 기술한다. 제 3장에서는 기반 기술인 센서 네트워크와 연속질의에 대해 간단히 기술하고, 본 연구의 동기를 기술한다. 제 4장에서는 다중 연속질의 분할의 세 단계를 기술한다. 제 5장에서는 실험 및 결과를 통해 분할 알고리즘에 대한 평가를 하고, 제 6장에서 결론을 맺는다.

2. 관련 연구

연속질의는 센서 네트워크가 출현하기 이전부터, 다양하게 연구되었다. 연속질의는 처음으로 1992년 Terry에 의해 제안되었다[14]. Terry는 첨부(append) 기반의 관계형 데이터베이스에서 연속질의를 처리하는 연구를 제안했다. Terry의 연속질의는 저장 프로시저처럼 일정한 시간적 주기에 따라 질의가 수행된다. 이것은 시간(Timer) 기반 연속질의로써, 본 논문에서 사용하는 연속질의와 유사한 의미를 갖고 있다. OpenCQ[15]는 분산 환경의 정보를 수집하기 위해 연속질의를 사용하였다. OpenCQ는 변동(Change) 기반과 시간(Timer) 기반의 연속질의 모두를 처리하였다. 하지만, Tapestry와 OpenCQ에는 모두 연속질의에 대한 최적화가 논의되지 않았다.

일반적으로, 관계형 데이터베이스의 다중 연속질의에 대한 최적화는 1988년 Sellis[16]에 의해 처음 시작되었다. Sellis는 다중 연속질의에 대한 최적화를 위해, 질의의 각 조건식(predicate)들의 관계성을 정립하여 총체적 계획(global plan)을 구성하였다. 다중 연속질의의 동일한 조건식들은 질의 영역에 따라 내포(imply)관계로 구성된 조건식들을 하나의 총체적 계획으로 구성하는 것이 Sellis 논문의 큰 장점이다. 하지만 총체적 계획 때문에, 각 질의 predicate가 서로 반대로 내포할 수 있는 경우를 처리할 수 없고, 내포가 아닌 일부 중복에 대한 최적화도 할 수 없다. 2001년, Chen이 제안한 NiagaraCQ[4]는 Sellis가 제안한 질의의 최적화를 동적인 상황에 맞게 최적화하기 위한 단계적 증식 방법을 제안했다. 이것은 중복되는 서브 표현들을 그들만의 서명(signature)을 통해 그룹핑을 시도한 것이다. 2002년, Madden의 CACQ[6]는 NiagaraCQ보다 좀 더 유연한 방법으로 다중 연속질의를 그룹핑하였다. 마지막으로, 2003년에 Yousuke[17]가 제안한 다중 연속질의 최적화 방법이 본 논문의 다중 연속질의 최적화 방법과 유사한데, Yousuke는 다중 연속질의의 시간 범위를 유사성 행렬을 이용해 클러스터링 시킨 후, DAG에 의해 질의 계획(Query Plan)을 최적화 시켰다.

기존 데이터베이스는 질의와 관계없이 다양한 데이터들을 보유하고 있다. 하지만, 센서 네트워크의 서버는 센서가 질의한 결과만을 보유하기 때문에, 서버에 관리되는 데이터들은 사용자가 정의한 질의와 의존적이다. 또, 기존 데이터베이스의 질의는 현재 보유하고 있는 데이터들만을 대상으로 하는 과거 질의(Past Query)이다. 하지만 센서 네트워크의 연속질의는 현재 이후에 센서들이 측정한 데이터들만을 대상으로 하기 때문에 미래 질의(Future Query)이다. 이런 기존 데이터베이스와의 차이 때문에 기존의 다중 연속질의의 최적화 방법은 적용할 수 없다. 기존 최적화 방법들은 공통되는 서브 표현들을 서버에서 그룹핑시키는 방법을 제안하였다. 이것은 공통된 서브 표현이 서버에서만 최적화되기 때문에, 센서에서는 최적화되지 않는다. 즉, 센서는 최적화되지 않게 질의를 처리를 함으로써, 불필요한 에너지가 소모하게 된다. 다시 말해서, 공통된 서브 표현들 사이에 발생하는 중간 결과(Intermediate Result)를 공유함으로써 질의의 수행속도를 개선시키지만, 센서 네트워크에서는 질의를 수행하는 센서와 데이터를 수집하는 서버가 이질적인 장소이기 때문에 데이터 공유가 이루어질 수 없다. 사실, 각 센서 노드마다 위의 질의 최적화 시스템이 구성된다면, 기존 데이터베이스 시스템과 동일하게 처리될 수 있다. 하지만 센서는 제한된 컴퓨팅 능력과 파워를 갖기 때문에, 다중 연속질의의 최적화 시스템을 모두 장착한다는 것은 현실적으로 불가능하다. 그러므로 센서의 저 전력을 위해서는 질의가 센서에 전달되기 이전에, 다중 연속질의의 최적화를 위해 질의의 재구성이 수행되어야 한다.

3. 기반 기술 및 문제 정의

본 연구는 야생 동물의 생태계나 해수의 조수 간만 차등을 관찰하기 위한 광역의 무선 센서 네트워크를 기반으로

하고 있다. 본 장에서는 본 연구에서 가정하는 무선 센서 네트워크의 기본적인 특징과 센서 네트워크에서 사용하는 TinyDB의 연속질의(Continuous Query)에 대해 간단히 기술한다.

3.1 무선 센서 네트워크 (Wireless Sensor Networks)

무선 센서 네트워크(WSN)는 많은 수의 작고 이질적인 센서 노드들 간의 네트워크를 의미하는데, 각 센서 노드들은 전원 유닛, 센싱 유닛, 컴퓨팅 유닛, 그리고 통신 유닛을 포함하고 있다. 이질적인 노드들 간의 무선 통신을 지원한다는 면에서 Ad-hoc 네트워크와 유사한 성격을 가진다. 하지만 각 노드들이 제한된 리소스의 단순한 구조를 가졌다는 점, 범용의 목적을 위한 네트워크가 아니라는 점, 그리고 통신에 있어서 데이터 중심(Data Centric)이라는 점에서 차이점을 가지고 있다. Ad-hoc 네트워크의 경우는 리소스의 제한이 없으며, 범용 목적을 위하여 구성되었으며, 통신에서 노드 중심(Node Centric)의 특징을 가지고 있다.

스마트 센서들은 저렴하고 작은 크기의 컴퓨터이다. 스마트 센서는 새로운 정보를 취득하기 위한 센싱 부분, 취득한 정보를 저장하고 분석하기 위한 컴퓨팅 부분, 분석된 정보를 제공하기 위한 무선통신 부분 그리고 전원 부분으로 구성되어 있으며, 각 센서 노드들이 정보를 실시간으로 수집할 수 있다. 또한 무선통신을 통하여 인접 센서노드와 통신을 할 수 있으며, 센서 노드들이 수집된 정보를 스스로 분석하고 처리할 수 있다. 지금까지 많은 센서들이 개발되었는데, 본 연구에서는 Greate Duck Island와 James Reserve[18]에 설치되어 연구된 적이 있는 Mica 플랫폼[18]을 기반으로 한다. Mica mote는 4Mhz, 8bit Atmel 마이크로프로세서, 초당 40kbit를 전송할 수 있는 RFM TR1000 라디오, 512kbyte의 플래시 메모리로 구성되어 있다. 센서는 제한된 배터리를 갖는데, 센서에서 다음의 세 가지 부분이 가장 많은 에너지를 소모한다[18,19]. 이웃 노드에 패킷을 전송하는 요소(sending cost is $1\mu J/bit$ [19]), 패킷을 전송 받는 요소(sending cost is $0.5\mu J/bit$ [19]), 전송받는 패킷을 버퍼에 쓰는 요소(writing/erasing cost is $1\mu J/byte$ [19])이다. 그래서 36bytes 크기의 한 패킷에 필요한 에너지양은 <표 1>과 같다.

물리적 환경의 관찰은 센서 네트워크에 연결된 모든 센서들에게 연속질의가 전달됨에 따라 측정될 수 있다. 다시 말해서, 일반 사용자에 의해 생성된 다수의 연속질의들이 모든 센서에 전달되고, 센서들은 질의를 처리하여 질의 결과를 주기적으로 서버에 반환한다.

〈표 1〉 한 패킷에 대한 에너지 양

요소	energy(μJ)
한 패킷을 송신하는 에너지	286
한 패킷을 수신하는 에너지	143
한 패킷을 쓰고/삭제하는 에너지	30

3.2 연속질의 (Continuous Query in TinyDB)

TinyDB[20]는 TinyOS[21]를 기반 한 센서들로부터 정보를 추출하기 위한 질의 처리 시스템이다. TinyOS의 기존 데이터 처리 방법과 달리, TinyDB는 사용자가 센서에 내장형 C 코드를 작성하는 것을 요구하지 않는다. 대신에, TinDB는 간단한 SQL 질의 인터페이스를 제공한다. 이것은 기존의 SQL과는 달리 연속적인 데이터를 추출할 수 있는 연속질의(Continuous Query) 형태로 구성된다. 연속질의는 그들의 실행에 사용되는 범주에 따라 두 가지로 분류된다. 하나는 측정 데이터의 변경에 따른 변동(Change)-기반 연속질의이며, 다른 하나는 일정 주기마다 질의를 실행시키는 시간(Timer)-기반 연속질의이다[4]. 본 논문의 센서 네트워크는 자연 현상을 계속적으로 관찰하기 위한 것이기 때문에, 시간 기반 연속질의를 사용한다. (그림 1)은 사용자가 관찰 시간을 지정할 수 있도록, TinyDB에 사용되는 연속질의[20]이다. 본 논문은 TinyDB 연속질의의 전체 스키마를 이용하지 않고, (그림 1-(a))의 문법 구조를 가지는 제한된 질의에 대해서만 질의 최적화를 제공한다. 그 예는 (그림 1-(b))와 같다.

TinyDB는 'Sensors'라는 하나의 테이블 만으로 구성되며, 센서 노드가 포함하는 센서 타입들은 'Sensor' 테이블의 컬럼에 해당한다. TinyDB의 이런 특징들 덕분에, 조인이나 재귀질의와 같은 복잡한 질의는 많이 사용되지 않는다. 그래서 본 논문도 이와 같이 복잡한 질의에 대한 최적화는 고려하지 않는다. 센서로부터 측정된 데이터는 'Sensors' 테이블에 튜플로 주기적으로 저장된다. SAMPLE INTERVAL 절과 LIFETIME 절은 질의 결과를 서버에 연속적으로 반환하기 위한 요소로써, 두 절중 하나만이 선택적으로 사용될 수 있다. SAMPLE INTERVAL 절은 사용자에게 질의 결과의 반환 주기를 직접적으로 정의하게 함으로써, 사용자에 의해 질의가 제거되지 않는 한 질의 결과를 계속해서 반환하게 한다. 이것은 seconds 단위의 정수 값으로 정의된다. 반면에, LIFETIME 절은 사용자에게 질의 유효 시간을 정의하게 함으로써, 질의가 서버에 초기화되면서 부터 정의된 시간 영역까

```
SELECT *
FROM Sensors
WHERE {A op constant}*
[SAMPLE INTERVAL period]
[LIFETIME period]
```

(a)

```
SELECT *
FROM Sensors
WHERE 온도 < 30 and 습도 > 20
SAMPLE INTERVAL 30 seconds
or
LIFETIME 2 days
```

(b)

(그림 1) 시간 기반 연속질의

지만 질의 결과를 반환하게 한다. 이것은 days/hours/minutes 단위의 정수 값으로 정의된다. WHERE 절은 (그림 1)처럼 정의되는데, 'A'는 'Sensors' 테이블의 하나의 컬럼을 지칭하며, op $\in$ { =, <, >, ≤, ≥}를 의미한다. 그리고 constant는 실수 값만을 의미하는데, 센서는 습도, 온도와 같은 실수 데이터만을 측정하기 때문이다.

물리적 환경의 데이터를 측정하는 센서 네트워크에서 센서들은 제한된 도메인을 갖는다. 예를 들어, 만일 온도 센서가 영하 50°C나 영상 100°C 이상의 값을 측정한다면, 센서 장치는 작동하지 않을 것이다. 제한된 도메인은 WHERE 절의 각 조건식(Predicate)을 가정 1로써 새롭게 정의할 수 있다. 이것은 질의 영역을 hyper-rectangle 형태로 구성하기 때문에, 다중 연속질의들 사이에 비교를 쉽게 해준다.

가정 1. 바운드 조건식 (t_j)

레코드가 $R[A_1, A_2, .., A_n]$ 라면, 바운드 조건식(t_j)은 t_j : lValue Θ A_i Θ rValue , where $\Theta \in$ { =, <, >, ≤, ≥}, lValue, rValue $\in D_i$,D_i는 A_i의 도메인이고, lValue ≤ rValue

일반적으로 사용자는 'Sensors' 테이블의 모든 컬럼을 WHERE 절에 포함시켜 질의하는 것은 아니기 때문에, 두 개의 질의를 비교하는 것이 쉽지 않다. 하지만 WHERE 절에 정의되지 않는 필드를 가정 1에 의해, 시스템 하한 값과 상한 값을 정의한다면 두 질의는 모두 동일한 필드를 갖고 질의하기 때문에 비교하기 쉽다. 예를 들어, 'Sensors' 테이블의 필드가 온도와 습도만으로 구성되었다고 가정할 때,

Q1	Q2
SELECT * FROM Sensors WHERE *temperature* > 30 and *humidity* > 20 SIMPLE PERIOD 1024	SELECT * FROM Sensors WHERE 50 > *temperature* SIMPLE PERIOD 1024

(그림 2) 일반적인 연속질의의 예제

Q1'	Q2'
SELECT * FROM Sensors WHERE 30 < *temperature* < 100 and 20 < *humidity* < 100 SIMPLE PERIOD 1024	SELECT * FROM Sensors WHERE 0 < *temperature* < 50 and 0 < *humidity* < 100 SIMPLE PERIOD 1024

(그림 3) 재구성된 연속질의의 예제

사용자가 (그림 2)의 Q1, Q2의 질의를 생성했다고 하자. (그림 2)의 Q1은 조건식들에 상한 값을 포함하고 있지 않다. 반면에 Q2의 조건식들은 온도의 상한 값만을 갖기 때문에, 두 질의를 비교하는 것은 쉽지 않다. 또, Q2는 Q1에 비해, 습도 필드를 조건식에 정의하고 있지 않기 때문에, Q1과 비교하기 쉽지 않다. 하지만 가정 1에 의해 습도와 온도가 잠재적으로 상한 값(100)과 하한 값(0)을 갖는다고 가정하면, (그림 2)는 (그림 3)처럼 재구성(rewriting)될 수 있다. (그림 3)의 Q1'와 Q2'의 WHERE 절은 모두 공통의 바운드된 필드로 구성되었기 때문에, 비교하기 쉽고 두 질의의 중복성을 간단하게 판단할 수 있다.

3.3 문제 정의 (Problem Formulation)

본 절은 본 논문에서 가정한 센서 네트워크에서 연속질의를 처리함으로써 발생할 수 있는 문제점을 정형화한다. 물리적 환경을 측정하기 위해서는 모든 센서들이 동일한 질의를 처리해야 한다. 처리된 질의 결과는 패킷 형태로 서버에 전달되는데, 계산의 용이성을 위해 하나의 튜플이 한 개의 패킷으로 변환되어 전송된다고 가정한다. 튜플은 (그림 4)와 같은 형태처럼 패킷화 되어있다[20,21].

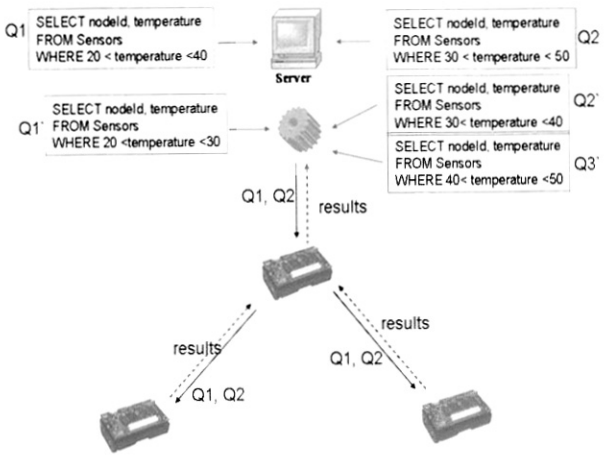
여러 사용자에 의해 구성된 질의는 상당히 다양하지만, (그림 3)처럼 질의들 사이에 중복된 영역을 포함할 수 있다. (그림 3)의 Q1'와 Q2'는 온도에서 30이상~50이하인 영역을, 습도에서 20이상~100이하인 영역에서 중첩된 영역을 갖는 것을 알 수 있다. 이렇게 중복된 영역에서 질의 결과가 발생한다면, 두 질의의 실행 결과는 중복될 것이다.

센서 네트워크에서 이런 중복된 영역의 질의들은 센서에게 중복된 패킷을 전송하게 함으로써 불필요한 에너지를 소모하게 한다. 사실 센서 네트워크는 일반적으로 (그림 5)와 같은 트리 구조[20,21]로 구성되기 때문에, 중복된 패킷이 발생한다면 발생한 센서 노드뿐만 아니라, 노드의 상위 노드들도 불필요한 에너지를 소모하게 된다. 또, 데이터들은 연속적으로 전송되는 스트림형태이기 때문에, 중복된 데이터가 발생하면 연속적으로 중복 패킷을 전송해야 한다. 즉, 센서 네트워크의 생명 주기를 짧게 하는 문제를 초래한다. 만일 중복된 패킷을 전송하지 않는다면, 센서 네트워크는 많은 에너지를 절약할 수 있다. <표 1>처럼, 센서 네트워크에서 가장 많은 에너지를 소모하는 세 요소만을 고려한다면, 중복 패킷의 전송 여부에 따라 다음과 같이 에너지를 보전할 수 있다. 즉, (그림 5)처럼 Q1과 Q2를 센서에 전달하지 않고, Q1', Q2', Q3'을 전달하여 센서의 에너지를 보전하는 것이다. 전체 에너지 소비량에 대한 식은 다음과 같다.

SELECT [illegible]
FROM sensors

Field Name : Size (bytes)	QID : 1	numFields : 1	notnull : 4	field 1 : size(1) ... field n : size (n)
Sample Data For Query	1	[illegible]	[illegible]	

(그림 4) TinyDB에 저장된 튜플 포맷[20]



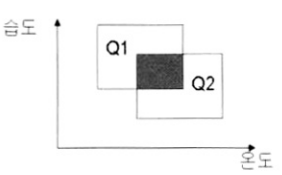
(그림 5) 센서 네트워크의 질의 처리

$$\sum_{i=1}^{N} d_i*(E_S+E_R+E_W)*(T_i^D-T_i),$$

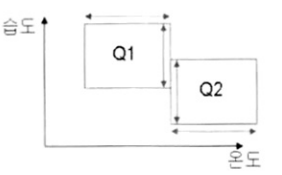
where N: 총 센서 노드 개수, d: 센서 노드의 깊이, E_S: 한 패킷을 전송하는데 필요한 에너지,
E_R : 한 패킷을 수신하는데 필요한 에너지, E_W: 한 패킷을 버퍼에 쓰는데 필요한 에너지,
T^D : 중첩 질의의 실행 결과인 튜플 개수, T: 중첩되지 않은 질의의 실행 결과인 튜플 개수

센서네트워크에서 중복된 패킷을 발생하는 원천적인 문제는 사용자가 정의한 다중 연속질의들 간의 중복 질의 영역이 발생하기 때문이다. 중복 영역의 요소는 WHERE 절의 조건식들이 중복된 영역을 갖기 때문이다(Selection). 다중 연속질의들 간의 중복 영역이 발생한다는 것은 (그림 6)처럼 조건식의 모든 필드들이 중복되었다는 것을 의미한다. 예를 들어, 두 질의가 습도에 대해 중복 영역을 갖지만 온도는 중복 영역을 갖지 않는다면, (그림 7)처럼 두 질의는 중복 영역을 갖지 않는다. 결국, 중첩된 질의 결과를 생성하지 않는다. 질의들은 가정 1에 의해 모든 필드가 제한된 영역을 갖기 때문에, (그림 6, 7)과 같이 hyper-rectangle로 표현될 수 있다.

질의 영역은 센서에서 제공하는 필드의 개수에 따라 n 차원의 유클리디언 공간으로 표현될 수 있다. 그래서 질의 영역 R은 (S,T)인 두 꼭지점으로 표현될 수 있는데, $S=[s_1, s_2, \ldots, s_n]$ 이고 $T=[t_1, t_2, \ldots, t_n]$, $s_i \le t_i$ for $1 \le i \le n$ 이다. 이러한 정의를 기반으로 두 질의 영역이 중첩되어 있다는 것은 정의 1처럼 명시할 수 있다.



(그림 6) 중첩 영역을 갖는 다중 연속질의



(그림 7) 중첩 영역을 갖지 않는 다중 연속질의

정의 1. 중첩성 (Ω)

두 질의 영역 R_1 과 R_2 가 중첩되어 있다는 것은 아래의 조건을 만족한다는 것을 의미한다.

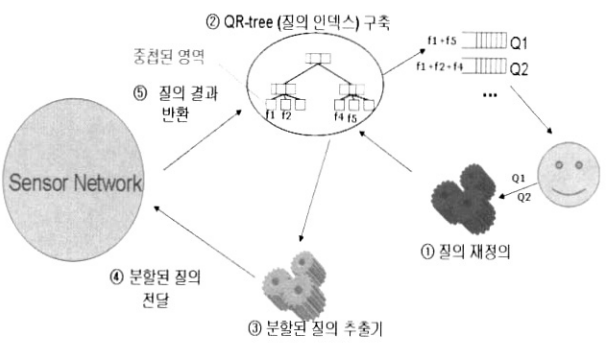
$$\forall D_i = true \; and \; D_i = \left\{ |C_1^i - C_2^i| < \frac{L_1^i + L_2^i}{2} \right\}$$

i는 차원축을 나타내며 $1 \le i \le n$, n 은 필드의 개수이며, L은 꼭지점 사이의 거리, C는 꼭지점들의 중심을 말한다. ($L = t_i - s_i$,$C = (t_i + s_i)/2$)

4. 다중 연속질의 분할 (Multiple Continuous Query Fragmentation)

다중 연속질의들 사이의 중복성을 제거하기 위해서는 몇 가지 단계가 필요하다. 그림 8은 사용자로부터 센서 네트워크로 질의가 전달하는 과정에서 필요한 몇 가지 모듈들을 명시한다. 일단 질의들 사이의 중복된 영역을 쉽게 비교하기 위해서 질의들을 동일한 형식으로 재구성(Wrapping) 할 필요가 있다. 재구성 단계에서는 가정 1에서 기술한 것처럼 조건식들을 바운드하거나, WHERE절의 OR 연산자를 제거하기 위해 질의를 분할하는 과정이 필요하다. 재구성 단계 이후에서는 질의 인덱스에 질의를 삽입하는 단계가 필요하다. 그 이후에 질의 인덱스에 질의를 삽입하여 질의 인덱스를 재구성하게 되면, 실제 센서에 전달할 분할 질의를 추출하는 단계가 필요하다. 이런 과정으로 센서 네트워크에 전달된 질의들은 질의에 따라 결과를 베이스스테이션에 전달하게 된다. 전달된 질의 결과는 원본 결과로 사용자에게 반환되기 위해 질의 인덱스를 통해 필터링 및 병합이 이루어진다.

중첩된 질의들을 분할할 때 고려되어야 하는 요소는 크게 두 가지이다. 하나는 분할된 질의의 개수를 최소화하는 것이다. 그 이유는 분할된 질의 개수가 많아질수록, 센서에게 불필요한 프로세싱이 요구되어 부가적인 에너지 소모를 유발할 수 있기 때문이다. 원본 질의의 개수와 변경이 없으면서, 중첩된 질의 영역을 제거하기 위한 방법이 연구되어야 한다. 다른 하나는, 연속질의는 오랜 시간동안 질의에 대한 결과를 반환하기 때문에, 베이스스테이션에 상당히 많은 연



(그림 8) 다중 연속질의 분할 전략

속질의들이 존재하게 될 것이다. 그래서 삽입 질의와 베이스스테이션에 존재하는 모든 질의들과 중첩된 영역을 비교하는 것은 상당히 많은 시간을 필요로 하기 때문에, 최대한 빨리 분할 질의를 추출할 수 있는 연구가 필요하다. 질의 하나를 삽입해도 분할된 질의들이 비교과정 중에 생성될 수 있기 때문에, 분할된 질의들을 다시 다른 모든 질의들과 비교하는 것은 더 많은 시간을 필요로 한다. 또한, 사용자가 원하는 정보가 분할 질의 획득을 위한 처리 시간 때문에 손실되어서는 안 된다. 본 연구는 두 요소 중 두 번째 요소를 해결하기 위한 질의 인덱스(QR-tree)를 제안한다.

4.1 질의 재구성

다중 질의를 비교하기 쉽게 구성하는 재구성 단계는 크게 세 단계로 구성된다. 첫 번째 단계로, 가정 1에 의해 다중 질의들 사이의 WHERE 절에 모두 동일한 필드를 갖도록 시스템 상한 값과 하한 값을 결정하는 것이다. 시스템 상한 값과 하한 값은 센서가 측정할 수 있는 한계 값을 기반으로 하여 결정된다. 두 번째 단계로, WHERE 절의 OR 연산자를 제거하기 위해 WHERE절의 조건식을 DNF (Disjunctive Normal Form) 형태로 변경해야 한다. OR 연산자로 구성된 질의에 대한 결과는 OR 연산자마다 분할된 질의 결과의 합과 동일하기 때문에, DNF로 구성된 조건식을 OR 연산자에 따라 질의를 분리해 OR 연산자를 제거해야 한다. 마지막 단계로, 연속질의의 시간 절(LIFETIME과 SAMPLE PERIOD)을 분리할 필요가 있다. 즉, LIFETIME 절로 구성된 질의들과 SAMPLE PERIOD 절로 구성된 질의들로 구별하여 다중 질의에 대한 인덱스를 구축할 필요가 있다. SAMPLE PERIOD는 지정된 값의 최대 공약수 형태로 질의들을 클러스터링 해야 한다.

4.2 QR-트리

QR-트리는 질의들 사이의 중첩 영역을 빨리 제거하기 위해 R*-트리[22]를 응용한 질의 인덱스이다. QR-트리는 R*-트리가 가지고 있는 일반적인 속성을 가지고 있다. 하나의 버킷이 여러 개의 노드들을 포함하는 밸런스 트리이다. 정의 2는 버킷과 노드가 포함하는 속성을 의미한다.

정의 2. 버킷(B) 과 노드(n)

$$B = \{ N, B, m \}$$

where N: 노드(n)들의 집합, B: 링크로 연결된 버킷

$$n = \{ R, Q, M, B \}$$

where R: 노드의 질의 영역, Q: 원본질의 집합,
B: 자식 버킷, M: 변경자

버킷은 최대 m 개수만큼 노드들을 포함하는데, 리프 버킷의 경우 B+트리의 속성에 따라 이웃 버킷의 레퍼런스를 포함한다. 노드는 질의 영역을 통해 질의를 대변하는데, 인덱스 노드들은 자식 버킷의 MBR을 자신의 질의 영역으로 갖는다. 인덱스 노드들은 자식 노드들의 영역을 모두 포함하기 때문에 질의 영역(R)에 따라 가지치기가 가능한 것이다. 원본질의 집합(Q)은 리프 노드들이 갖는 속성으로써, 사용

```
QUERYNODE newNode
Add newNode in CQS

insertQuery (BUCKET)
   BUCKET curBucket //Current Bucket
   For i := 1 to curBucket.length
        QUERYNODE curNode := curBucket.Node_i
        For j :=1 to CQS.length
             QUERYNODE newNode := CQS.Node_j
     If curBucket.type = LEAF Then
          fragmentLeafNodes(curBucket, curNode, newNode) //Redundant CASE
     Else
          If checkRedundancy(curNode, newNode) Then
               insertQuery(curNode.childBucket)

insertCleanNode (BUCKET,QUERYNODE)
    BUCKET curBucket //Current Bucket
    QUERYNODE curNode
    For i := 1 to curBucket.length
       If curBucket.Type = LEAF Then
            curNode.modifier = TRUE
            insertEntry(curBucket, curNode)
       Else
            Choose the node in curBucket whose rectangle has the smallest margin with
            the rectangle of the newNode
            curBucket := the child bucket of the node
            insertCleanNode(curBucket)
```

(그림 9) QR-트리 알고리즘

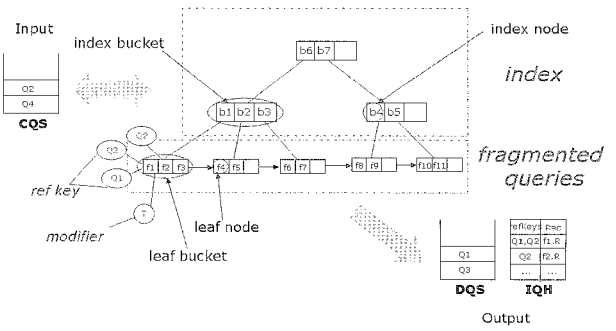
자가 전달한 실제 질의를 참조한다. 이것은 분할 질의 생성 및 질의 결과 반환 시 이용된다. 변경자(M)는 리프 노드들만 갖는 속성으로써, 질의 인덱스에서 변경된 노드들만 센서 네트워크에 전달해야 하기 때문에, 노드가 변경되었는지 유무를 파악한다. 자식 버킷(B)은 인덱스 노드들만 갖는 속성으로써, 탐색을 용이하게 한다.

버킷의 MBR은 버킷에 포함된 노드들의 모든 영역(R)들을 포함하는 최대 바운더리를 의미하고, 이것은 연결된 상위 노드의 질의 영역(R)에 대변하게 된다. 질의 영역(R)은 앞서 기술한 것처럼, 탐색의 기준을 의미하기 때문에 최대한 인덱스 노드들 끼리 중첩 영역이 없도록 구성해야 잘못된 곳을 찾아가는 일을 줄일 수 있다. 즉, 버킷에 m+1개의 노드수가 포함되어 분할하게 될 경우, 노드들을 적절하게 버킷에 분배해야 한다. 만일 버킷의 MBR을 불필요하게 크게 구성할 경우, 중첩된 질의 영역을 찾는데 오랜 시간이 걸려 결과적으로 분할 질의를 획득하는 데 오랜 시간을 소요하게 된다. 그래서 본 연구에서는 R*-트리의 장점을 받아들여 최대한 버킷의 MBR에서 여백(margin)을 최소화시킬 수 있도록 버킷을 구성하고자 한다. 만일 여백이 동일할 경우에는 두 집합의 넓이 합이 최소가 되도록 구성한다. 여백을 적게 한 이유는 탐색 실패율(fault rate)을 줄이기 위함이고, 넓이를 최소화하는 것은 영역들을 균등하게 분포시키게 위함이다. 리프 노드에서 R*-트리처럼 오버랩(overlap)의 요소는 고려하지 않고 여백과 넓이(area)만을 고려한 것은 QR-트리의 구조가 R*-트리의 구조와는 달리 리프 노드들의 영역이 모두 disjoint한 관계를 갖기 때문이다.

중첩 질의를 찾아 중첩 영역을 제거하기 위해서는 탐색하는 동안 질의 비교 스택(CQS)과 질의 삭제 스택(DQS)이 필요하다. CQS는 질의 인덱스의 노드들과 비교를 해야 하는 삽입 노드들의 집합이다. 앞서 기술한 것처럼 하나의 질의만 삽입되지만 질의가 분할됨으로써, 삽입해야 하는 노드가 여러 개가 존재하게 될 수 있다. 만일 분할된 삽입 노드들이 지금까지 비교되었던 QR-트리의 노드들과 다시 비교한다면, 탐색 시간에 많은 시간이 소요될 것이다. 그래서 분할된 삽입 노드들에 대해 다시 QR-트리를 검색하지 않도록 하기 위해 CQS가 필요하다. CQS에는 비교해야할 노드들이 삽입(Push)되고, 비교되어야 할 노드가 변경되거나 QR-트리에 삽입될 경우에는 CQS에서 재거(Pop)된다. DQS는 질의 변경에 따라 삭제되어야 하는 센서 네트워크의 질의를 포함하는 집합이다. DQS에는 QR-트리의 리프 노드들 중 변경된 노드들을 삽입(Push)한다(그림 10).

QR-트리의 구성하는 알고리즘은 (그림 9)와 같이 두 단계로 구성된다. 먼저, 삽입 노드와 중첩되는 인덱스 노드들을 DFS(Depth First Search)방법으로 탐색하고, 중첩된 리프 노드가 발견되면 삽입 노드와의 중첩 관계에 따라 분할을 수행한다. 이 단계가 수행된 후, CQS에 아직까지 삽입되지 않는 노드들을 QR-트리의 적절한 리프 버킷에 삽입해야 한다. 앞서 기술한 버킷 구성의 전략에 따라 CQS의 나머지 노드들은 여백을 최소화시킬 수 있는 리프 버킷을 찾아 삽입된다.

질의들 사이의 분할은 삽입 노드와 리프 노드들 사이의중

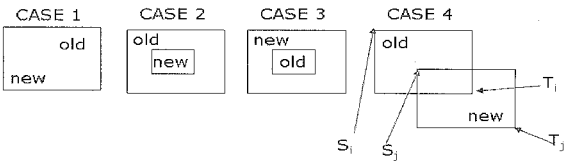


(그림 10) QR-트리 구성

첩 관계에 따라 수행된다. 사실 중첩되었다고 모든 질의를 분할하거나 새롭게 질의를 분할할 필요는 없다. 만일 완전히 동일한 질의들은 동일한 결과를 반환하기 때문에, 결과 반환 시 어떤 질의에 대한 결과인지 질의에 대한 레퍼런스만 알 수 있도록 하면 된다. 즉, 정의 3과 같이 네 가지 중첩 관계에 따라 질의 분할에 대한 알고리즘을 구별할 필요가 있다. 정의 3은 중첩 관계를 의미하는 것으로써, 질의 영역에서 발생할 수 있는 모든 중첩관계를 명시한다. 정의 3은 각 경우가 모두 독립적인 것으로, 어떤 CASE가 다른 CASE를 포함하거나 하지는 않는다.

정의 3. 중첩 관계 (Redundancy Relationship)

CASE 1: $S_i = S_j$ and $T_i = T_j$
CASE 2: $S_i < S_j$ and $T_i > T_j$
CASE 3: $S_i > S_j$ and $T_i < T_j$
CASE 4: CASE 1,2,3을 제외한 나머지 경우



(그림 11) 중첩 관계가 가지는 모든 CASE

첫 번째 중첩 관계는 기존 노드의 질의영역(R_i)과 삽입 노드의 질의영역(R_j)이 같은 영역인 것을 의미한다. 먼저 삽입 노드의 원본질의 집합(Q_j)을 기존 노드에 부착시킨다. 삽입 노드는 기존 노드에 포함된 것이나 마찬가지이므로 CQS에서 삽입 노드를 제거하고, 기존 노드가 변경되었기 때문에 DQS에 기존 노드를 삽입하고, 기존 노드의 변경자(M_i)를 체크한다.

두 번째 중첩 관계는 기존 노드의 질의 영역(R_i)이 삽입 노드의 질의 영역(R_j)을 포함할 경우를 의미한다. 이런 경우, 기존 노드는 삽입 노드에 따라 분할되어, 버킷에서 제거된다. 기존 노드는 버킷에서 제거됨에 따라 DQS에 삽입된다. 기존 노드가 있던 자리에 삽입 노드가 삽입되는데, 삽입 노드가 버킷에 포함됨으로 CQS에 있던 삽입 노드는 제거된다. 노드들 사이의 분할은 두 영역 사이에 중첩된 영역을 기반으로 중첩 영역을 포함하는 노드가 분할된다. 분할하는 과정은 필드의 개수만큼 중첩 영역의 필드 축을 기반으로

분할되는데, 각 단계는 분할한 나머지를 기반으로 분할을 수행한다. 각 단계에서 발생하는 분할 노드의 개수는 최대 2개인데, 분할된 영역의 너비나 높이가 0이상이여야 노드로서 존재할 수 있다. 분할 노드는 필드의 개수(n)에 의존적이기 때문에, 최대 2*n개의 분할 노드가 발생할 수 있다. 이에 대한 알고리즘은 (그림 12)와 같다. 이 알고리즘은 CASE 1의 경우를 제외하고 모든 경우에 사용된다. (그림 12)의 분할 알고리즘으로 생성된 분할 노드들은 버킷에 삽입되어야 하

```
QUERYNODE [] fragment(RECTANGLE, RECTANGLE)
    RECTANGLE oldR
    RECTANGLE redR
    QUERYNODE [] fragmentedNode
    RECTANGLE curR := oldR
    integer chValue //changeable value
    For i :=1 to d // d is the dimension
        For j := 1 to 2
            RECTANGLE newR := curR
            changeValue := redR.getValue(j,i)
            If i = 1 Then
                newR.setValue(2,i,chValue)
            Else
                newR.setValue(1,i,chValue)
            If newR.width >0 and newR.height >0 Then
                Create QUERYNODE newNode
                newNode.setRectangle(newR)
                fragmentedNode := fragmentedNode + newNode
        For j := 1 to 2
            curR.setValue(j,i,redR.getValue(j,j))
```

(그림 12) 분할(Fragmentation) 알고리즘

는데, 분할된 노드들이 인접 노드들과 최적의 MBR을 구성하기 위해 현재 버킷에 삽입하지 않고, 상위 버킷에서 적절한 위치에 삽입되도록 한다. 상위 버킷에 삽입하면 버킷의 구성 전략에 따라 여백이 최소화될 수 있는 노드를 선택하여 삽입하기 때문에, 현재 버킷에 삽입하는 것보다 최적의 MBR을 구성할 수 있다. 마지막으로 버킷에 삽입된 노드들이 뒤에 센서 네트워크에 전달될 수 있도록 변경자(M)를 체크한다.

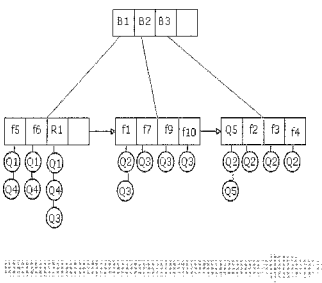
세 번째 중첩 관계는 기존 노드의 질의 영역(R_i)이 삽입 노드의 질의 영역(R_j)에 포함되는 경우를 말한다. 이 경우, 기존 노드는 변경될 필요 없이, 삽입 노드의 원본질의 집합(Q_j)만을 부착시키면 된다. 기존 노드가 변경되었으므로, 기존 노드의 변경자(M_i)를 체크하고 DQS에 기존 노드를 삽입한다. 삽입 노드는 기존 노드를 기준으로 (그림 12)의 분할 알고리즘에 의해 분할된다. 삽입 노드는 변경되었으므로 CQS에서 제거되고, 분할된 질의 노드들은 다른 노드들과의 비교를 위해 CQS에 삽입된다.

네 번째 단계는 기존 노드의 질의 영역(R_i)과 삽입 노드의 질의 영역(R_j)이 서로 일부분만 중첩된 관계를 의미한다. 이 경우, 먼저 중첩되는 영역(R_k)을 기반으로 하는 중첩 노드는 생성한다. 이 단계를 CASE 2와 CASE 3의 복합된 관계로 두 단계에서 수행한 작업이 병합하여 수행한다. 다시 말해서, 중첩 노드는 기존 노드에 대해서 CASE 2의 경우를 의미하며, 중첩 노드는 삽입 노드에 대해서 CASE 3을 의미한다. 4 단계로 구성되는 중첩 관계에 따른 분할 알고리즘은 부록 A와 같다.

4.3 분할된 질의 추출기 (Fragmented Query Extractor)

QR-트리로 구성된 질의 인덱스에서 리프 노드들은 센서 네트워크에 전달할 질의들을 의미한다. 새롭게 삽입된 질의는 리프 노드와의 중첩 관계를 판단하여 새롭게 인덱스에 포함된다. 삽입된 질의는 4.2절의 알고리즘들에 의해 리프 노드에 존재하게 되는데, 이 과정 중에 기존 리프 노드들을 변경하거나 삭제하여 새롭게 센서 네트워크에 전달해야 하는 질의들을 발생시킨다. 그래서 센서 네트워크에 새롭게 삽입될 질의들은 일단 리프 노드들의 변경자가 체크된 노드들만을 고려한다. 리프 노드들은 리프 버킷에 포함되며, 리프 버킷은 정의 2와 같이 이웃 버킷을 레퍼런스하고 있기 때문에, 리프 버킷들을 탐색하면서 센서 네트워크에 삽입되어야 할 리프 노드들을 추출할 수 있다. 질의 인덱스에서 분할된 리프 노드의 개수가 많기 때문에, 분할된 모든 리프 노드들을 질의로써 센서 네트워크에 전달하는 것은 많은 시간과 비용을 필요로 한다. 그래서 되도록 질의의 개수를 줄이기 위해, 질의 재구성 단계에서 OR 연산자에 사용한 방법을 역으로 이용한다. 원본질의 집합(Q)이 같은 리프 노드들은 동일한 질의들에게 결과를 반환하기 때문에, OR 연산자로 병합하여 하나의 질의로 구성할 수 있다. 질의들의 영역은 hyper-rectangle 형태이기 때문에, 분할되는 질의 영역들도 hyper-rectangle로 구성하기 위해 축을 기반으로 분할하였다. 즉, 중첩되지 않는 질의 영역들도 분할되어 많은 질의 분할 노드들을 생성시킨 것이다. 그래서 센서 네트워크에 전달할 질의들은 리프 노드들을 그대로 전달할 필요가 없다.

센서로 전달할 삽입 질의들은 (Q, GQ)의 집합인 해쉬 테이블(IQH)로 구성한다. GQ는 질의 영역의 집합(RS)을 갖는데, 이것은 동일한 원본질의 집합(Q)의 질의 영역(R)을 OR 연산자로 병합하기 위한 것이다. IQH가 해쉬 테이블로 구성된 것은 동일한 원본 질의 집합(Q)을 갖는 GQ를 빨리 찾아, 질의 영역(R)을 질의 영역 집합(RS)에 부착하기 위한 것이다. 이렇게 구성된 IQH는 DQS와 같이 센서네트워크에 전달할 질의들로 변환되어 전달된다. (그림 13)은 분할된 질의를 추출하여 IQH를 구성하는 알고리즘이고 (그림 14)는 그 예를 나타낸 것이다.



Original Queries (5) : (Q1), (Q2), (Q3), (Q4), (Q5)
Fragmented Queries (6) : (f5,f6), (R1), (f1), (f7,f9,f10), (Q5), (f2,f3,f4)

(그림 14) 분할된 질의 추출 예제

```
extractFragmentedQueries(BUCKET)
    BUCKET curBucket
    For i :=1 to curBucket.length
        QUERYNODE curNode := curBucket.Node_i
        If curNode.M Then
            If IQH.containKey(curNode.Q) Then
                GQUERY query := IQH.get(curNode.Q)
                query.RS := query.RS + curNode.R
            Else
                Create GQUERY query
                query.RS := query.RS + curNode.R
                IQH.put(curNode.Q, query)
    extractFragmentedQueries(curBucket.nextBucket)
```

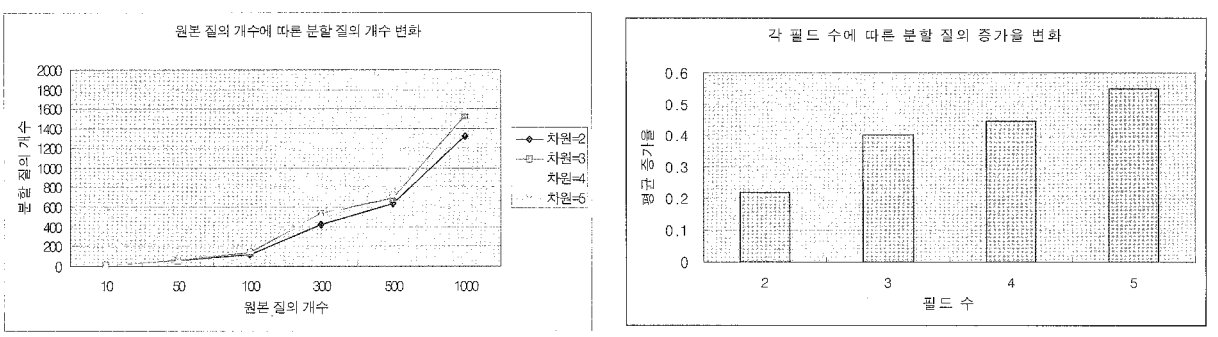
(그림 13) 분할된 질의 추출 알고리즘

5. 실험 및 결과

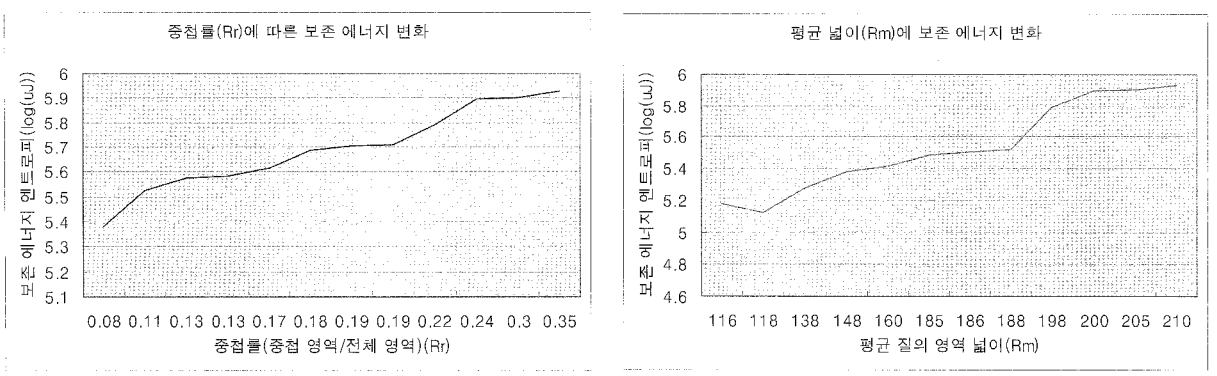
실험은 일단 QR-트리로 획득된 분할 질의의 개수를 평가하고, 원본 질의 개수에 따른 센서 네트워크 에너지 보존율을 측정한다. 실험을 위해 사용된 서버 컴퓨팅 환경은 2.00GHz, 512MB RAM, Windows XP 이고, 센서 플랫폼은 Mica 플랫폼[19-21]이며, 에너지량을 측정하기 위해 PowerTOSSIM[23]의 시뮬레이터를 이용하였다. 첫 번째 실험은 Sensors 테이블의 필드 수에 따라 분할되는 질의 개수를 평가한다. (그림 15)와 같이 분할 질의는 원본 질의 개수가 증가하거나, 필드 수가 증가함에 따라 분할되는 질의 수는 많은 것을 알 수 있다. 실험을 통해 분할되는 질의의 평균적인 증가율은 30%로 측정되었다. 즉, 100개의 질의에 대해서는 130개 정도의 분할된 질의가 발생한다는 것을 알 수 있었다. (그림 15)의 오른쪽 그림은 필드 수에 따라 분할되는 질의 개수가 원본 질의 개수에 대해 증가한 비율을 보여준다. 이것은 필드 수가 많아짐에 따라 분할되는 질의의 개수도 많다는 것을 보여주지만, 증가율의 폭은 줄어드는 것도 보여준다. 이러한 이유는 필드 수가 증가함에 따라 분할되어야 하는 축도 많아지지만, 중첩되는 비율도 상대적으로 감소하기 때문이다.

두 번째 실험은 본 논문에서 제안한 분할 알고리즘으로 얻어지는 센서 네트워크의 보존율을 측정하는 것이다. 이 시뮬레이션에서 사용된 환경 센서는 습도, 온도, 자기장, 마이크로폰 센서이다. 보존 에너지 비율의 정확한 측정을 위해 전체 데이터의 분포는 4차원에 균등하게 분포하도록 구성하였다. 왜냐하면 특정 영역에 중첩이 많고, 센싱한 데이터가 그 영역에만 집중되는 현상에서는 원본 질의에 대해 센서 네트워크에 에너지가 어느 정도 보존되는지 알 수 없기 때문이다. 센서 노드의 개수는 100개, 반환되는 데이터의 인터벌(Epoch)은 1초, 전체 처리시간은 1600초 동안 질의 처리를 수행하였다. 센서 노드들의 깊이(depth)는 10의 깊이 범위를 랜덤하게 선택하여 결정하였다.

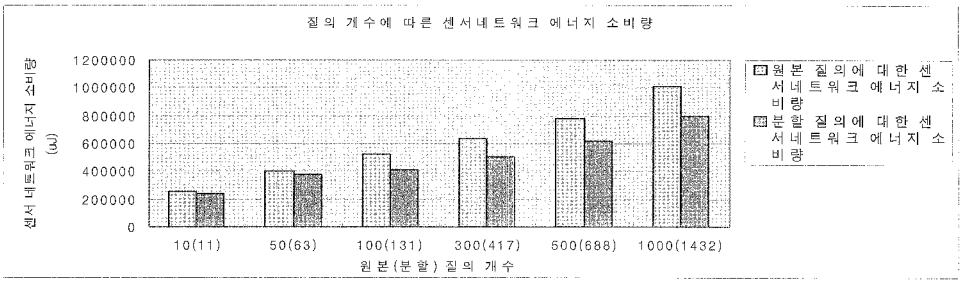
본 실험은 원본 질의의 개수에 따라 어느 정도의 에너지가 보존되는지 측정하고자 한다. 하지만 질의들을 랜덤하게 생성하였기 때문에, 생성된 질의들이 동일한 기준이 아니다. 즉, 에너지가 어느 정도 보존되는지 정확히 측정할 수 없다. 그래서 본 논문은 보존 에너지와 선형 관계에 있을 만한 요소들(평균, 분산, 중첩률 등)을 분석하여 선형 관계에 있는 두 가지 요소를 실험을 통해 추출하였다. 각 요소들은 실험의 정확성을 판단하기 위해 독립적으로 분석되었다. 하나는



(그림 15) 원본 질의 개수에 따른 분할 질의 개수 변화



(그림 16) 보존 에너지에 영향을 끼치는 두 가지 요소(Rm, Rr) 분석



(그림 17) 원본(분할) 질의 개수에 따른 센서 네트워크 에너지 소비량

중첩률(Rr)인데, 전체 질의 영역에서 중첩되는 영역이 차지하는 비율을 의미한다. 다른 하나는 중첩된 영역의 평균 넓이(Rm)로써, 중첩된 영역들의 빈도수(Q.size-1)와 넓이를 기반으로 계산된다. 중첩률과 평균 넓이는 크기가 커지면 전체 영역에 대해 중첩되는 영역의 넓이가 커져, 중첩 결과가 반환할 수 있는 확률이 높아진다. (그림 16)의 실험은 1000개의 원본 질의에 대한 중첩률과 평균 넓이를 변화하여 실험한 보존 에너지 변화 그래프이다. 보존 에너지의 단위는 uJ인데, 수치가 너무 커서 로그를 취하였다.

앞의 실험을 기반으로 두 요소(Rm, Rr)를 일정하게 유지하면서, 원본 질의 개수 변화에 따라 센서 네트워크의 에너지 소비량 측정하였다. 실험은 10, 50, 100, 300, 500, 1000으

부록 A. 중첩 관계에 따른 분할 알고리즘

```
fragmentLeafNodes(Bucket, QUERYNODE, QUERYNODE)
    QUERYNODE oldNode
    QUERYNODE newNode
    Bucket curBucket
    If oldNode.R = newNode.R Then // CASE 1
        oldNode.refkeys := oldNode.refkeys + newNode.refkeys
        oldNode.modifier := TRUE
        CQS := CQS - newNode
        DQS := DQS + oldNode
    Else If oldNode.R ⊃ newNode.R Then //CASE 2
        curBucket := curBucket - oldNode
        DQS := DQS + oldNode
        CQS := CQS - newNode
        newNode.modifier = TRUE
        QueryNode [] fragNodes = fragment(oldNode,newNode)
        For i := 1 to fragNodes.length
            If curBucket.ParentBucket != NULL Then
                insertCleanNode(curBucket.ParentBucket, fragNodes.Node_i)
            Else insertEntry(curBucket, fragNodes.Node_i)
            Set modifier to be the TRUE in fragNode.Node_i
    Else If oldNode.R ∩ newNode.R ≠ ∅ Then //CASE 3
        QUERYNODE redNode := getRedundanctNode(oldNode, newNode)
        redNode.modifier := TRUE
        curBucket := curBucket - oldNode
        DQS := DQS + oldNode
        curBucket.insertEntry(redNode)
        QueryNode [] fragNodes := fragment(oldNode, redNode)
        For i := 1 to fragNodes.length
            If curBucket.ParentBucket != NULL Then
                insertCleanNode(curBucket.ParentBucket, fragNodes.Node_i)
            Else insertEntry(curBucket, fragNodes.Node_i)
            fragNode.Node_i.modifier := TRUE
        QueryNode [] fragNodes := fragment(newNode, oldNode)
        CQS := CQS - newNode
        For i := 1 to fragNodes.length
            CQS := CQS + fragNodes.Node_i
    Else If oldNode.R ⊂ newNode.R Then //CASE 4
        oldNode.modifier := TRUE
        DQS := DQS + oldNode
        oldNode.addRefKeys(newNode.refkeys)
        CQS := CQS - newNode
        QueryNode [] fragNodes := fragment(newNode, oldNode)
        For i := 1 to fragNodes.length
            Add fragNodes.Node_i to CQS
```

로 원본 질의의 개수를 변화시키면서 진행되었다. (그림 17)은 이 실험에 대한 결과로써, 약 20%의 에너지가 보존되는 것을 알 수 있었다. 만일 Rm, Rr의 요소가 증가한다면, 이에 따라 보존되는 에너지 비율도 증가할 것이다.

6. 결론 및 향후 과제

우리는 센서 네트워크의 에너지를 오랜 시간동안 보전하기 위해, 센서에 전송되는 연속질의의 중복성을 제거하였다. 센서 네트워크에서 발생하는 데이터는 베이스스테이션에서 전달된 연속질의를 기반으로 하고 있다. 그래서 연속질의사이에 중복된 영역이 발생한다면 센서들은 중복된 결과를 반환할 수 있다. 즉, 센서들은 중복된 데이터를 반환함으로써 불필요한 에너지를 소모하게 된다. 이러한 문제점을 해결하고자 중복된 데이터의 원천적인 문제가 되는 다중 연속질의들 사이의 중복성을 해결하고자 하였다.

다중 연속질의들 사이의 중복성은 WHERE절의 중첩된 조건식을 차원 축을 기반으로 분할하는 알고리즘을 통해 간단히 해결할 수 있었다. 하지만 차원이 증가하거나 베이스스테이션에 존재하는 질의가 증가함에 따라 분할에 필요한 처리 시간이나 분할된 질의 개수가 많아지는 문제를 초래하였다. 그래서 본 연구는 후자보다는 전자의 문제점이 먼저 해결되어야 한다고 판단되어, R*-트리 기반의 QR-트리를 제안하였다. QR-트리는 R*-트리의 정책을 받아들여, 버킷의 여백을 최소화하도록 구성하여 빠르게 중첩된 질의를 발견할 수 있도록 하였다. 또한, 탐색시간을 줄이기 위해, CQS라는 스택을 두어 분할된 질의들도 다시 QR-트리를 탐색하지 않도록 하였다.

하지만 본 연구에서 구성한 분할 알고리즘은 아직 분할 질의 개수가 많은 문제점을 갖고 있다. 현재까지의 연구는 질의에 대한 결과는 고려하지 않았다. 사실 반환된 질의의 결과가 중복이 되어야 실제로 질의가 중첩되었다고 말할 수 있기 때문에, 분할 질의를 실제 질의 결과에 반영해야 한다. 만일 질의 결과가 반환되지 않는 질의들은 센서 네트워크의 에너지에 어떤 영향도 끼치지 않기 때문에, 분할할 필요가 없다. 즉, 분할되는 질의 개수를 줄일 수 있다. 그래서 향후 반환된 데이터의 시놉시스를 기반으로 분할 질의 개수를 줄일 수 있는 방법을 연구하고자 한다.

참 고 문 헌

[1] B. Babcock, S. Babu, M. Datar, R. Motwani, J. Widom. Models and Issues in Data Streams. In Proc. ACM Symp. on Principles of Database Systems, pp.1-16, 2002.

[2] P. Bonnet, J. Gehrke, P. Seshadri. Towards Sensor Database Systems. In Proc. Int. Conf. on Mobile Data Management, pp.3-14, 2001.

[3] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. Madden, V. Raman, F. Reiss, M. Shah. TelegraphCQ: Continuous Data ow Processing for an Uncertain World. In Proc. Conf. on Innovative Data Syst. Res, pp.269-280, 2003.

[4] J. Chen, D. DeWitt, F. Tian, Y. Wang. NiagaraCQ: A Scalable Continuous Query System for Internet Databases. In Proc. ACM Int. Conf. on Management of Data, pp.379-390, 2000.

[5] S. Madden, M. J. Franklin, J. M. Hellerstein, W. Hong. TAG: a Tiny AGgregation Service for Ad-Hoc Sensor Networks. In Proc. Symp. on Operating Systems Design and Implementation, 2002.

[6] S. Madden, M. Shah, J. Hellerstein, V. Raman. Continuously Adaptive Continuous Queries Over Streams. In Proc. ACM Int. Conf. on Management of Data, pp.49-60, 2002.

[7] D. Estrin, L. Girod, G. Pottie, and M. Srivastava. Instrumenting the world with wireless sensor networks. In International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2001), Salt Lake City, Utah, May, 2001.

[8] R. Min, M. Bhardwaj, S.-H. Cho, A. Sinha, E. Shih, A.Wang, and A. Chandrakasan. An architecture for a power-aware distributed microsensor node. In IEEE Workshop on Signal Processing Systems (SiPS '00), October, 2000.

[9] P. Saffo. Sensors: The next wave of innovation. Communications of The ACM, 40(2):92 - 97, February, 1997.

[10] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. Computer Networks, 38:393-422, 2002.

[11] M. A. Sharaf, J. Beaver, A. Labrinidis and P. K. Chrysanthis, TiNA: A Scheme for Temporal Coherency-Aware in-Network Aggregation. in Proceedings of the 3rd ACM MobiDE Workshop. September, 2003.

[12] R. Motwani, J. Widom, A. Arasu, B. Babcock, S. Babu, M. Datar, G. Manku, C. Olston, J. Rosenstein, and R. Varma. Query Processing, Resource Management, and Approximation in a Data Stream Management System. In Proc. of the 2003 Conf. on Innovative Data Systems Research (CIDR), January, 2003.

[13] D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, G. Seidman, M. Stonebraker, N. Tatbul, and S. Zdonik. Monitoring Streams: A New Class of Data Management Applications. In proceedings of the 28th International Conference on Very Large Data Bases (VLDB'02), Hong Kong, China, August, 2002(Aurora).

[14] D. Terry, D. Goldberg, D. Nichols, and B. Oki. Continuous queries over append-only databases. In Proc. of the 1992 ACM SIGMOD Intl. Conf. on Management of Data, pages 321-330, June, 1992.

[15] L. Liu, C. Pu, and W. Tang. Continual queries for internet scale event-driven information delivery. IEEE Trans. on Knowledge and Data Engineering, 11(4):583 - 590, Aug., 1999.

[16] TIMOS K. SELLIS. Multiple-Query Optimization. Transaction on Database System of the ACM, Vol.13, No.1, pp.23-52, March, 1988,

[17] Yousuke Wantanabe. Hiroyuki Kitagawa. A Multiple Continuous Query Optimization Method Based on Query Execution Pattern Analysis. DASFAA 2004, LNCS 2973, Pages 443-456.
[18] A. Mainwaring, J. Polastre, R. Szewczyk, and D. Culler. Wireless sensor networks for habitat monitoring. In ACM Workshop on Sensor Networks and Applications, 2002.
[19] J. Hill, R. Szewczyk, A.Woo, S. Hollar, and D. C. K. Pister. System architecture directions for networked sensors. In ASPLOS, November, 2000.
[20] Samuel R. Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. The Design of an Acquisitional Query Processor for Sensor Networks, Proceedings of the 2003 ACM SIGMOD international conference on Management of sdata, June, 2003.
[21] TinyOS : www.tinyos.net
[22] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger, "The R*-tree: An efficient and robust access method for points and rectangles," Proceedings of ACM SIGMOD Int'l. Conf. on Management of Data, pp.322-331, 1990.
[23] V. Shnayder, M. Hempstead, B. Chen, G. W. Allen, and M. Welsh, "Simulating the Power Consumption of LargeScale Sensor Network Applications," SenSys'04, November 3-5, 2004, Baltimore, Maryland, USA.

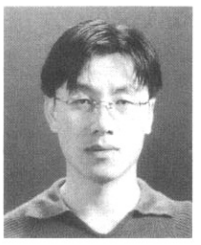
박 정 업

e-mail : jupark@cse.hanyang.ac.kr
2005년 한양대학교 전자컴퓨터공학부 학사
2005년~현재 한양대학교 컴퓨터공학과 석사과정
관심분야 : e-비즈니스, 그리드 컴퓨팅, 센서 네트워크

조 명 현

e-mail : mhjo@cse.hanyang.ac.kr
2004년 한양대학교 전자컴퓨터공학부 학사
2006년 한양대학교 컴퓨터공학과 석사
2006년~현재 코난테크놀로지 근무
관심분야 : 그리드 컴퓨팅, 센서 네트워크, 시멘틱 웹, e-비즈니스

김 학 수



e-mail : hagsoo@cse.hanyang.ac.kr
2004년 한양대학교 전자컴퓨터공학부 학사
2006년 한양대학교 컴퓨터공학과 석사
2006년~현재 한양대학교 컴퓨터공학과 박사과정
관심분야 : 그리드 컴퓨팅, XQuery, 시멘틱 웹, XML, 데이터베이스

이 동 호

e-mail : dhlee72@cse.hanyang.ac.kr
1995년 홍익대학교 컴퓨터공학과 학사
1997년 서울대학교 컴퓨터공학과 석사
2001년 서울대학교 전기컴퓨터공학부 박사
2001년 3월~2004년 2월 삼성전자 책임연구원
2004년 3월~현재 한양대학교 컴퓨터공학과 조교수
관심분야 : 멀티미디어 데이터베이스, 멀티미디어 정보검색, 고차원색인구조, 내장형 데이터베이스, Flash Memory용 시스템소프트웨어

손 진 현

e-mail : jhson@cse.hanyang.ac.kr
1996년 서강대학교 전산학과 학사
1998년 한국과학기술원 전산학과 석사
2001년 한국과학기술원 전자전산학과 박사
2001년 9월~2002년 8월 한국과학기술원 전자전산학과 박사후 연구원
2002년 9월~현재 한양대학교 컴퓨터 공학과 조교수
관심분야 : 데이터베이스, e-비즈니스, 유비쿼터스 컴퓨팅, 임베디드 시스템