센서 네트워크에서 집계연산을 위한 적응적 필터링
(Adaptive Filtering for Aggregation in Sensor Networks)

박 노 준 [†] 현 동 준 [††] 김 명 호 [†††]
(No Joon Park) (Dongjoon Hyun) (Myoung Ho Kim)

요 약 각 센서들이 측정한 데이타의 평균값 등을 구하는 집계연산은 센서 네트워크에서 자주 사용되는 응용이다. 센서 네트워크를 구성하는 센서는 작은 배터리로 작동되기 때문에 센서의 에너지 소모를 줄이는 것은 센서 네트워크의 중요한 문제이다. 센서의 에너지 소모를 줄이기 위한 가장 중요한 요소는 전송되는 메시지 수를 줄이는 것인데, 네트워크 내 집계연산과 데이타 필터링이 집계연산 시 전송되는 메시지 수를 줄이기 위한 효과적인 방법이라고 알려져 있다. 네트워크 내 집계연산과 데이타 필터링을 동시에 수행하면 더 많은 메시지를 줄일 수 있으며, 최근에 이 방법을 근간으로 한 연구가 있었다. 본 논문에서는, 기존의 데이타 필터링 방법보다 더 효율적이고 효과적인 방법을 제안한다. 본 논문에서 제안하는 방법은 센서 노드의 자기 조정에 기반하고 있기 때문에 더 쉽고 간단하다. 다양한 실험을 통해서, 본 논문에서 제안하는 방법이 다른 방법들보다 더 많은 메시지를 줄이는 것을 확인하였다.
키워드 : 센서 네트워크, 집계연산, 적응적 필터링

Abstract Aggregation such as computing an average value of data measured in each sensor commonly occurs in many applications of sensor networks. Since sensor networks consist of low-cost nodes with limited battery power, reducing energy consumption must be considered in order to achieve a long network lifetime. Reducing the amount of messages exchanged is the most important for saving energy. Earlier work has demonstrated the effectiveness of in-network data aggregation and data filtering for minimizing the amount of messages in sensor networks. In this paper, we propose an adaptive error adjustment scheme that is simpler, more effective and efficient than previous work. The proposed scheme is based on self-adjustment in each sensor node. We show through various experiments that our scheme reduces the network traffic significantly, and performs better than existing methods.
Key words : Sensor Networks, Aggregation, Adaptive Filtering

1. 서 론

센서 네트워크(sensor network)는 배터리로 작동되는 작은 무선 센서들로 구성 된다. 이 센서들은(e.g. Berkeley Motes[1], MIT μamps nodes[2]) 서로 협력하면서 주위 환경으로부터 여러 정보들을 수집해서 모니터링 응용(monitoring applications[3])에게 필요한 데이타를 제공하는데, 이 데이타에는 온도나 습도와 같은 기상 데이타도 있을 수 있고, 소음이나 진동의 정도, 그리고 사물의 움직임 여부와 같은 데이타도 있을 수 있다.

사용자가 기지국(base station)에 원하는 질의를 입력하면, 기지국은 그 질의를 모든 센서에게 전송을 해 준다. 그리고, 각 센서는 질의를 받은 후 그 질의에 필요한 데이타를 읽어서 기지국에 보내준다. 이때 센서와 기지국은 멀리 떨어져 있기 때문에, 데이타는 센서에서 기지국까지 한번에 전송되지 못하고, 여러 센서를 거쳐 전송된다. 센서에서 기지국까지 데이타가 전송되는 경로를 그리면, 센서 네트워크를 하나의 커다란 트리 구조로 볼 수 있다.

센서 네트워크를 설치하고 사용하는 데 있어서 가장 고려해야 할 문제 중 하나는, 센서가 소용량 배터리로 작동된다는 것이다. 센서가 설치되는 곳은, 사람에게 위험한 곳(예를 들어, 방사능 노출 지역)이거나 험한 정글과 같이 일일이 센서의 위치를 찾기가 힘든 곳일 수 있

· 본 연구는 한국과학재단 특정기초연구(과제번호 R01-2003-000-10627-0) 지원으로 수행되었음
† 비 회 원 : 삼성전자 기술총괄 시스템연구소
njoon.park@samsung.com
†† 비 회 원 : 한국과학기술원 전자전산학과
djhyun@dbserver.kaist.ac.kr
††† 종신회원 : 한국과학기술원 전자전산학과 교수
mhkim@dbserver.kaist.ac.kr
논문접수 : 2005년 1월 20일
심사완료 : 2005년 5월 16일

다. 그러므로 센서의 배터리가 다 소진되면, 그 배터리를 교체해 주는 것은 매우 어렵거나, 때로는 아예 불가능 할 수도 있다. 따라서 센서 네트워크 관련 분야에서는, 각 센서의 전력 소모를 최소화하여 센서 네트워크의 유효기간(lifetime)을 증가시키는 것이 주요 연구 주제가 되어 왔다.

어떤 센서 네트워크의 유효기간이 끝났는지의 여부를 결정하는 것은 여러 가지 기준으로 정할 수 있다. 예를 들어, 전체 센서 네트워크 중에서 어떤 센서 하나라도 동작하지 않으면 그 센서 네트워크의 유효기간이 끝났다고 여길 수도 있다. 하지만, 전체 센서들 중 일정 부분만큼(예를 들어, 전체 센서 중 10%)의 센서가 동작하지 않을 때, 그 센서 네트워크의 유효기간이 끝난 것이라고 보는 것이 일반적이다. 따라서, 센서 네트워크의 유효기간을 증가시키기 위해서는, 특정 센서들만 배터리가 먼저 소진되는 것을 방지해 주어야 한다. 즉, 각 센서의 배터리 소모가 어느 정도 균형을 이루도록 만들어, 다른 센서에 비해 너무 일찍 배터리를 소진하는 센서의 수를 적게 해 줌으로써, 센서 네트워크의 유효기간을 증가시킬 수 있다.

센서의 배터리 소모를 줄여주기 위해서는 센서의 데이타 전송 횟수를 줄여 주는 것이 가장 효과적이다. 센서의 내부 처리(local processing)를 줄여 주거나 전송할 데이타의 양을 줄여주는 것도 에너지 소모를 줄이는 중요 요인이지만, 무선 데이타 수신과 전송에 요구되는 에너지 소비가 매우 크기 때문에[4,5], 데이타 전송 횟수를 최대한 줄여주는 것이 센서의 배터리 소모를 줄이기 위한 가장 좋은 방법으로 연구되고 있다[2,5-8,10,11].

지금까지 센서 네트워크의 유효기간을 증가시키려는 연구들은 크게 세 가지 형태로 구분될 수 있다. 첫째, 에너지 인지 프로토콜(energy-aware protocol)을 사용하는 방법이다[5,11]. 사용자가 입력한 질의들(queries)을 대상으로 최적화(optimization) 과정을 거칠 때, 각 센서의 에너지 보유 정도를 기반으로 최적화 시킨다. 즉, 에너지가 적은 센서의 작동(operations)을 줄이고 에너지가 많은 센서의 작동을 늘려 주어, 전체적으로 모든 센서의 에너지 소모 정도가 균형을 이루도록 해 주는 방법이다. 둘째로, 네트워크 내 집계연산(in-network aggregation)을 사용해서 센서들이 전송하는 데이타의 양을 줄여주는 방법이 있다[5,7]. 네트워크 내 집계연산에 대해서는 2.1절에서 설명된다.

위에서 언급한 두 가지 방법은, 사용자에게 정확한 결과 값을 제공해 주면서 전송되는 데이타의 양이나 횟수를 줄여주는 방법이다. 하지만 센서 네트워크에서는, 질의 결과에 어느 정도의 오차가 있어도 문제가 되지 않는 경우가 많다[8-10]. 즉, 질의 결과의 정확도를 약간 희생하면서 데이타 전송 횟수를 줄일 수가 있는데, 이 방법이 필터링(filtering)이다[8,9]. 필터링에 대한 자세한 내용은 2.2절에서 설명된다.

본 논문에서는, 센서 네트워크의 유효기간을 늘려주기 위해 각 센서에서의 데이타 전송을 줄이기 위한 방법을 제안한다. 본 논문에서는 제안하는 방법은, 기본적으로 네트워크 내 집계와 필터링을 결합한 것이다. 즉, 네트워크 내 집계를 하면서 동시에 필터링을 수행함으로써, 모든 센서의 데이타 전송을 보다 효과적으로 줄이고자 한다. 센서 네트워크로부터 필요한 데이타를 수집할 때 네트워크 내 집계와 필터링을 동시에 사용하게 되면, 각 센서의 데이타 전송 억제를 극대화 할 수 있다. 센서 네트워크에서 네트워크 내 집계와 필터링을 동시에 수행하면서 데이타를 수집하는 과정은 2.3절에서 설명된다.

센서 네트워크를 이루고 있는 센서들이 주기적으로 데이타를 측정할 때, 그 측정되는 데이타의 특성은 센서마다 다르다. 어떤 센서가 측정하는 데이타는 계속해서 변화가 없어서 거의 같은 값일 수도 있고, 또 어떤 센서가 측정하는 데이타는 값이 큰 폭으로 자주 변할 수 있다. 본 논문에서는 데이타 값의 변화 정도가 미미한 센서 노드를 *안정 노드(stable node)*, 그리고 데이타 값의 변화 정도가 큰 센서 노드를 *변동 노드(volatile node)*라고 부를 것이다. 물론 데이타 값의 변화는 시시각각 다를 수 있기 때문에, 각 노드는 시간이 지나면서 안정 노드가 될 수도 있고 변동 노드가 될 수도 있다. 그러므로 어떤 노드가 안정 노드, 또는 변동 노드라는 것은 일정 시간 동안에만 국한해서 말하는 것이다.

센서에서 데이타 필터링을 하기 위해서는 각 센서에 에러 범위가 할당되어야 하는데, 그 에러 범위를 적응적으로 조정해 주면 데이타 전송을 보다 효과적으로 감소시킬 수 있다. 안정 노드는 데이타 값의 변화가 거의 없기 때문에, 필터링을 위한 에러 범위를 크게 할당할 필요가 없다. 반면에 변동 노드는 데이타 값의 변화가 크기 때문에, 보다 큰 에러 범위를 할당해 주면 필터링을 더 효과적으로 할 수 있다. 따라서 안정 노드에 할당되어 있는 에러를 줄여주고, 그렇게 줄여서 남게 되는 에러를 변동 노드에게 더해줄 수 있다면, 전체적으로 데이타 필터링을 보다 효과적으로 할 수 있다. 이것이 필터링을 하면서 각 데이타 소스의 에러 범위를 조정해야 하는 이유다. 본 논문에서는 이 에러 범위를 어떻게 하면 차별적으로 분산시킬 수 있을지에 대해 기존 연구보다 효과적인 방법을 제안하고자 한다. 에러 범위의 적응적 조정을 위해 본 논문에서는 ARA(Adaptive eRror-bound Adjustment) 프로토콜을 제안한다. 각 센서는 데이타를 측정하는 매 순간마다 ARA 프로토콜을 이용해서 자신의 에러 범위를 스스로 조정하게 된다.

논문의 구성은 다음과 같다. 2장에서는 ARA 프로토콜을 설명하기 전에 알아야 할 개념에 대해 설명하고, 3장에서 관련 연구에 대한 소개를 한다. 4장에서는 본 논문이 제시하는 ARA 프로토콜에 대해 설명하고, 5장에서 이 프로토콜의 우수성을 실험으로 증명해 보인다. 마지막 6장에서는 본 논문의 결론을 내리고 향후 연구 방향을 제시한다.

2. 연구 배경

센서 네트워크에 대한 집계 질의를 처리할 때, 각 센서 노드에서 필터링을 수행하면서 질의 처리를 네트워크 내부에서 수행하면, 센서 네트워크의 유효기간을 크게 증가시킬 수 있다. 이번 장에서는 ARA 프로토콜을 설명하기 전에 알아야 할 개념에 대해 설명하겠다.

2.1 네트워크 내 집계연산

센서 데이타를 집계하기 위한 방법으로 쉽게 생각할 수 있는 것은, 일단 센서가 읽은 모든 데이타를 기지국으로 보내고, 그 기지국에서 모든 집계 처리를 수행하는 것이다. 하지만 이 중앙 처리 방식(centralized, server-based approach)은 매우 많은 메시지 전송이 필요하고 처리 시간도 오래 걸린다. 그렇기 때문에 제안된 방식이 네트워크 내 집계(in-network aggregation) 방식이다 [5,7]. 네트워크 내 집계를 사용하면 중앙 처리 방식보다 메시지 전송이나 지연(latency), 그리고 센서의 배터리 소모를 크게 줄일 수 있다.

그림 1에서처럼, 센서 네트워크에서 네트워크 내 집계 연산을 수행하기 위해서는, 먼저 센서 노드를 이용해서 집계 트리를 구성한다. 집계 트리는 각 센서 노드가 기지국으로 데이타를 전송하는 경로를 트리로 표현한 것이다. 센서는 다른 센서들과 클락(clock)을 동기화 시킬 수 있다. 그래서 데이타를 읽은 후에 바로 부모 노드에게 전송하지 않고, 모든 자식 노드들이 데이타를 보낼 때까지 정해진 시간만큼 기다린다. 모든 자식 노드들로부터 데이타를 다 받으면, 그 노드는 부분 집계 결과(partial aggregate)를 계산해서, 그 부분 집계 결과를 부모 노드에게 전송한다. 집계 트리의 단말 노드부터 루트 노드까지 모든 노드가 이 같은 과정을 거치고 나면, 루트 노드는 구해야 할 최종 결과를 갖게 된다. 주어진 집계 질의에 대한 최종 결과를 루트 노드가 기지국에 보고함으로써 집계 처리가 끝나게 된다. 집계 트리를 구성하는 방법과 네트워크 내 집계 과정은 [5]와 [7]에서 살펴볼 수 있다.

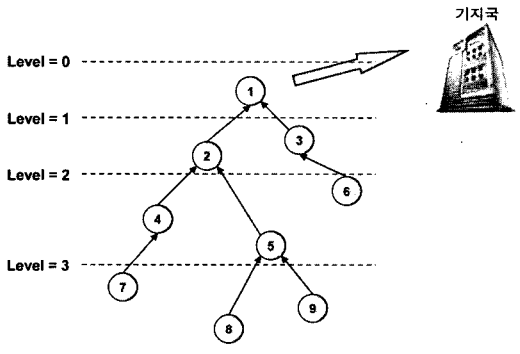

그림 1 센서 네트워크에서의 네트워크 내 집계연산

2.2 필터링

센서 데이타에 기반하는 많은 응용들은, 근사 결과(approximate answer)를 어느 정도의 부정확도 안에서 허용할 수 있다. 다시 말해, 센서 데이타는 그 값에 어느 정도의 오차가 있어도 큰 문제가 되지 않는 경우가 많다는 것이다. 이것은 센서 데이타의 특성에서 기인하는데, 센서가 측정하는 온도나 습도, 진동, 소음, 그리고 밝기 등과 같은 센서 데이타는, 그 값 자체가 100% 정확하기가 쉽지 않다. 또 때로는 굳이 그 값이 100% 정확할 필요가 없을 수도 있다. 예를 들어, 주변의 온도를 측정한 온도 데이타가 12.3°C인 것과 12.4°C인 것은 크게 차이가 없는데, 실제 온도가 12.3°C일 때 온도 데이타 값이 12.4°C라고 해서 크게 문제가 되지는 않는다. 센서 데이타를 사용하는 많은 응용들이 이러한 특성을 가지고 있다.

이와 같은 특징을 이용해서 센서 노드에서의 메시지 전송을 줄일 수가 있는데, 이것이 데이타 필터링이다. 어떤 센서가 새롭게 측정한 데이타 값과 이전에 측정해서 보냈던 데이타 값과의 차이가 미리 정한 에러 범위 안에 있을 경우, 새 데이타를 전송하지 않는 것이 필터링이다. 어떤 자식 노드가 새 데이타를 보내지 않으면, 부모 노드는 그 자식 노드가 이전에 보냈던 데이타를 캐싱해 놓고 있다가, 그 캐싱되어 있던 이전 데이타를 사용해서 필요한 처리를 수행한다. 모든 센서 노드에서 이와 같은 필터링을 수행하면, 전송되어야 할 메시지 수가 크게 줄어든다.

그림 2에서 센서 노드 N_i에서의 데이타 필터링을 좀 더 자세히 보여주고 있다. 예로 주어진 질의를 보면 WITHIN 절이 있다. 사용자는 이 WITHIN 절을 이용해서, 주어진 질의에 대한 허용 가능한 에러 범위를 명시할 수 있다. 이 예에서는 전체 에러 범위를 1.2로 설정했다. 집계 질의에서 사용자가 명시한 전체 에러 범위를 기반으로, 각 센서 노드에게 개별 에러 범위(ne_i)를 할당한다. 이 예에서는 각 노드에게 ne_i를 0.3씩 균등하게 할당했다. 또한 모든 센서 노드 N_i와 그 부모 노드는, N_i가 이전에 전송했던 데이타 값(D_i)을 유지하고 있다. 이 값을 N_i의 *기본값(default value)*이라고 한다. N_i

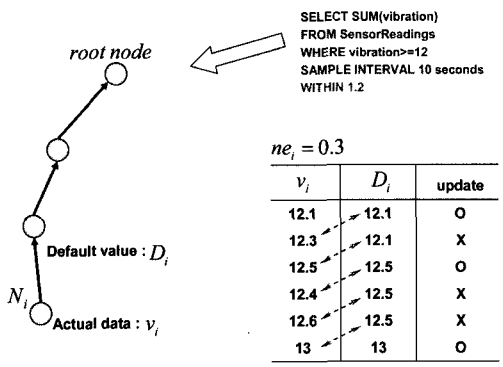

그림 2 센서 노드에서의 데이타 필터링

가 처음 읽은 값이 12.1이었을 때, 제일 처음이므로 데이타를 부모 노드에게 전송하고 D_i를 12.1로 설정한다. 두 번째 읽은 값이 12.3이라면, 이 값과 D_i와의 차이가 0.2 이므로 에러 범위 안에 있다. 그래서 N_i는 새 데이타 12.3을 전송하지 않고, N_i의 부모 노드는 D_i(=12.1)를 이용해서 필요한 처리를 수행한다. 즉, N_i의 두 번째 값도 12.1이 되는 것이다. N_i가 세 번째 읽은 값은 12.5다. 12.5는 D_i와의 차이가 0.4이므로 에러 범위를 초과한다. 그러므로 N_i는 세 번째 읽은 값을 부모 노드에게 전송하고, D_i를 12.5로 변경한다. 부모 노드에서는 N_i로부터 새로운 데이타를 받았으므로, N_i에 대해 캐싱되어 있는 D_i를 12.5로 바꾸고, 새로 전송 받은 데이타를 이용해서 필요한 처리를 수행한다.

2.3 필터링과 네트워크 내 집계연산

앞 절에서 설명한 네트워크 내 집계연산과 필터링을 동시에 사용하면, 센서 네트워크에서의 데이타 전송을 더욱 크게 감소시킬 수 있다. 이 절에서는 네트워크 내 집계연산을 수행하면서 동시에 필터링 하는 과정을 하나의 예를 가지고 설명하겠다.

그림 3에 5개의 센서 노드로 구성된 센서 네트워크가 있다. 우리는 이 센서 네트워크에서 수집한 데이타를 SUM하려고 하는데, 네트워크 내 집단화와 필터링을 동시에 이용하려고 한다. 이 예에서는 모든 노드의 ne_i를 3으로 설정했다.

어떤 시점에서 각 센서 노드는 기본값(D_i)을 가지게 된다. N_i의 기본값은 N_i와 N_i의 부모 노드가 동시에 유지하는데, 그림에서는 D_i를 N_i와 그 부모 노드와의 에지(edge)에 표현해 놓았다. 각 센서 노드 옆에 씌어 있는 수는 방금 각 센서가 읽은 데이타 값을 나타낸다. 이제, 각 센서 노드는 측정한 값을 부모 노드에게 전송해야 하는데, 레벨이 제일 큰 노드부터 순차적으로 전송한다. 이때, 무조건 전송하지 않고, 각 노드의 ne_i를 이용하여 필터링을 한다. 다음으로, 레벨이 2인 2번 노드와 3번 노드를 보겠다. 먼저 2번 노드를 보면, 2번 노드가 알고 있는 4번 노드의 값은, 4번 노드가 데이타를 전송하지 않았기 때문에 기본값인 113이 되고, 2번 노드가 알고 있는 5번 노드의 값은, 5번 노드가 전송한 104가 된다. 이제 2번 노드는 자신이 새로 읽은 값 113과 4번 노드와 5번 노드의 값을 합하여, 부분 집계 결과 A_2가 330(113+104+113)이라고 계산한다. 2번 노드는 A_2를 이용해 필터링 여부를 결정한다. 2번 노드의 기본값은 337이다. 여기서 우리는 한 노드의 에러 범위를 나타내는 ne 이외에, 어떤 노드 N_i를 루트로 하는 부-트리(sub-tree)의 에러 범위인 te라는 개념이 필요하다. te_2는 2번 노드를 루트로 하는 부-트리의 ne를 모두 합한 것인데, 그림에서는 te_2가 9이다. 4번 노드나 5번 노드와 같은 단말 노드는 ne와 te가 같다.

te_2는 9, A_2는 330, 그리고 D_2가 337일 때, 2번 노드에서의 필터링 과정을 보도록 하겠다. 언뜻 보기에는, A_2와 D_2의 차이가 7이고 te_2가 9이므로 데이타를 전송하지 않아도 되는 것처럼 보이지만, 비 단말 노드(non-leaf node)에서는 추가적으로 고려해야 하는 것이 있다. 2번 노드에서는 4번 노드가 데이타를 전송하지 않아 4번 노드의 기본값 113을 사용했지만, 4번 노드가 실제로 읽은 값을 알지는 못한다. 다만 4번 노드의 실제 값이 110에서 116사이의 값(113-3, 113+3)이라는 것만 알 수 있다. 그림에서는 4번 노드가 실제로 읽은 값이 112이므로, 기본값과의 차이가 1이지만, 부모 노드의 입장에서는 제일 차이가 많이 나는 경우, 즉 110이나 116일 수 있다는 가정을 하고 계산을 해야 한다. 그렇게 해야 SUM을 한 결과가 갖게 되는 에러가, 주어진 범위를 벗어나지 않게 된다. 그러므로 2번 노드에서는, te_2에서 데이타를 전송하지 않은 자식 노드의 부-트리의 에러 범위 te_4를 뺀 값을 가지고 필터링 여부를 결정해야 한다. 즉, $te_2-te_4=6$이고 $|A_2-D_2|=7$이므로, 2번 노드는 데이타를 전송한다.

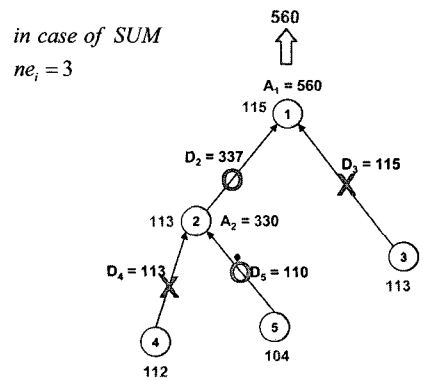

그림 3 필터링과 네트워크 내 집계연산

3번 노드도 단말 노드이므로, 4번 노드나 5번 노드에서와 같은 방법으로 필터링을 한다. 그리고 마지막으로 루트 노드에서는, 2번 노드가 전송한 값 330과 3번 노드의 기본값 115, 그리고 1번 노드가 읽은 값 115의 합인 560을 기지국으로 전송한다. 이 560이라는 값은 최종 결과이므로, 루트 노드는 그 값을 기지국으로 항상 전송해 주어야 한다.

3. 관련 연구

필터링을 적응적으로 하는 방법은 이전에 스트림 데이타를 처리하기 위해 연구된 여러 알고리즘에 기반을 두고 있다. 그 알고리즘 중에 대표적인 것이 [9]이다. [9]는 많은 데이타 소스에서 발생하는 스트림 데이타에 대한 연속 질의(continuous queries)를 처리할 때, 질의 결과가 전체 에러 범위를 벗어나지 않으면서 동시에 전체 메시지 전송 횟수를 줄일 수 있도록, 각 데이타 소스의 에러 범위를 적응적으로 조정하는 알고리즘을 제안했다. 하지만 이 알고리즘은 데이타 소스들 사이에 계층 구조를 고려하지 않았기 때문에, 모든 소스들이 독립적이라고 가정하였다. 반면에 센서 네트워크에서는 집계연산이 트리 구조로 이루어지기 때문에, 각 노드의 필터링이 자신의 조상 노드들에게도 영향을 미치게 되므로 각 노드들이 독립적이 아니다. 또한, [9]에서는 각 데이타 소스의 필터를 조절하는 작업을 중앙 서버에서 담당하기 때문에, 앞에서 언급하였듯이 센서 네트워크에서 그대로 적용할 수가 없다. 그래서 센서 네트워크에 적합한 데이타 필터링 방법이 필요하게 되었는데, 우리가 아는 한 [8]의 연구가 지금까지 이 문제에 관한 연구이다. [8]의 연구에서는 필터링과 네트워크 내 집계연산을 결합함으로써, 각 센서에서의 메시지 전송을 최소화 하였다. 다음 절에서 [8]에서 제안한 알고리즘을 설명하도록 하겠다.

3.1 계층적 데이타 집계를 위한 Deligiannakis의 프레임워크

[9]와 비슷하게, [8]의 연구에서도 다섯 개의 표준 집계 함수인 SUM, AVG, COUNT, MAX, 그리고 MIN을 대상으로 하고 있다. COUNT 함수는 항상 정확하게 계산할 수 있고, AVG는 SUM과 COUNT를 이용해서 계산할 수 있다. MAX와 MIN은 서로 대칭적인데, [4]에서 MAX와 MIN은 몇 개의 AVG 질의를 통해 계산할 수 있다는 것을 보였다. 따라서 논의의 초점이 되는 함수는 SUM이나 AVG가 되는데, [9]에서는 AVG에 초점을 두고 알고리즘을 전개했고, [3]에서는 SUM을 기준으로 알고리즘을 제안했다.

센서 노드에서 에러 범위를 적응적으로 조정할 때 중요한 것은, 과연 무엇을 기준으로 에러 범위를 조정할 것인가 이다. 어떤 노드의 에러 범위를 늘여주고, 또 어떤 노드의 에러 범위를 줄여 주어야 하는가가, 필터링을 적응적으로 하는데 있어서 제일 중요한 문제가 된다. Deligiannakis가 제안한 프레임워크에서는, 각 노드의 에러 범위를 조정했을 때 얻게 되리라 예상되는 예상 이득(expected gain)에 근거해서 각 노드의 에러 범위를 조정한다.

[8]에서는, 일정 주기 T마다 한번씩 각 노드의 에러 범위를 조정하는데, 이를 위해 지난 주기 동안에 필요한 정보를 수집해 놓는다. 그리고 그 정보를 이용해서 에러 범위를 조정했을 때 예상되는 *잠재 이득(potential gain)*을 계산해서, 그 잠재 이득이 큰 노드에게 더 많은 에러를 더해주고, 잠재 이득이 작은 노드의 에러 범위는 작아지도록 조정을 한다. 여기에서 에러 범위 조정을 위해 필요한 정보는 각 노드가 계산하는 두 개의 통계 수치인데, C_{shrink}와 C_{expand}라 한다.

에러 범위 조정을 할 때 [8]에서 처음 하는 일은, 일단 모든 노드의 에러 범위를 *shrinkFactor*만큼 줄인다. 그리고 지난 주기 동안에 계산된 각 노드의 C_{shrink}와 C_{expand}를 기반으로, 각 노드의 추가 에러 범위(additional error bound)를 구해서 각각의 에러 범위에 더해준다. 각 노드에서는 W라는 에러 범위를 기준으로 필터링을 수행하는데, 만약 에러 범위를 *shrinkFactor*만큼 줄였을 경우, 즉 에러 범위가 *shrinkFactor***W*인 경우와, 에러 범위를 dW만큼 늘였을 경우, 즉 에러 범위가 $W+dW$인 경우에 대해, 각각 메시지를 몇 번 전송하게 되는지를 계산한다. 이 값이 각각 C_{shrink}와 C_{expand}가 된다. 그렇게 한 주기 동안 C_{shrink}와 C_{expand}를 구하면, 당연히 C_{shrink}가 C_{expand}보다 큰 값이 된다. 에러 범위가 커지면 전송하게 되는 메시지 수가 줄어들기 때문이다.

한 주기 T가 끝나면, C_{shrink}에서 C_{expand}를 빼서 *잠재 이득*을 구한다.

$$Gain_i = C_{shrink} - C_{expand}$$

*Gain*이 구해지면, 모든 노드는 자신의 *Gain*을 부모 노드로 전송해 가면서 *누적 이득(cumulative gain)*을 구한다. 어떤 노드 N_i의 누적 이득은 N_i를 루트 노드로 하는 부-트리에 속하는 모든 노드들의 잠재 이득을 다 합친 것이다. 각 노드의 추가 에러 범위를 구하는 것은, 루트 노드부터 탑-다운(top-down) 방식으로 진행이 되는데, 그때 이 누적 이득이 사용 된다.

모든 노드의 누적 이득이 구해지면, 루트 노드부터 아래로 내려가면서 각 노드의 에러 범위에 더해주어야 할 추가 에러를 구한다. 물론 이때, 이미 각 노드의 에러 범위는 *shrinkFactor*만큼 줄여져 있다. 이처럼 모든 노드가 에러 범위를 *shrinkFactor*만큼 줄이게 되면,

$E_{global}(1-shrinkFactor)$만큼의 전체 에러에 대한 여유분 *E_Additional*이 생기는데, 이 *E_Additional*을 각 노드의 누적 이득을 기준으로 모든 노드에게 분배해서 더해준다. 그리고 나면, 다음 주기 동안에 필터링에 적용 되어야 할 각 노드의 조정된 에러 범위가 정해진다.

3.2 기존 연구의 단점

[8] 뿐만 아니라, 데이타 소스에서의 적응적 필터링을 위한 연구들에서 제안하는 방법들은 두 가지 공통된 단점이 있다.

- 에러 범위 조정 주기 T를 어떻게 정할 것인가?
  T를 너무 짧게 잡으면, 에러 범위 조정을 너무 자주 하게 되어, 에러 범위 조정으로 인한 오버헤드가 더 커지게 된다. 또, T가 너무 짧을 경우에는, 에러 범위 조정을 위해 지난 주기 동안에 수집한 에러 범위 조정을 위한 정보가 부정확 할 수 있다.
- 에러 범위 조정을 위해 필요한 통계 정보를 계속해서 계산해야 하고, 그 수집한 정보를 일정 주기마다 한번씩 전송해야 하는 오버헤드가 있다.

그러므로, 에러 범위 조정이 위에서 설명한 문제들을 극복할 수 있다면, 보다 효과적으로 데이타 필터링을 수행할 수 있을 것이다. 다음 장에서 제시할 *ARA* 프로토콜은 각 노드에서 매 순간 스스로 에러 범위를 조정하기 때문에 사용자가 에러 조정 주기를 결정해 주지 않아도 된다. 또한, 이로 인해 기존 방법보다 더 효과적이고 적응적인 필터링을 제공하게 된다. 마지막으로 가장 중요한 점은 *ARA* 프로토콜은 에러 범위 조정을 위해 사용되는 부가적인 메시지 교환이 거의 없기 때문에, 기존 방법에 비해 훨씬 에너지 효율적이다.

4. 제안하는 방법

*ARA*는 프로토콜 기반(protocol-based)의 적응적 에러 범위 조정 방법이다. 프로토콜 기반이라는 것은, [8]과 같은 기존 연구들과 달리, 에러 범위 조정을 위한 특별한 메시지를 주고 받지 않고, 노드와 그 부모 노드와의 정해진 약속에 따라 에러 범위를 스스로 조정하는 것(self-adjustment)을 뜻한다. 센서 노드가 에러 범위를 스스로 조정한다는 것은, 약속되어 있는 만큼 에러 범위를 스스로 늘이거나 줄인다는 것인데, 이 에러 범위 조정은 특별한 주기에 한번씩 발생하는 것이 아니라, 매 순간 센서 노드가 데이타를 측정해서 부모 노드에게 전송하거나 전송하지 않을 때마다 계속 일어난다. 즉 기존 연구들과 달리, 특별한 에러 범위 조정 기간이 없다. 또, 다른 연구들과 달리, 매 주기마다 에러 조정을 위해 통계 수치와 같은 정보를 전송해야 하는 오버헤드가 없다. 그러므로 본 연구에서 제안하는 *ARA*를 이용하면, 기존 연구들에서 발생하는 단점을 극복할 수 있다.

본 논문에서는, 센서 네트워크에 새로운 노드가 추가되거나 기존의 노드가 제거되어 집계 트리가 변경되는 것은, [5]에서 제안하는 방법 등을 이용해서 빠른 시간 안에 집계 트리가 재구성된다고 가정한다. 또한 [5]에서 말하는 것처럼, 모든 센서 노드는 자신의 부모 노드와 자식 노드들에 대한 정보를 가지고 있어서, 어떤 노드가 자신의 부모 노드이고, 또 어떤 노드들로부터 데이타를 기다리고 있어야 하는지를 알고 있다고 가정한다.

표 1 ARA를 설명하기 위한 표기

기호	의미
n	센서 노드의 수
N_i	센서 노드 (i=1,…, n)
E_{global}	한 질의에 대한 최대 허용 에러 (입력 매개변수)
V_i	N_i의 측정된 값
D_i	N_i의 기본값
A_i	N_i를 루트 노드로 하는 부-트리의 집계 결과
ne_i	N_i의 최대 허용 에러
te_i	N_i를 루트 노드로 하는 부-트리의 허용 가능한 에러
δ_s	최대 허용 에러 감소 인자
δ_e	최대 허용 에러 증가 인자

표 1에서는 본 논문에서 사용할 기호에 대해 정리해 놓았다. 이 중 E_{global}은, 질의를 요청하는 사용자가, 그 질의 결과에 대해 허용 가능한 에러 범위를 명시한 수치인데, 각 센서 노드의 에러 범위를 할당하고 조정하는데 근간이 되는 값이다.

사용자가 센서 네트워크에 E_{global}과 함께 어떤 질의를 요청하면, 각 센서 노드에 그 질의에 대한 에러 범위가 초기화되어야 한다. 에러 범위 초기화는 여러 방법을 사용해서 할 수 있지만, *ARA*에서는 모든 센서 노드에 똑같은 에러 범위를 할당한다. 즉, 아래처럼 센서 노드 N_i의 에러 범위 ne_i의 초기값을 E_{global}/n으로 균일하게 할당한다.

$$ne_i := \frac{E_{global}}{n}$$

또한 N_i의 트리 에러인 te_i도 초기화 시켜 주어야 하는데, N_i와 N_i의 모든 자손 노드(descendant nodes)의 ne_i를 모두 합해주면 된다. 그런데 ne_i가 모두 균일하게 초기화되기 때문에, te_i는 아래처럼 초기화 할 수 있다. $N_Sub(N_i)$는 N_i를 루트 노드로 하는 부-트리의 노드 개수를 나타낸다.

$$te_i := N\_Sub(N_i) \times \frac{E_{global}}{n}$$

*ARA*를 이용하면 일반적인 집계 함수를 모두 처리할 수 있지만, 우리는 [8]에서처럼 SUM을 포함하고 있는

질의에 초점을 두고 설명하겠다.

4.1 능동 조정과 수동 조정

사용자가 입력한 질의를 받은 센서 노드는, 질의에 명시되어 있는 일정 주기마다 데이타를 측정한다. 그리고 측정된 데이타의 값 V_i와 기본값 D_i와의 차이가 ne_i보다 크면, 그 데이타를 부모 노드에게 전송한다. [8]이나 [9]와 같은 기존의 적응적 필터링 방법과 달리, *ARA*는 한번 데이타를 전송하거나 전송하지 않을 때마다 매번 그 노드의 에러 범위를 스스로 조정한다. 이른바 에러 범위의 자기 조정(self-adjustment)이다. 에러 범위의 자기 조정에는 두 가지 형태가 있다. *ARA*는 이 두 개의 조정을 *능동 조정*(active adjustment)과 *수동 조정*(passive adjustment)이라고 부른다. 능동 조정은 데이타를 보내는 노드에서 발생하는데, 즉 부모 노드에게 데이타를 전송해야 할 자식 노드에서 발생한다. 노드 N_i의 데이타가 필터링이 되어 부모 노드에게 전송되지 않으면, 자식 노드는 자신의 ne_i와 te_i를 δ_s만큼 줄인다. 반대로 N_i가 부모 노드에게 데이타를 전송했으면, N_i는 자신의 ne_i와 te_i를 δ_e만큼 증가시킨다. 이것이 *능동 노정*이다.

반면에, 수동 조정은 데이타를 받는 부모 노드에서 발생한다. 자식 노드는 데이타를 부모 노드에게 전송하거나 전송하지 않는 두 개의 경우가 있다. 자식 노드에서 능동 조정이 발생해서 te가 변경되었다면, 부모 노드는 자식 노드의 변경된 te를 반드시 알아야 한다. 하지만 에러 조정은 자식 노드와 부모 노드가 서로 약속하고 있는 것이기 때문에, 부모 노드는 자식 노드의 능동 조정을 암묵적으로 알 수 있다. 모든 노드는 자식 노드가 데이타를 보냈는지 보내지 않았는지를 알기 때문에, 정해진 시간 안에 어떤 자식 노드로부터 데이타가 오지 않으면, 그 자식 노드의 데이타는 필터링이 되어 전송되지 않은 것이라는 것을 부모 노드는 알 수 있다. 물론 그 자식 노드가 작동이 멈추었거나 전송 에러가 생겨서 데이타가 전송되지 않을 수도 있다. 하지만 그런 예외적인 상황은 별도로 고려할 수 있다[5]. 자식 노드 N_i가 능동 조정을 해서 te_i가 변경이 되면, 부모 노드 N_j는 ne_j를 조정함으로써 자신의 te_j가 변하지 않게 한다. 이것이 수동 조정이다.

위에서 설명한 것처럼, 능동 조정은 반드시 수동 조정을 야기한다. 그러므로 능동 조정이 아무 제약 없이 항상 이루어질 수 있는 것은 아니다. 능동 조정은 반드시 수동 조정을 수반하기 때문에, 수동 조정이 아무 문제없이 수행될 수 있어야 능동 조정도 수행될 수 있다. 예를 들어, 어떤 노드가 데이타를 전송해서 자신의 ne와 te를 δ_e만큼 늘여 주었다고 가정하자. 그러면 그 노드의 부모 노드에서는 수동 조정이 일어나서 ne를 δ_e만큼 줄여야 한다. 그런데 부모 노드는, 자신의 ne가 δ_e보다 작아서 δ_e만큼 줄이지 못하는 상황이 있을 수 있다. 이와 같은 예외 상황에 대해서는 반드시 적절한 처리를 해 주어야 하는데, 4.2절에서 이러한 예외 처리에 대해 설명한다.

이 능동 조정과 수동 조정의 상호 작용이 *ARA*의 핵심이다. 센서 노드에서의 데이타 필터링을 적응적으로 하기 위해 제일 중요한 것은, 전체 에러 범위 E_{global}이 데이타 전송이 많은 노드에게 더 많이 분배가 되어서, 모든 센서 노드의 데이타 전송 횟수가 최대한 균형을 이루게 하는 것이다. 위에서 설명한 에러 범위의 능동 조정과 수동 조정은, 전체 에러 범위 E_{global}을 보다 효과적으로, 또 거의 오버헤드가 없이 분배 해 준다.

ARA에서는 매 라운드마다 모든 부모와 자식 사이에서 능동 조정과 수동 조정이 이루어진다. 즉, 주어진 에러 범위가 지역적으로 조정 또는 분배된다. 다른 식으로 표현하면, 매 라운드마다 각 노드에 주어진 에러 범위들이 메시지 발생이 빈번한 노드들로 흘러가게 된다. ARA의 특징적인 장점은 이런 '흐름'이 추가적인 메시지 교환 없이 매 라운드마다 계속 일어난다는 점이다. 따라서 집계 트리의 높이(height)만큼의 라운드 후에는 리프 노드에서 줄인 에러 범위가 루트 노드에 도달할 수 있고, 그 반대의 경우도 가능하게 된다. 즉, ARA에서도 전역적인 에러 범위 조정을 이루게 된다. 기존 연구 [8]에서는 정해진 주기마다 전역적인 에러 범위 조정이 일어나는데, 이 주기는 일반적으로 집계 트리의 높이(height)보다 훨씬 크다. 즉, ARA는 지역적인 에러 범위 조정을 통하여, 기존 연구보다 더 빠르고 적응적으로 '전역적'인 에러 범위 조정을 달성하는 것이다. 본 논문에서는 실험을 통하여 '메시지 발생 횟수 분포'를 분석함으로써 ARA가 더 효과적인 에러 범위 조정을 이루는 것을 보였다.

4.2 예외 처리

ARA 프로토콜을 이용한 자기 조정(self-adjustment)은, 대부분 부모 노드와 자식 노드간의 아무런 메시지 교환 없이도 가능한데, 예외적으로 메시지 교환이 필요한 경우가 있다. 능동 조정은 노드 스스로 예외적인 경우를 고려해서 대처할 수 있지만, 그 능동 조정에 의해 야기되는 부모 노드에서의 수동 조정에 예외가 생기는 경우에는, 능동 조정을 했던 자식 노드가 그 예외를 알 수 없다. 예를 들어, 어떤 노드 N_i가 데이타를 전송해서 ne_i와 te_i를 δ_e만큼 증가시키게 되면, 그 노드의 부모 노드 N_j에서는 ne_j 를 δ_e만큼 줄여야 한다. 그런데 ne_j가 δ_e보다 작은 경우가 있을 수 있다. ne_j는 0보다 작으면 안되기 때문에, N_i가 ne_i를 δ_e만큼 줄였더라도 N_j는 ne_j를 0까지 밖에 줄일 수 없다. 즉 N_j는 ne_j를 δ_e보다 적게 줄여야 하는 경우가 생기는 것이다. 하지만 N_i는 이

미 능동 조정을 통해 ne_i를 δ_e만큼 줄인 상태이기 때문에, 이 예외 상황에 대한 처리를 해 주어야 한다.

*ARA*에서는 이 예외 상황을 만나면 부모 노드가 자식 노드에게 *알림 메시지(notification message)*를 보낸다. 알림 메시지의 내용은, ne_j가 δ_e만큼 줄지 못하고 그보다 적게 줄었기 때문에(수동 조정의 예외 발생), 자식 노드 N_i는 ne_i와 te_i를 δ_e만큼 늘인 능동 조정을 취소하고, N_j가 줄인 만큼만 ne_i와 te_i를 증가시키라는 메시지가 될 것이다.

알림 메시지 줄이기. 일반적인 경우에 알림 메시지는 많이 발생하지 않는다. 하지만 전체 에러가 지나치게 작은 경우와 같은 극단적인 경우에는, 알림 메시지가 많이 발생할 수 있다. 그래서 *ARA*에서는, 어떤 노드 N_i가 한 번 알림 메시지를 받으면, 그 후 몇 라운드 동안은 데이타를 전송하더라도 능동 조정을 하지 않는다. 물론 이 사실을 그 부모 노드도 알 수 있다. N_i의 부모 노드 N_j는, N_i에게 알림 메시지를 보내고, 약속된 라운드만큼 N_i와 연관된 수동 조정을 하지 않으면 되는 것이다. N_i가 알림 메시지를 받았다고 가정하자. 부모 노드와 자식 노드와의 관계만 고려해 볼 때, 0이었던 ne_j가 δ_e보다 커지기 위해서는, N_j의 자식 노드 개수가 b_j개라고 가정할 때, 최소한 $\delta_e/(b_j \times \delta_s)$ 라운드가 지나야 한다. 이 기간을 에러 범위 감소 기간(error bound decreasing period)라 부르는데, 이 주기는 N_j의 모든 자식 노드가 데이타를 전송하지 않을 때, ne_j가 δ_e보다 커질 때까지 필요한 최소 라운드 수다. 그렇기 때문에, 노드 N_i가 알림 메시지를 받으면, 이어지는 에러 범위 감소 기간 동안에 데이타를 전송하는 경우가 있더라도, 그 때에는 에러 조정을 하지 않기로 부모 노드와 약속한다. 즉, N_i는 에러 범위 감소 기간에는 에러 범위를 줄이기만 하면서, 부모 노드의 에러 범위가 δ_e까지 증가하게 해 준다. 이렇게 하면, 어떤 노드가 연속적으로 데이타를 전송하더라도, 알림 메시지가 계속해서 많이 발생하지 않는다. 추가적으로, 알림 메시지를 더 줄이기 위해서 형제 노드의 상태를 고려할 수 있다. 어떤 노드의 에러 범위 감소 기간을 결정할 때, 계산된 $\delta_e/(b_j \times \delta_s)$의 값에 형제 노드의 남아있는 에러 범위 감소 기간 중 최대값을 더해주면, 보다 효과적인 에러 범위 감소 기간을 정할 수 있다.

에러 범위 조정 인자값. δ_s와δ_e가 모두 크면 클수록 에러 범위 조정이 크게 이루어지지만, 대신 알림 메시지가 발생할 가능성이 많아진다. 예를 들어, δ_s와 δ_e가 초기 에러 범위 값인 eGlobal/n 보다 크다면, 첫 라운드부터 알림 메시지가 발생할 것이다. 따라서 적절한 값을 결정하는 것이 중요한데, 본 논문에서는 다음과 같은 원칙으로 결정하였다. δ_e는 각 노드의 초기 에러 범위인 eGlobal/n의 10분의 1을 사용하고, δ_s는 δ_e의 10분의 1을 사용하였다.

5. 실험 및 분석

본 논문에서는 모의 실험을 통하여, 센서 노드의 수, 전체 에러 범위, 에러 조정 주기 등 여러 매개변수들을 변경해 가면서, 에러 조정이 적응적으로 잘 이루어지는가를 테스트 하였다. [5]에서처럼, 각 센서 노드들은 동기화 되어 있다고 가정했다. 실험에서는 아래의 세 알고리즘을 비교 분석했다.

1. *Uniform*: 이것은 허용 가능한 전체 에러를 균일하게 분포한 후, 아무런 에러 조정 없이 정적으로 필터링을 하는 경우를 나타낸다. 따라서 에러 조정을 위한 아무런 오버헤드가 발생하지 않는다.

2. *GAIN*: 이것은 [8]에서 제안한, 예상 이득에 기반해 에러를 조정하는 것을 나타낸다. [8]에서 설명하고 있는 알고리즘을 구현하였다. [8]에서 제시된 모든 최적화 방법을 구현하였다. 본 논문의 실험결과에서는 piggy-back방식 최적화는 포함되지 않았으나, 이 경우에도 실험결과는 동일한 경향을 보여주었다.

3. *ARA*: 본 논문에서 제안하는 에러 조정 방법이다. *GAIN*과 달리 에러 조정 주기가 없고, 각 센서 노드가 자신의 데이타 전송 여부에 따라 스스로 에러를 조정한다.

5.1 실험 환경

집계 트리 구성과 데이타 생성. 집계 트리 구성과 데이타 생성을 위해서, 우리는 [8]에서 사용한 방법과 거의 유사한 방법을 택했다. 확장성(scalability) 실험을 제외한 다른 모든 실험에서는 500개의 센서 노드를 사용해서 집계 트리를 랜덤으로 구성했는데, 한 노드의 최대 자식 노드의 개수는 4로 했다. 세 알고리즘 모두 각 노드의 초기 에러 범위는 처음에 균일하게 할당했다. 우리는 현실에 가까운 데이타 생성을 위해서, [8]에서처럼 센서 노드를 *workaholic* 노드와 *regular* 노드로 구분했다. regular 노드는 1%의 확률로 데이타 값이 변하는 노드다. 즉, 센서가 데이타를 100번 읽으면 그 중에 99번의 데이타 값에 변화가 없는 노드를 regular 노드라고 부른다. 반면, workaholic 노드는 계속해서 데이타 값이 변하는 노드를 말한다. $P_{workaholic}$은 센서 노드들 중 workaholic 노드가 차지하는 비율을 말한다. 데이타는 랜덤하게 생성하였는데, regular 노드와 workaholic 노드의 데이타 변동 폭은 (0...2) 사이의 수 중에서 랜덤하게 선택하였다. 추가로, workaholic 노드의 데이타 변동 폭을 더 늘려서, (0...20) 사이의 수 중에서 선택하여 실험하기도 했다. 확장성 실험을 위해서는 센서 노드의 개수를 500개에서 5000개까지 늘려 가면서, 에러 조정이 효과적으로 이루어지는지 확인했다.

매개변수 설정. *GAIN*을 위해서는 *shrinkFactor*와 에러 조정 주기를 설정해야 하는데, 실험을 통해 *GAIN*의 결과가 가장 좋은 에러 조정 주기를 찾아서, 그 조정 주기마다 에러를 조정했다. *shrinkFactor*는 [8]에서 실험한 대로 0.95로 설정했다. *ARA*를 위해서는 δ_s와 δ_e를 결정해야 한다. 본 실험에서는, δ_e는 에러 초기값의 10분의 1(즉, 10%)로 설정했고, δ_s는 δ_e의 10분의 1(즉, 초기값의 1%)로 설정했다. 이 설정 값들이 최적의 결과를 내는 것은 아니지만, 실험을 통해서 허용할 만한 좋은 결과를 내는 것을 확인하였다. 그림 4~8의 실험에서는, $P_{workaholic}$을 0.2로 설정하고 실험했고, 그림 9의 실험에서는, $P_{workaholic}$을 0에서 1까지 변화시키면서 각 알고리즘의 성능을 비교했다.

5.2 실험 결과

에러 조정 주기 테스트. 그림 4에서는, 본 실험에서 사용할 데이타에 대해 *GAIN*의 에러 조정 주기를 변화시켜 가면서 *GAIN*의 성능을 실험했다. 센서 노드는 500개를 사용했고 *eGlobal*은 300으로, *shrinkFactor*는 0.95로 설정하고, 전체 10,000라운드를 실험했다. 그림에서 보듯이, *GAIN*은 에러 조정 주기를 얼마로 하느냐에 따라 성능 차이가 많이 나는 것을 볼 수 있다. 이 실험에서는 에러 조정 주기가 100일 때, *GAIN*이 발생시키는 전체 메시지 수가 제일 적었다. 그림에 표시하지는 않았지만, 이 때에도 *ARA*보다는 결과가 좋지 않았다. 에러 조정 주기를 얼마로 하느냐는 데이타 값의 변화 정도나 *shrinkFactor*등 여러 가지 변수에 따라 달라져야 하지만, 이 실험을 토대로 다른 모든 실험에서 *GAIN*의 에러 조정 주기를 100으로 설정하고 실험했다. 이 실험에서는 정해진 데이타에 대해서 *GAIN*을 위한 최적의 에러 조정 주기를 찾은 것이다. 하지만 실제로는 이렇게 최적의 에러 조정 주기를 찾을 수는 없다는 것이 *GAIN*의 단점이다.

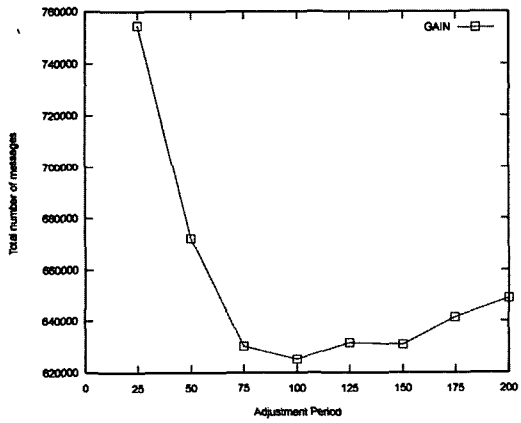

그림 4 에러 범위 조정 주기에 따른 메시지 수 변화

eGlobal 테스트. 이 실험에서 우리는, 500개의 센서 노드를 사용해서, *eGlobal*을 100에서 1000까지 변화시켜가면서, 각 알고리즘이 발생시키는 전체 메시지 개수를 비교하였다. 그림 5에서 보듯이, 모든 구간에서 *ARA*의 성능이 가장 좋은 것을 확인할 수 있다. *eGlobal*이 매우 커지게 되면, 세 알고리즘의 성능이 거의 비슷해진다. 하지만, 실질적으로는 전체 에러가 지나치게 큰 경우는 의미가 없다.

노드별 메시지 발생 횟수 분포 테스트. 그림 6은 10,000라운드의 실험을 마쳤을 때, 각 센서 노드가 생성한 메시지 개수를 나타낸다. 센서 노드는 500개를 사용했고, *eGlobal*은 300으로 설정했다. 세 알고리즘 모두, 루트 노드는 항상 데이타를 기지국으로 전송하기 때문에, 루트 노드가 전송한 메시지 수는 모두 10,000이 된다. 그래서, 그래프에는 루트 노드를 제외한 499개의 노드에 대해서만 메시지 수를 표시했다. 그림에서 보듯이, *Uniform*에서 더 많은 노드들이 많은 메시지를 발생시

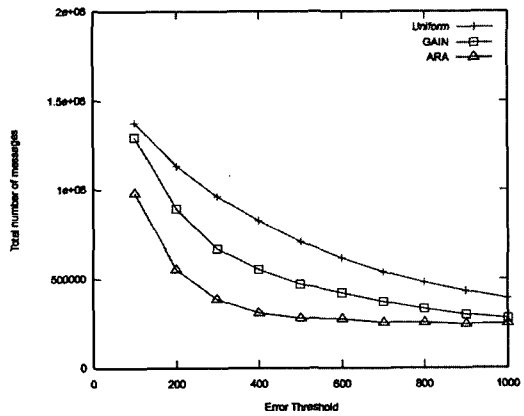

그림 5 최대 허용 에러의 크기에 따른 메시지 수 변화

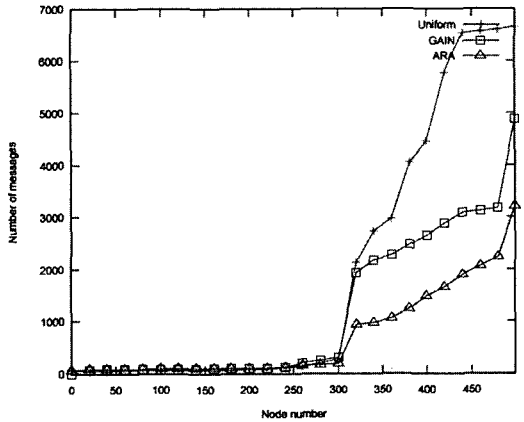

그림 6 메시지 발생 횟수 분포

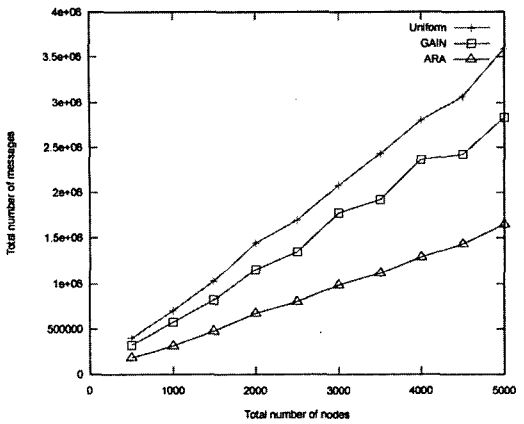

그림 7 센서 네트워크 크기에 따른 메시지 수 변화

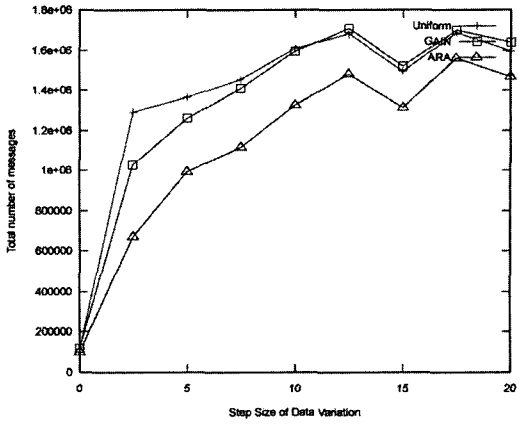

그림 8 데이타 변동폭에 따른 메시지 수 변화

키고 있음을 볼 수 있다. 반면에 *ARA*에서는 다른 알고리즘보다 많은 메시지를 발생시키는 노드의 수가 적음을 알 수 있다. 이 그림을 통해서 우리는, *ARA*가 다른 두 알고리즘보다 실질적으로 에러 범위를 더 효율적으로 분배하여 전체적인 센서 네트워크의 유효 기간을 증가시킨다는 것을 알 수 있다.

확장성 테스트. 그림 7에서는 확장성 실험을 하였다. 센서 노드를 500개에서 5000개까지 늘렸을 때, 각 알고리즘의 성능을 비교했다. 센서 노드의 수를 늘릴 때 *eGlobal*도 같은 비율로 늘려주면서 실험을 했는데, *eGlobal*은 센서의 개수에 0.6을 곱해서 구했다. 즉, 센서가 500개일 경우에 *eGlobal*은 300이 된다. 그림에서 보듯이, 센서 노드의 수가 늘어나면서 *ARA*의 성능이 더 좋아지는 것을 확인할 수 있다. 센서의 수가 늘어난다는 것은, 집계 트리의 level이 증가한다는 것인데, 집계 트리의 규모가 커지더라도 *ARA*의 성능이 좋아지는 것을 보이고 있다.

데이타 변동 폭 테스트. 그림 8에서는 workaholic 노드의 데이타 변동 폭을 늘려가며 실험을 했다. 이 실험을 제외한 다른 모든 실험에서는, workaholic 노드와 regular 노드의 데이타 변동 폭이 0에서 2 사이의 수로 같았다. *eGlobal*은 300으로 고정하고 workaholic 노드의 최대 데이타 변동 폭을 0에서 20까지 늘려가며, 세 알고리즘의 성능을 비교했다. 그림에서 보듯이, workaholic 노드의 최대 데이타 변동 폭이 증가하더라도, *ARA*의 성능이 우수함을 알 수 있다.

Workaholic 노드 비율 테스트. 그림 9에서는 $P_{workaholic}$을 0에서 1까지 변화시키면서 세 알고리즘을 비교했다. $P_{workaholic}$이 0이라는 것은 workaholic 노드가 하나도 없다는 것이고, $P_{workaholic}$이 1이라는 것은 모든 노드의 데이타 값이 항상 변한다는 뜻이다. 그림에서처럼, $P_{workaholic}$이 증가하더라도 *ARA*의 성능이 제일 좋다

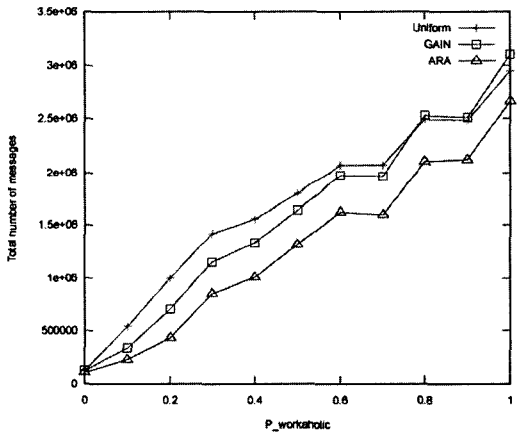

그림 9 Pworkaholic의 변화에 따른 메시지 수 변화

는 것을 알 수 있다.

이 외에도 다양한 실험을 수행하였다. 위에서 설명한 실험에서는, 처음부터 끝까지 regular 노드와 workaholic 노드를 고정시켜 놓고 실험을 한 것이다. 즉, 어떤 노드가 처음에 workaholic 노드로 선택이 되면 그 노드는 실험 끝까지 workaholic 노드라고 가정하고 실험을 했다. 하지만 실질적인 센서 네트워크의 상태를 반영하기 위해서, 주기적으로 workaholic 노드를 변경해 가면서 실험을 해 보았는데, 역시 *ARA*의 성능이 *Uniform*과 *GAIN*에 비해 우수함을 확인할 수 있었다. 또, regular 노드의 데이타 변동 폭과 workaholic 노드의 데이타 변동 폭을 변경해 가면서 실험을 해 보았다. 즉, workaholic 노드의 데이타 변동 폭을 regular 노드의 데이타 변동 폭보다 크게 해서 실험하기도 하고, 또 regular 노드의 데이타 변동 폭보다 작게 해서 실험하기도 하였다. 두 가지 경우 모두, *ARA*의 성능이 *Uniform*과 *GAIN*에 비해 우수하다는 것을 확인할 수 있었다.

6. 결론 및 향후 연구 계획

본 논문에서는, 센서 데이타에 대해 주어지는 집계 질의를 처리할 때, 센서 네트워크의 생명 주기를 증가시키기 위한 방법을 제안했다. 각 질의는 허용 가능한 에러 범위를 명시하고, 이 전체 에러 범위를 기반으로 각 센서에서는 필터링을 수행한다. 본 논문에서는 센서에서의 데이타 필터링을 적응적으로 할 수 있는 효과적인 방법인 *ARA*를 제안했다. 이전 연구들과 달리, *ARA*는 에러 조정 주기를 결정할 필요가 없고, 또 에러 조정을 위한 오버헤드가 매우 적다. 그리고 데이타 값의 변화 정도를 에러 조정에 즉시 반영할 수 있다.

*ARA*는 두 개의 간단한 규약에 기반하고 있다. 첫째는, 센서가 데이타를 전송하지 않으면 에러 범위를 정해진 만큼 줄인다는 것이고, 둘째는, 센서가 데이타를 전송하게 되면 에러 범위를 정해진 만큼 늘인다는 것이다. 이와 같은 자기 조정(self adjustment)은, 모든 센서가 네트워크 내 집계연산을 하기 때문에, 에러 조정을 위한 특별한 메시지 없이도 가능하다.

실험을 통해서 *ARA*의 성능이 에러 조정을 하지 않는 경우와 [3]에서 제안한 예상 이득에 기반한 방법보다 더 효과적인 것을 보여주었다.

향후 연구 과제로는, 예외 상황이 발생할 경우에 발생하게 되는 알림 메시지를 최소화 하거나 없앨 수 있는 연구가 필요하며, 또 메시지 수를 최소화할 수 있는 최적의 δ_s와 δ_e를 결정할 수 있는 연구도 필요하다.

참 고 문 헌

[1] J. Kahn, R. Katz, and K. Pister. Next century challenges: Mobile networking for 'smart dust'. In *Proceedings of MOBICOM*, pp. 271-278, Seattle, America, 1999.

[2] A. Chandrakasan, R. Min, M.Bhardwaj, S. Cho, and A. Wang. Power Aware Wireless Microsensor Systems. In *Proceedings of ESSCIRC*, Florence, Italy, 2002.

[3] A. Cerpa, J. Elson, M. Hamilton, and J. Zhao. Hibitat Monitoring: Application Driver for Wireless Communications Technology. In *Proceedings of SIGCOMM*, 2001.

[4] V. Shnayder, M. Hempstead, B. Chen, G. W. Allen, and M. Welsh. Simulating the Power Consumption of Large-Scale Sensor Network Applications. In *Proceedings of ACM SenSys*, 2004.

[5] S. Madden, M. J. Franklin, J.M. Hellerstein, and W. Hong. Tag: A Tiny Aggregation Service for ad hoc Sensor Networks. In *Proceedings of OSDI*, Boston, America, 2002.

[6] C. Intanagonwiwat, D. Estrin, R. Govindan, and J Heidermann. Impact of Network Density on Data Aggregatio in Wireless Sensor Networks. In *Proceedings of ICDCS*, 2002.

[7] Y. Yao and J. Gehrke. The Cougar Approach to In-Network Query Processing in Sensor Networks. *SIGMOD Record*, 2002.

[8] Antonios Deligiannakis, Yannis Kotidis, and Nick Roussopoulos. Hierarchical In-Network Data Aggregation with Quality Guarantees. In *Proceedings of EDBT*, pp. 658-675, 2004.

[9] Chris Olston, Jing Jiang, and Jennifer Widom. Adaptive Filters for Continuous Queries over Distributed Data Streams. In *Proceedings of SIGMOD*, pp. 563-574, San Diego, America, 2003.

[10] Xingbo Yu, Sharad Mehrotra, Nalini Venkatasubramanian, and Weiwen Yang. Approximate Monitoring by Aggregation-Oriented Clustering in Wireless Sensor Networks, ICDE, submitted, Bangalore, India, 2003.

[11] S. Madden, M. J. Franklin, J.M. Hellerstein, and W. Hong. The Design of an Acquisitional Query processor for Sensor Networks. In *Proceedins of SIGMOD*, pp. 491-502, San Diego, America, 2003.

[12] R. Min, M.Bhardwaj, S. Cho, A. Sinha, E. Shih, A. Wang, and A. Chandrakasan. Low-power wireless sensor networks. In *Proceedings of VLSI Design*, Bangalore, India, 2001.

[13] K. Kalpakis, K. Dasgupta, P. Namjoshi. Efficient Algorithms for Maximum Lifetime Data Gathering and Aggregation in Wireless Sensor Networks. *UMBC CS TR-02-13*, 2002.

박 노 준

2003년 아주대학교 정보 및 컴퓨터공학부 학사. 2005년 한국과학기술원 전자전산학과 석사. 2005년~현재 삼성전자 기술총괄 시스템연구소 연구원. 관심분야는 데이타베이스, 센서 네트워크, 운영체제

현 동 준

1999년 한국과학기술원 전산학과 학사 2000년 한국과학기술원 전자전산학과 석사. 2000년~현재 한국과학기술원 전자전산학과 박사과정. 관심분야는 데이타베이스, 데이타마이닝, 센서 네트워크

김 명 호

정보과학회논문지 : 데이타베이스 제 32 권 제 3 호 참조