데이타 중심 센서 네트워크에서 에너지 효율성을 고려한 비균등 네트워크 분할 기법†

A Non-Uniform Network Split Method for Energy Efficiency in a Data Centric Sensor Network

강홍구* / Hong-Koo Kang, 김정준** / Joung-Joon Kim, 한기준*** / Ki-Joon Han

요 약

데이타 중심 센서 네트워크에서는 측정된 데이타의 값에 따라 데이타를 저장하는 센서 노드가 결정되기 때문에 같은 값을 갖는 데이타가 빈번하게 발생하면 이를 저장하는 센서 노드에 부하가 집중되어 에너지가 빠르게 고갈되는 문제가 있다. 또한 센서 네트워크가 확장되면 데이타 저장 및 질의 처리시 목적 센서 노드로의 라우팅 거리가 멀어져 센서 네트워크의 통신비용이 증가되는 문제가 있다. 그러나 기존 연구들은 데이타 저장의 효율적인 관리에만 치우쳐 이와 같은 문제를 효율적으로 해결하지 못하고 있다.

본 논문에서는 데이타 중심 센서 네트워크에서 센서 노드의 부하를 분산시키고 센서 네트워크의 확장에 따른 통신비용을 효율적으로 줄이기 위한 비균등 네트워크 분할(Non-Uniform Network Spilt: NUNS) 기법을 제안한다. NUNS는 센서 네트워크를 센서 노드 개수와 분할된 영역 크기의 차이가 최소가 되도록 비균등 크기의 Partition으로 분할하고 각 Partition에서 발생한 데이타를 각 Partition 내의 센서 노드가 저장함으로써 센서 노드의 데이타 저장 부하를 분산시키고 센서 네트워크의 확장에 따른 통신비용을 줄인다. 또한 NUNS는 각 Partition을 분할된 영역 크기 차이가 최소가 되도록 센서 노드 개수만큼 Zone으로 비균등하게 분할하여 각 센서 노드의 처리 영역으로 할당함으로써 센서 노드에 부하가 집중되는 것을 막고 불필요한 라우팅 비용을 줄인다.

Abstract

In a data centric sensor network, a sensor node to store data is determined by the measured data value of each sensor node. Therefore, if the same data occur frequently, the energy of the sensor node to store the data is exhausted quickly due to the concentration of loads. And if the sensor network is extended, the communication cost for storing data and processing queries is increased, since the length of the routing path for them is usually in the distance. However, the existing researches that generally focus on the efficient management of data storing can not solve these problems efficiently.

† 본 연구는 서울시 산학연 협력사업의 신기술 연구개발 지원사업의 지원으로 수행되었음.

■ 논문접수 : 2007.10.16 ■ 심사완료 : 2007.11.20

* 교신저자 건국대학교 대학원 컴퓨터공학과 박사과정(hkkang@db.konkuk.ac.kr)

** 건국대학교 대학원 컴퓨터공학과 박사과정(jjkim9@db.konkuk.ac.kr)

*** 건국대학교 컴퓨터공학과 교수(kjhan@db.konkuk.ac.kr)

In this paper, we propose a NUNS(Non-Uniform Network Split) method that can distribute loads of sensor nodes and decrease the communication cost caused by the sensor network extension. By dividing the sensor network into non-uniform partitions that have the minimum difference in the number of sensor nodes and the splitted area size and storing the data which is occurred in a partition at the sensor nodes within the partition, the NUNS can distribute loads of sensor nodes and decrease the communication cost efficiently. In addition, by dividing each partition into non-uniform zones that have the minimum difference in the splitted area size as many as the number of the sensor nodes in the partition and allocating each of them as the processing area of each sensor node, the NUNS can protect a specific sensor node from the load concentration and decrease the unnecessary routing cost.

주요어 : 데이타 중심 센서 네트워크, 센서 노드, 네트워크 분할, 부하 균등, 에너지 효율성

Keyword : Data-Centric Sensor Network, Sensor Node, Network Split, Load Balancing, Energy Efficiency

1. 서 론

최근 무선 통신과 처리 및 저장 기능을 갖춘 초소형 센서 기술의 발전으로 환경 모니터링(Environmental Monitoring), 위치 기반 서비스(Location Based Service), 텔레매틱스(Telematics), 홈 네트워킹(Home Networking) 등 센서 네트워크의 활용 분야가 점차 확산되고 있다[1,2,3]. 센서 네트워크는 일반적으로 적게는 수백 개에서 많게는 수만 개의 센서 노드들로 구성되며 하나의 센서 노드는 주변 환경을 측정하는 하나 이상의 센서를 가지고 있다. 이러한 센서 노드는 온도, 습도, 조도 등과 같은 측정된 데이타를 스칼라 형태로 저장한다[4,5].

센서 네트워크는 센서 노드가 측정한 데이타를 저장하는 방식에 따라 크게 세 가지로 분류할 수 있다. 센서 노드가 측정한 데이타를 센서 네트워크의 외부 저장소(또는 중앙 시스템)에 저장하는 외부 저장(External Storage: ES) 방식, 센서 네트워크 내에서 데이타를 측정한 센서 노드가 직접 데이타를 저장하는 지역 저장(Local Storage: LS) 방식, 그리고 센서 노드가 측정한 데이타의 값에 따라 센서 네트워크 내의 해당 센서 노드에 저장하는 데이타 중심 저장(Data-Centric Storage: DCS) 방식이 그것이다[6,7]. 그 중에서 데이타 중심 저장 방식은 외부 저장소와 가까운 센서 노드에 부하가 집중되는 외부 저장 방식의 문제와 질의 처리에서 필요치 않은 센서 노드가 관여되는 내부 저장 방식의 문제를 해결하기 위해 제안된 것으로 현재 데이타 중심 저장 방식을 기반으로 하는 센서 네트워크 연구가 활발히 진행되고 있다[4,7,8,9].

데이타 중심 저장 방식을 사용하는 센서 네트워크(즉, 데이타 중심 센서 네트워크)에서는 측정된 데이타의 값에 따라 데이타가 저장되는 센서 노드가 결정되기 때문에 같은 값을 갖는 데이타가 빈번하게 발생하면 이를 저장하는 센서 노드에 부하가 집중되어 빠르게 에너지가 고갈되는 문제가 발생한다[10,11]. 그리고, 새로운 센서 노드의 추가로 센서 네트워크가 확장되면 데이타를 측정한 센서 노드와 데이타를 저장하는 센서 노드간의 거리가 멀어지면서 데이타 저장 및 질의 처리시 통신비용이 증가하는 문제가 발생한다. 그러므로 데이타 중심 센서 네트워크에서는 센서 노드의 부하를 효율적으로 분산시키고 센서 네트워크의 확장에 따른 통신비용을 줄임으로써 센서 네트워크의 에너지 효율성을 높이는 것이 매우 중요하다[12,13,14].

대표적인 데이타 중심 센서 네트워크에 관한 GHT[7], DIFS[10], DIM[4]과 같은 연구들은 데이타 저장 및 질의 처리를 위해 센서 네트워크를 균등한 크기로 분할하여 센서 노드의 처리 영역을 할당한다. 특히, GHT나 DIFS보다 데이타 저장 및 질의 처리 성능이 항상 우수하다는 것이 입증된 DIM은 센서 네트워크를 균등한 크기로 분할하기 때문에 센서 노드가 존재하지 않는 분할 영역이 발생하고 이들 영역에 해당하는 데이타를 이웃 센서 노드가 대신 저장하게 되므로 특정 센서 노드에 부하가 집중되는 문제가 있다. 또한 센서 네트워크가 확장됨에 따라 데이타 저장 및 질의 처리를 위한 통신비용도 증가하는 문제를 가지고 있다. 이러한 센서 네트워크 확장에 따르는 통신비용 증가 문제 해결을 위한 선행 연구[6]에서 센서 네트워크에 대한 비균등 영역 분할인 Region 생성 알고리즘이 제시되었으나, DIM에서와 같이 센서 노드가 존재하지 않는 Zone으로 인해 발생하는 센서 노드의 부하 집중과 불필요한 라우팅 비용 문제는 해결하지 못하였다.

이와 같은 DIM의 문제점을 해결하기 위해 본 논문에서는 센서 노드의 부하를 분산시키고 센서 네트워크의 확장에 따른 비용을 줄이는 비균등 네트워크 분할(Non-Uniform Network Spilt: NUNS) 기법을 제안한다. 이를 위해 NUNS는 K-D 트리 형태로 네트워크 분할을 2 단계로 수행한다. 먼저 NUNS는 센서 노드의 데이타 저장 부하를 분산시키기 위해 센서 네트워크를 센서 노드 개수와 분할된 영역 크기의 차이가 최소가 되도록 비균등 크기의 Partition으로 분할하고, 각 Partition에서 발생한 데이타를 각 Partition 내의 센서 노드가 저장 및 관리하게 한다. 그러므로 센서 네트워크에서 측정된 데이타는 각 Partition으로 분산 저장되어 센서 노드의 부하가 분산되고, 데이타를 측정한 센서 노드와 데이타를 저장하는 센서 노드 사이의 거리가 줄어들어 센서 네트워크 확장에 따른 통신비용도 크게 줄일 수 있다. 또한 NUNS에서는 각 Partition을 센서 노드 개수만큼 분할된 영역 크기 차이가 최소가 되도록 비균등 크기의 Zone으로 분할한다. 이를 통해 NUNS는 DIM에서와 같이 센서 노드가 존재하지 않는 Zone으로 인해 발생하는 센서 노드의 부하 집중과 불필요한 라우팅 비용을 줄일 수 있다.

본 논문의 구성은 다음과 같다. 제2장의 관련 연구에서는 데이타 중심 센서 네트워크에 관한 기존 연구에 대해 분석한다. 제3장에서는 본 논문에서 제시한 NUNS와 관련 알고리즘에 대해 기술한다. 제 4 장에서는 NUNS와 기존 연구에 대한 비교 실험을 수행하고 결과를 보여준다. 마지막으로 제5장에서는 결론에 대해 언급한다.

2. 관련 연구

본 장에서는 데이타 중심 센서 네트워크에 관한 기존 연구들에 대해 분석한다.

2.1 GHT(Geographic Hash Table)

GHT는 데이타 중심 센서 네트워크에서 측정된 데이타 값을 기반으로 지리적 위치를 생성하고 생성된 지리적 위치와 가장 가까운 센서 노드에 데이타를 저장하는 인덱스이다[7]. GHT는 데이타 저장을 위한 Put 연산과 질의 처리를 위한 Get 연산을 사용하고, 요구하는 데이타를 가진 센서 노드를 찾아가는 라우팅 기법으로 GPSR[2]을 사용한다. 예를 들어, 센서 노드가 Put(event, data)를 호출하면 event를 해싱한 결과로 지리적 위치가 생성되고, 생성된 지리적 위치와 가장 가까운 센서 노드에 data를 저장한다. 그리고 센서 노드가 질의 처리를 위해 Get(event)를 호출한 경우에도 event를 해싱한 결과로 생성되는 지리적 위치와 가장 가까운 센서 노드에 질의를 전달한 후 질의 결과를 돌려받는다.

GHT에서는 센서 네트워크의 확장 시 센서 노드의 데이타 저장 비용을 줄이기 위해 분할 레벨을 d가 주어지면, 전체 센서 네트워크가 같은 크기의 $4^d(d \geq 0)$ 개 영역으로 분할되는 Structured Replication을 사용한다[7,8]. Structured Replication에서는 root point라 불리는 최상위 대표 센서 노드를 지정하고

분할 레벨의 각 영역마다 대표 센서 노드를 지정한다. 질의가 발생하면 root point로부터 하위 레벨의 대표 센서 노드들로 질의가 전달된다. 그러나 Structured Replication은 각 분할 영역마다 대표 센서 노드를 두는 계층 구조를 구성하기 때문에 질의 부하가 root point로 집중되어 root point의 에너지 고갈이 빠르게 진행되는 문제가 있다.

2.2 DIFS(Distributed Index for Features in Sensor networks)

DIFS는 센서 노드의 접근 부하를 줄이고 범위 질의를 지원하는 GHT의 확장이다[10]. GHT의 Structured Replication에서 root point에 부하가 집중되는 문제를 해결하기 위해 DIFS는 자식 노드가 여러 개의 부모 노드를 갖는 Quad-Tree의 변형을 사용한다. 그리고 DIFS는 질의 시 전체 인덱스 노드를 접근하는 Structured Replication에 비해 인덱스 노드에 저장된 데이타 값의 범위와 분할 영역의 크기를 이용하여 인덱스 노드 접근을 줄이고 범위 질의도 지원한다. DIFS도 GHT와 마찬가지로 라우팅 기법으로 GPSR을 사용한다.

DIFS는 GHT의 Structured Replication에서 상위 레벨의 센서 노드에 집중되는 부하를 줄이고 범위 질의를 지원하지만 Structured Replication에 비해 인덱스 노드가 많고 센서 네트워크의 전체 에너지 소모가 다소 높아지는 문제가 있다. 그리고 DIFS는 범위 질의를 지원하지만 일차원 데이타에 대한 인덱스이므로 다차원 데이타에 대한 범위 질의는 지원하지 못하는 한계가 있다. 또한, 같은 값의 데이타 발생이 증가하면 데이타를 저장하는 해당 센서 노드에 부하가 집중되고, 센서 네트워크가 확장되면 통신비용이 증가하는 문제도 가지고 있다.

2.3 DIM(Distributed Index for Multi-dimensional data)

DIM은 데이타 영역과 센서 네트워크의 공간 영역을 서로 매핑하면서 공간 영역과 지리적으로 인접한 센서 노드에 데이타를 저장하는 인덱스이다[4]. DIM은 센서 네트워크를 X축과 Y축으로 번갈아 가면서 분할된 영역에 센서 노드 한 개가 남을 때까지 균등한 크기로 분할한다. DIM에서는 이렇게 분할된 영역을 Zone이라 부른다. 센서 노드가 데이타를 측정하면 데이타를 해싱하여 Zone을 식별하는 비트 스트링 형태의 Zone 코드를 생성하고, 생성된 Zone 코드에 해당하는 Zone 내에 있는 센서 노드에 데이타를 저장한다. 만일 해당 Zone에 센서 노드가 존재하지 않는 경우에는 Backup Zone에 있는 센서 노드에 데이타를 저장한다. Backup Zone은 센서 노드가 존재하지 않는 Zone에 저장될 데이타를 대신 저장하는 이웃 Zone이다. DIM도 라우팅 기법으로 GPSR을 사용한다.

이처럼 DIM은 공간 영역을 데이타 영역과 매핑하여 데이타를 저장하므로 다중 속성 데이타에 대한 저장과 질의 처리가 가능하다. 그러나 DIM도 역시 같은 값의 데이타 발생이 증가하면 데이타를 저장하는 해당 센서 노드에 부하가 집중되고, 센서 네트워크가 확장되면 통신비용이 증가하는 문제를 가지고 있다. 또한 센서 노드가 존재하지 않는 Zone에 해당하는 데이타가 Backup Zone에 해당하는 센서 노드에 저장되므로 이러한 센서 노드의 부하가 증가하는 문제도 발생한다.

3. NUNS(Non-Uniform Network Spilt)

본 장에서는 본 논문에서 제시한 NUNS와 관련 알고리즘에 대해 설명한다.

3.1 Partition 생성

NUNS는 센서 노드의 부하를 효율적으로 분산시키고 센서 네트워크의 확장에 따른 통신비용을 줄이기 위해 센서 네트워크를 비균등 크기의 Partition으로 분할한다. 센서 네트워크를 구성하는 모든 센서 노드의 센싱 영역을 포함하는 사각형을 S(X-Y 평면)이라고 할 때, Partition은 사각형 S를 센서 노드 개수와 분할된 영역 크기의 차이가

최소가 되도록 X축과 Y축을 번갈아 가면서 비균등하게 분할하여 얻어진 사각형이다.

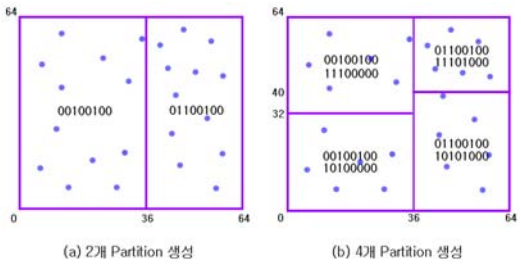
만일 사각형 S에 있는 센서 노드 개수가 짝수 개이면 분할된 두 Partition이 가지는 센서 노드 개수는 같아지지만 센서 노드 개수가 홀수 개이면 두 Partition이 가지는 센서 노드 개수는 같아질 수 없다. 이때는 분할된 두 Partition 중 크기가 더 큰 Partition에 센서 노드가 하나 더 많도록 사각형 S를 분할한다. 따라서, 분할된 두 Partition은 서로 센서 노드 개수가 같거나 최대 한 개 차이가 난다. 이와 같은 과정을 X축과 Y축을 번갈아 가면서 요구하는 분할 횟수 i 만큼 계속 수행하여 최종적으로 Partition들을 구하게 되며, Partition의 분할 횟수가 i이면 생성되는 Partition 개수는 $2^i(i \geq 0)$가 된다. 특히, NUNS에서는 Partition의 분할 횟수가 증가하여 Partition 개수가 늘어나면 데이타 저장 비용이 감소하지만 반대로 질의 처리 비용이 증가하기 때문에 센서 네트워크에서 발생하는 데이타 저장 횟수와 질의 처리 횟수를 고려하여 효율적인 Partition 분할 횟수 i를 적절하게 결정하는 것이 필요하다.

NUNS에서 Partition은 서로 다른 Partition과 구별되는 Partition 코드를 가진다. Partition 코드는 비트 스트링 형태이며 분할 축, Partition 구분자, 분할 축의 위치로 구성된다. <표 1>은 8비트로 구성된 Partition 코드를 나타낸다.

<표 1> 8비트 Partition 코드

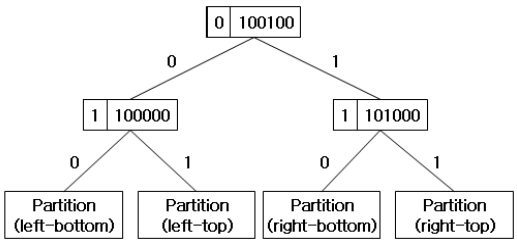
분할 축 (1비트)	Partition 구분자 (1비트)	분할 축의 위치 (6비트)
X 축: 1 Y 축: 0	좌측 (하단): 0 우측 (상단): 1	000000~111111

<표 1>에서 분할 축은 분할시 사용되는 축을 구분하는 비트로 비트 1은 X축을 나타내고 비트 0은 Y축을 나타낸다. Partition 구분자는 분할된 Partition이 좌측(하단)인지 우측(상단)인지를 구분하는 비트로 비트 0은 분할된 Partition이 좌측(하단)임을 나타내고 비트 1은 분할된 Partition이 우측(상단)임을 나타낸다. 그리고 분할 축의 위치는 X, Y축에서 Partition을 분할하는 축의 위치를 나타내는 비트 스트링이고 분할 축의 위치를 위한 비트 개수가 k이면 0과 $2^k(k > 0)$사이의 정수값으로 분할 축의 위치를 나타낸다. 예를 들어, 6비트인 분할 축의 위치는 0과 64사이 정수값으로 분할 축의 위치를 나타낼 수 있다. <그림 1>은 8비트 Partition 코드를 사용하여 Partition을 생성한 예를 보여준다.



<그림 1> Partition 생성 예

<그림 1>에서 실선으로 된 사각형이 Partition을 나타낸다. <그림 1(a)>에서 보는 것과 같이 센서 네트워크는 먼저 센서 노드 개수와 분할된 영역 크기의 차이가 최소가 되도록 X축의 36 위치에서 비균등하게 분할되고, 분할된 두 Partition의 Partition 코드는 각각 (00100100)과 (01100100)이 된다. 그리고 <그림 1(b)>에서 보는 것과 같이 <그림 1(a)>의 좌측 Partition과 우측 Partition은 각각 센서 노드 개수와 분할된 영역 크기의 차이가 최소가 되도록 Y축의 32, 40 위치에서 비균등하게 분할된다. 따라서 분할된 4개의 Partition은 좌측 하단, 좌측 상단, 우측 하단, 우측 상단 순서로 (00100100 10100000), (00100100 11100000), (01100100 10101000), (01100100 11101000)의 Partition 코드를 가진다. <그림 2>는 <그림 1(b)>의 4개 Partition 코드에 대한 K-D 트리 형태의 Partition 트리를 보여준다.



<그림 2> Partition 트리

<그림 2>에서 보는 것과 같이 Partition 트리의 각 노드는 분할 축과 분할 축의 위치를 가지고 있다. 데이타 저장이나 질의 처리 시에 특정 데이타 값에 대한 Partition 코드는 Partition 트리를 루트 노드부터 리프 노드까지 탐색하면서 구할 수 있다. Partition 트리 탐색 시 분할 축에 따라 분할된 Partition이 좌측(하단)이면 Partition 구분자를 비트 0으로 설정하고 좌측 하위 노드로 내려가며, 분할된 Partition이 우측(상단)이면 Partition 구분자를 비트 1로 설정하고 우측 하위 노드로 내려간다. 이렇게 구한 Partition 코드를 통해 Partition의 대각선 좌표들을 구할 수 있다. 예를 들어, <그림 2>의 Partition 트리에서 구한 좌측 하단 Partition의 Partition 코드 (00100100 10100000)이므로 이 Partition의 대각선 좌표는 (0,0), (36,32)임을 알 수 있다. NUNS에서 Partition을 생성하는 알고리즘은 <그림 3>과 같다.

```
Algorithm : CreatePartition(partition, factor, axis)
 1: Begin
 2:   loc = null; pArray[2] = {null, null};
 3:   loc = FindSplitPoint(partition, axis);
 4:   SplitPartition(partition, pArray, axis, loc);
 5:   UpdatePartitionCode(pArray, axis, loc);
 6:   if(axis) then
 7:     axis = 0;
 8:   else
 9:     axis = 1;
10:   end if
11:   for i from 0 to 1 do
12:     if(factor) then
13:       CreatePartition(pArray[i].area, factor--, axis);
14:     else
15:       CreateZone(pArray[i].area, 0);
16:       SavePartition(pArray[i]);
17:     end if
18:   end for
19: End
```

<그림 3> Partition 생성 알고리즘

<그림 3>의 Partition 생성 알고리즘에서 입력 인자인 *partition*은 분할을 수행할 Partition이고, *factor*는 *partition*의 분할 횟수이고, *axis*는 분할 축을 나타낸다. 라인 2는 분할 축의 위치를 저장할 변수 *loc*과 분할된 Partition 정보를 저장할 구조체 *pArray* 배열을 초기화한다. 라인 3은 *partition*을 센서 노드 개수와 분할된 영역 크기의 차이가 최소가 되는 분할 축의 위치를 구하여 변수 *loc*에 저장한다. 라인 4는 분할 축, 분할 축의 위치를 이용하여 *partition*을 비균등하게 분할하고 분할된 *partition* 정보를 구조체 *pArray* 배열에 저장한다. 라인 5는 분할된 Partition에 대한 Partition 코드를 생성하고, 라인 6부터 라인 10까지는 *partition*을 X축과 Y축을 번갈아 분할하기 위해 현재 사용된 분할 축을 다음 분할 축으로 변경한다. 라인 11부터 라인 18까지는 *partition*을 분할하여 얻어진 두 Partition에 대하여 각각 분할 횟수를 검사하여 분할 횟수가 0보다 크면 Partition 분할을 반복 수행하고, 그렇지 않으면 분할된 Partition을 저장한 후 Partition에 대한 Zone을 생성하는 CreateZone() 함수를 호출한다.

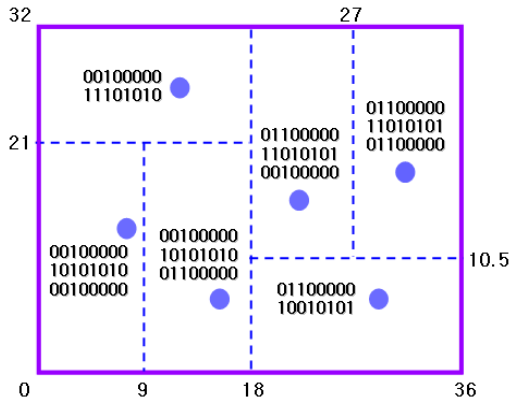
3.2 Zone 생성

NUNS는 각 센서 노드의 처리 영역을 할당하기 위해 각 Partition을 비균등 크기의 Zone으로 분할한다. 임의의 초기 Partition을 P_0(X-Y 평면)라고 할 때, Zone은 Partition P_0를 센서 노드 개수와 분할된 영역 크기의 차이가 최소가 되도록 X축과 Y축을 번갈아 가면서 한 개의 센서 노드가 남을 때까지 비균등하게 계속 분할하여 얻어진 사각형이다.

분할 과정 중에서 특정 Partition P_i ($0 \le i \le n$)에 있는 센서 노드 개수가 짝수 개이면 분할된 두 Partition P_{i+1}과 $P_i'_{+1}$이 가지는 센서 노드 개수는 같아지지만 센서 노드 개수가 홀수 개이면 두 Partition이 갖는 센서 노드 개수는 같아질 수 없다. 이때는 분할된 두 Partition 중 크기가 더 큰 Partition에 센서 노드 개수가 한 개 더 많도록 한다. 최종적으로 초기 Partition P_0의 모든 Zone들

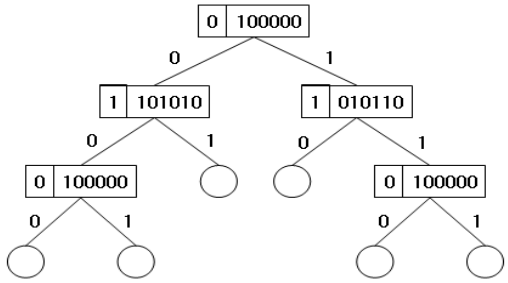
이 각자 한 개의 센서 노드를 포함하게 되고 Partition P_0내에 존재하는 Zone들의 개수는 Partition P_0 내의 센서 노드 개수와 같아질 때까지 이러한 과정을 X축과 Y축을 번갈아 가면서 계속 수행한다. 이처럼 Zone 생성은 Partition 생성과 유사하지만 Zone은 Partition과 달리 센서 노드 개수만큼 생성되고, 모든 Zone은 센서 노드 한 개를 포함하게 된다.

NUNS에서 Zone은 Partition 코드와 같이 서로 다른 Zone과 구별되는 Zone 코드를 가진다. Zone 코드는 Partition 코드와 마찬가지로 비트 스트링 형태이며 분할 축, Zone 구분자, 분할 축의 위치로 구성된다. <그림 4>는 8비트 Zone 코드를 사용하여 <그림 1(b)>에서 나타난 4개의 Partition 중 좌측 하단 Partition에서 Zone을 생성한 예를 보여준다.



<그림 4> Zone 생성 예

<그림 4>에서 점선으로 된 6개의 사각형이 Zone을 나타낸다. <그림 4>에서 보는 것과 같이 Partition은 센서 노드 개수만큼 Zone을 가지게 된다. <그림 5>는 <그림 4>의 Zone 코드에 대한 K-D 트리 형태의 Zone 트리를 보여준다.



<그림 5> Zone 트리

<그림 5>에서 보는 것과 같이 Zone 트리의 각 노드는 분할 축과 분할 축의 위치를 가지고 있다. 데이타 저장이나 질의 처리 시에 특정 데이타 값에 해당하는 Zone 코드는 Zone 트리를 루트 노드부터 리프 노드까지 탐색하면서 구할 수 있다. Zone 트리 탐색 시 만일 분할 축에 따라 분할된 Zone이 좌측(하단)이면 Zone 구분자는 비트 0이 되고 좌측 하위 노드로 내려간다. 만일 분할된 Zone이 우측(상단)이면 Zone 구분자는 비트 1이 되고 우측 하위 노드로 내려간다. Partition 코드로 Partition의 대각선 좌표를 구하는 것과 마찬가지로 Zone 코드를 통해 Zone의 대각선 좌표를 구할 수 있다. NUNS에서 Zone을 생성하는 알고리즘은 <그림 6>과 같다.

```
Algorithm : CreateZone(zone, axis)
 1: Begin
 2:   loc = null; zArray[2] = {null, null};
 3:   loc = FindSplitpoint(zone, axis);
 4:   SplitZone(zone, zArray, axis, loc);
 5:   UpdateZoneCode(zArray, axis, loc);
 6:   if(axis) then
 7:     axis = 0;
 8:   else
 9:     axis = 1;
10:   end if
11:   for i from 0 to 1 do
12:     if(CountSensor(zArray[i].area)>1) then
13:       CreateZone(zArray[i].area, axis);
14:     else
15:       SaveZone(zArray[i]);
16:     end if
17:   end for
18: End
```

<그림 6> Zone 생성 알고리즘

<그림 6>의 Zone 생성 알고리즘에서 입력 인자인 *zone*은 분할할 Zone이고, *axis*는 분할 축을 나타낸다. 라인 2는 분할 축의 위치를 저장할 변수 *loc*과 분할된 Zone 정보를 저장할 구조체 *zArray* 배열을 초기화한다. 라인 3은 *zone*에 대해 센서 노드 개수와 분할된 영역 크기 차이가 최소가 되는 분할 축의 위치를 구하여 변수 *loc*에 저장하고, 라인 4는 분할 축, 분할 축의 위치를 이용하여 *zone*을 비균등하게 분할하고 분할된 Zone 정보를 구조체 *zArray* 배열에 저장한다. 라인 5는 분할된 *zone*에 대한 Zone 코드를 생성하고, 라인 6부터 라인 10까지는 *zone*을 X축과 Y축을 번갈아 분할하기 위해 현재 사용된 분할 축을 다음 분할 축으로 변경한다. 라인 11부터 라인 17까지는 *zone*을 분할하여 얻어진 두 Zone에 대하여 각각 Zone 내에 존재하는 센서 노드 개수가 1보다 크면 Zone 분할을 반복 수행하고, 그렇지 않으면 Zone에 대한 분할을 종료하고 Zone을 저장한다.

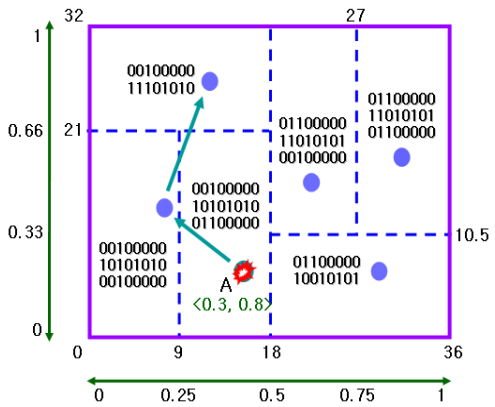
이처럼 NUNS에서의 Partition 및 Zone 생성은 네트워크 분할시 균등 분할을 수행하는 DIM에 비해 비균등 분할 정보 저장에 대한 오버헤드가 발생하지만, 센서 노드는 자신이 포함되는 Partition 내의 Zone 정보만을 갖기 때문에 인덱스 저장 및 관리에 따르는 공간 오버헤드를 줄일 수 있다. 그리고, NUNS에서는 네트워크의 비균등 분할이 주기적으로 수행되므로 센서 노드의 추가, 삭제시에도 센서 네트워크는 정상적으로 동작될 수 있다.

3.3 데이타 저장

NUNS에서는 한 Partition 내에서 측정된 데이타를 그 Partition 내의 센서 노드가 저장 및 관리한다. 일단 센서 노드가 데이타를 측정하면 센서 노드는 측정된 데이타를 저장할 센서 노드를 찾기 위해 데이타 값을 해싱하여 Zone 코드를 생성하고, 생성된 Zone 코드에 해당하는 Zone 내에 있는 센서 노드를 찾아 데이타를 저장한다. 이때, NUNS에서 Zone 코드는 Zone 트리의 루트 노드에서 리프 노드까지 탐색하면서 노드에 저장되어 있는 분할 축, 분할 축의 위치와 측정된 데이타 값을 비교하여 얻을 수 있다.

예를 들어, 데이타가 가지는 속성이 두 개이고 두 속성이 가질 수 있는 값은 0과 1사이 값으로 정형화된다고 가정하자. 그러면 NUNS에서는 Partition의 X, Y축이 가지는 값의 범위와 측정하는 데이타 속성이 가지는 값의 범위가 서로 매핑되므로 한 Partition의 X, Y축에 매핑되는 값은 0과 1사이의 값이 된다. 이러한 경우에 <그림 1(b)>에서 나타난 4개의 Partition 중 좌측 하단 Partition에서 센서 노드 A가 측정된 데이타가 <0.3, 0.8>이면 측정된 데이타에 대한 Zone 코드 생성은 다음과 같다.

먼저 루트 노드에서 측정된 데이타의 첫 번째 속성값 0.3이 분할 축의 위치에 해당하는 매핑값인 0.5보다 작기 때문에 좌측 자식 노드로 내려간다. 그리고 측정된 데이타의 두 번째 속성값 0.8이 분할 축의 위치에 해당하는 매핑값인 0.66보다 크기 때문에 우측 자식 노드로 내려온다. 이때, 접근된 하위 노드가 리프 노드이기 때문에 Zone 코드 생성이 종료된다. 생성된 Zone 코드는 (00100000 11101010)이고, 이 Zone 코드가 나타내는 Zone 내에 포함되어 있는 센서 노드에 데이타를 저장한다. <그림 7>은 이러한 데이타 저장 예를 보여주는데, 좌표축과 평행인 두 선분 상의 숫자는 속성이 가질 수 있는 값을 0과 1사이로 정형화한 것을 보여준다.



<그림 7> 데이타 저장 예

<그림 7>에서 보는 것과 같이 한 Partition에서 측정된 데이타는 그 Partition 내의 센서 노드에 저장되기 때문에 센서 네트워크에서 같은 값을 가지는 데이타가 빈번히 발생하는 경우에 하나의 센서 노드에 데이타 저장이 집중되는 것을 줄일 수 있다. 그리고 데이타를 측정한 센서 노드와 데이타를 저장하는 센서 노드의 거리가 가까워지면서 데이타 저장을 위한 통신비용도 줄일 수 있다. 또한, NUNS에서는 모든 Zone이 한 개의 센서 노드를 포함하고 있기 때문에 DIM에서와 같이 Zone에 센서 노드가 없어서 발생하는 불필요한 라우팅 비용을 줄일 수 있다. NUNS에서 데이타를 저장하는 알고리즘은 <그림 8>과 같다.

```
Algorithm : StoreData(loc, data)
 1: Begin
 2:   dc = null; n = 0; dl = loc; tl = null;
 3:   dc = Hash(data);
 4:   if(ContainSensor(dc, loc)) then
 5:     SaveData(data);
 6:   else
 7:     n = CountNeighbor(loc);
 8:     for i from 0 to n do
 9:       tl = GetLocation(i);
10:       if(IsNearest(dc, tl, dl)) then
11:         dl = tl;
12:       end if
13:     end for
14:     if(dl == loc) then
15:       dl = RightHandRouting(loc, dc);
16:     end if
17:     StoreData(dl, data);
18:   end if
19: End
```

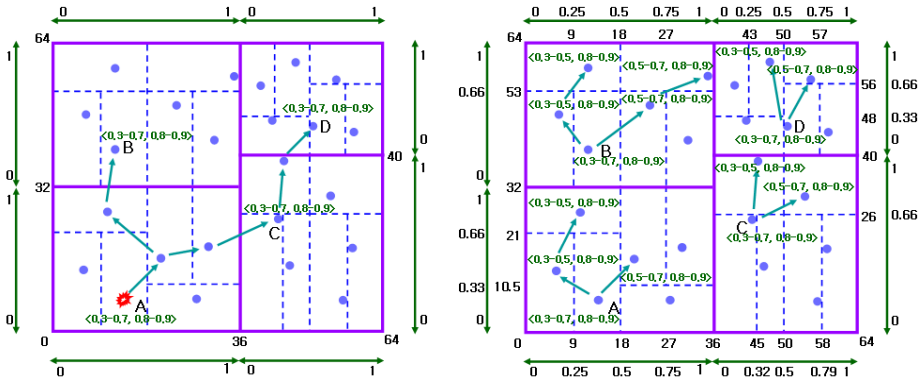
<그림 8> 데이타 저장 알고리즘

<그림 8>의 데이타 저장 알고리즘에서 입력 인자 *loc*은 데이타를 생성한 센서 노드의 좌표이고 *data*는 저장할 데이타이다. *data*는 하나 이상의 속성 데이타를 포함한다. 라인 3은 *data*를 해싱하여 목적 Zone의 Zone 코드를 생성한다. 라인 4와 라인 5는 생성된 Zone 코드에 해당하는 목적 Zone이 현재 데이타를 전송하고 있는 센서 노드를 포함하는지 검사하고, 만일 포함하면 그 센서 노드가 데이타를 저장할 목적 센서 노드이므로 데이타를 저장하고 알고리즘을 종료한다. 만약 포함하지 않으면 라인 7에서 목적 Zone으로 데이타를 전달할 센서 노드를 찾기 위해 이웃 센서 노드 개수를 가져오고, 라인 8부터 라인 13까지는 이웃 센서 노드들 중 목적 Zone과 가장 가까운 센서 노드의 위치를 찾아 변수 *dl*에 저장한다. 라인 14부터 라인 16까지는 만일 목적 Zone과 가장 가까운 센서 노드를 찾지 못하면 라우팅 경로를 오른쪽으로 우회하여 만나는 센서 노드 위치를 찾아 변수 *dl*에 저장한다. 라인 17에서는 변수 *dl*에 저장된 센서 노드와 전송할 데이타를 가지고 다시 StoreData() 함수를 호출한다.

3.4 질의 처리

NUNS에서는 한 Partition 내에서 측정된 데이타를 그 Partition 내의 센서 노드가 저장 및 관리하기 때문에 임의의 센서 노드가 질의를 발생하면 질의 결과를 얻기 위해 센서 네트워크의 모든 Partition 내에서 질의를 처리할 목적 센서 노드에게 질의를 전달해야 한다. 그러므로 질의를 발생한 센서 노드는 모든 Partition으로 질의를 전달하기 위해 모든 Partition의 위치를 알아야 하는데, Partition의 위치는 Partition 코드를 통해 알 수 있으며 Partition 코드는 전체 센서 노드가 가지고 있는 Partition 트리를 통해 얻을 수 있다. 이렇게 Partition의 위치를 가지고 각 Partition까지 도달하면 그 Partition 내에서 질의를 처리할 목적 센서 노드에게 질의를 전달해야 한다. <그림 9>는 <그림 1(b)>의 4개의 Partition에서 질의를 전달하는 예를 보여주는데, 좌표축과 평행인 선분들 상의 숫자는 Partition 마다 속성이 가질 수 있는 값을 0과 1사이로 정형화한 것을 보여준다.

<그림 9(a)>에서 보는 것과 같이 센서 노드 A는 모든 Partition의 위치를 가지고 질의 <0.3−0.7, 0.8−0.9>를 전달한다. 센서 노드 A가 전달한 질의는 각 Partition 내에서 질의를 처음 전달받게 되는 센서 노드인 B, C, D에 전달된다. 그리고 <그림 9(b)>에서 보는 것과 같이 센서 노드 A, B, C, D는 각각 전달받은 질의 <0.3−0.7, 0.8−0.9>를 해싱하여 Zone 코드를 생성하고 생성된 Zone 코



(a) Partition으로 질의 전달 (b) 목적 센서 노드로 질의 전달

<그림 9> 질의 전달 예

드에 해당하는 Zone 내에 있는 목적 센서 노드를 찾아 질의를 전달한다. 해싱 과정에서 질의는 각 Zone이 가지는 데이타 범위에 따라 분해되고, 분해된 질의마다 Zone 코드가 생성되며, 생성된 각 Zone 코드에 해당하는 Zone 내에 존재하는 센서 노드에게 분해된 질의가 전달된다. 이때, Zone 코드는 Zone 트리의 루트 노드에서 리프 노드까지 탐색하면서 노드에 저장되어 있는 분할 축, 분할 축의 위치와 질의에 사용된 데이타의 속성값을 비교하여 얻을 수 있다.

예를 들어, <그림 9(b)>에서 나타난 4개의 Partition 중 좌측 하단 Partition에서 센서 노드 A가 발생한 질의 <0.3−0.7, 0.8−0.9>에 대한 Zone 코드 생성은 다음과 같다. 먼저 루트 노드에서 첫 번째 속성값이 <0.3−0.7>이므로 분할 축의 위치에 해당하는 매핑값인 0.5에 의해 질의가 <0.3−0.5, 0.8−0.9>, <0.5−0.7, 0.8−0.9>로 분할되고, 각각 좌측과 우측 하위 노드로 전달된다. 먼저 좌측 하위 노드에 전달된 질의는 두 번째 속성값이 <0.8−0.9>이므로 분할 축의 위치에 해당하는 매핑값인 0.66보다 크기 때문에 우축 하위 노드로 내려온다. 이때, 우측 하위 노드는 리프 노드이므로 생성되는 Zone 코드는 (00100000 11101010)이 된다. 다음으로 우측 하위 노드에 전달된 질의는 두 번째 속성값이 <0.8−0.9>이므로 분할 축의 위치에 해당하는 매핑값인 0.33보다 크기 때문에 다시 우측 하위 노드로 내려온다. 우측 하위 노드로 내려온 질의는 첫 번째 속성값이 <0.5−0.7>이므로 분할 축의 위치에 해당하는 매핑값인 0.75보다 작기 때문에 좌측 하위 노드로 내려온다. 이때, 좌측 하위 노드는 리프 노드이므로 생성되는 Zone 코드는 (01100000 11010101 00100000)이 된다. 결과적으로 질의는 2개의 하위 질의로 분해되며 각 하위 질의의 Zone 코드는 (00100000 11101010), (01100000 11010101 00100000)이 된다.

이처럼 NUNS는 센서 네트워크에서 측정된 데이타를 Partition 별로 분산 저장하기 때문에 질의 결과를 전달하는 센서 노드의 통신비용을 다소 줄일 수 있으나 질의가 모든 Partition으로 전달되고 결과를 전달받아야 하기 때문에 결과적으로 질의 처리시 센서 네트워크의 통신비용은 증가하게 된다. NUNS에서 <그림 9(a)>와 같이 전체 Partition으로 질의를 전달하고 결과를 처리하는 알고리즘은 <그림 10>과 같다.

<그림 10>의 질의 처리 알고리즘에서 입력 인자 *loc*은 질의를 전달하는 센서 노드의 좌표이고, *query*는 질의이고, *count*는 Partition의 개수이다. 라인 2는 알고리즘에서 사용할 변수를 초기화한다. 라인 3부터 라인 19까지는 Partition 개수만큼 해당 Partition으로 질의를 전달한다. 라인 4는 질의를 전달할 Partition을 찾기 위해 Partition 코드를

```
Algorithm : ProcessPQuery(loc, query, count)
 1: Begin
 2:    dc = null; n = 0; tl = null; pc = count; dl = loc;
 3:    for i from 0 to pc do
 4:       dc = Hash(pArray[i]);
 5:       while(!EqualCode(dc, dl)) do
 6:          n = CountNeighbor(loc);
 7:          for j from 0 to n do
 8:             tl = GetNeighbor(j);
 9:             if(IsNearest(dc, tl, dl)) then
10:                dl = tl;
11:             end if
12:          end for
13:          if(dl == loc) then
14:             dl = RightRoutingQuery(loc, dc);
15:          end if
16:       end while
17:       ProcessZQuery(dl, query);
18:       ReturnResult(query);
19:    end for
20: End
```

<그림 10> 질의 처리 알고리즘 1

```
Algorithm : ProcessZQuery(loc, query)
 1: Begin
 2:    dc = null; n = 0; tl = null; dl = loc;
 3:    c = SplitQuery(query);
 4:    for i from 0 to c do
 5:       dc = Hash(qArray[i]);
 6:       if(Contain(dc, loc)) then
 7:          ReturnResult(qArray[i]);
 8:       else
 9:          n = CountNeighbor(loc);
10:          for j from 0 to n do
11:             tl = GetNeighbor(j);
12:             if(IsNearest(dc, tl, dl)) then
13:                dl = tl;
14:             end if
15:          end for
16:          if(dl == loc) then
17:             dl = RightRoutingQuery(loc, dc);
18:          end if
19:          ProcessZQuery(dl, qArray[i]);
20:       end if
21:    end for
22: End
```

<그림 11> 질의 처리 알고리즘 2

생성한다. 라인 5부터 라인 16까지는 생성된 Partition 코드에 해당하는 Partition이 질의하는 센서 노드를 포함하는지 검사한다. 만약 Partition이 질의하는 센서 노드를 포함하지 않으면 라인 6에서 질의를 전달할 센서 노드를 찾기 위해 이웃 센서 노드 개수를 가져온다. 라인 7부터 라인 12까지는 이웃 센서 노드들 중 최종 전송할 Partition과 가장 가까운 센서 노드 위치를 찾아 변수 *dl*에 저장한다. 만일 전송할 Partition의 가장 가까운 센서 노드를 찾지 못하면 라인 13부터 라인 15까지에서 라우팅 경로를 오른쪽으로 우회하여 만나는 센서 노드 위치를 찾아 변수 *dl*에 저장한다. 라인 17에서는 변수 *dl*에 저장된 센서 노드와 전송할 데이타를 가지고 해당 Partition에서 목적 센서 노드를 찾기 위해 ProcessZQuery() 함수를 호출한다.

NUNS에서 <그림 9(b)>와 같이 Partition 내에서 목적 센서 노드로 질의를 전달하고 처리하는 알고리즘은 <그림 11>과 같다.

<그림 11>의 질의 처리 알고리즘에서 입력 인자 *loc*은 질의하는 센서 노드의 좌표이고, *query*는 질의이다. 라인 2는 알고리즘에서 사용하는 변수를 초기화한다. 라인 3은 질의의 속성값 범위가 여러 Zone을 포함할 경우 해당되는 각 Partition의 Zone에 따라 질의를 분할한다. 라인 4부터 라인 21까지는 분할된 질의 개수만큼 질의를 처리한다. 라인 5는 분할된 질의에 대한 최종 전달할 Zone을 찾기 위해 데이타를 해싱하여 Zone 코드를 생성한다. 라인 6과 라인 7은 생성된 Zone 코드에 해당하는 Zone이 질의하는 센서 노드를 포함하는지 검사하고 만일 포함하면 현재 센서 노드에서 질의를 처리한다. 만약 포함하지 않으면 라인 9에서 질의를 전달할 센서 노드를 찾기 위해 이웃 센서 노드 개수를 가져온다. 라인 10부터 라인 15까지는 이웃 센서 노드들 중 최종 전송할 Zone과 가장 가까운 센서 노드 위치를 찾아 변수 *dl*에 저장한다. 라인 16부터 라인 18까지는 만일 최종 전송할 Zone과 가장 가까운 센서 노드를 찾지 못하면 라우팅 경로를 오른쪽으로 우회하여 만나는 센서 노드를 찾아 변수 *dl*에 저장한다. 라인 19는 변수 *dl*에 저장된 센서 노드와 전송할 질의를 가지고 다시 ProcessZQuery() 함수를 호출한다.

4. 성능 평가

본 장에서는 데이타 중심 센서 네트워크에서 GHT나 DIFS 보다 항상 데이타 저장 및 질의 처리

성능이 우수한 DIM과 본 논문에서 제시한 NUNS 간의 성능을 비교 평가하였다. 성능 평가에서 사용된 시스템 사양은 Intel Core2 CPU 2.13GHz, 2GB RAM이고, 운영체제는 Windows XP Professional을 사용하였다.

4.1 데이타 저장 비용

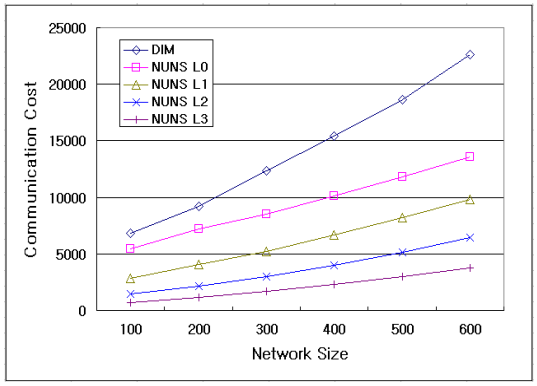
센서 네트워크에서 측정된 데이타를 저장할 때 발생되는 통신비용에 대해 NUNS와 DIM을 비교하였다. NUNS에서 Partition의 개수는 레벨에 따라 다르다. 여기서 레벨은 Partition의 분할 횟수와 같은 의미이다. 예를 들어, 레벨이 k인 NUNS는 $2^k(k \geq 0)$ 개의 Partition을 갖는다. 실험 조건으로 센서 노드의 통신 범위를 50m로 설정하고, 센서 노드를 100개에서 600개까지 증가시키면서 두 개의 속성을 갖는 데이타 1,000개를 랜덤으로 생성하였다. <그림 12>와 <그림 13>은 각각 센서 네트워크의 데이타 저장 비용과 Hotspot의 데이타 저장 비용을 보여준다. 여기서 Hotspot은 센서 네트워크에서 에너지 소모가 가장 높은 센서 노드를 말한다. 여기서 NUNS L0, NUNS L1, NUNS, L2, NUNS L3은 각각 분할을 0, 1, 2, 3번을 수행한 것을 의미하며, 이들은 각각 Partition 개수를 1, 2, 4, 8개를 가지게 된다.

<그림 12>에서 보는 것과 같이 센서 네트워크 크기가 커질수록 Partition 개수가 많은 NUNS에서의 데이타 저장 비용이 DIM 보다 효율적으로 나타났다. 이는 NUNS가 한 Partition 내에서 측정된 데이타를 그 Partition 내의 센서 노드에 저장함으로써 데이타를 측정한 센서 노드와 데이타를 저장하는 센서 노드 사이의 거리가 짧아져 데이타 저장 시 통신 횟수가 줄어들고, Partition을 한 개의 센서 노드를 포함하는 Zone으로 비균등하게 분할함으로써 DIM에서 목적 Zone에 센서 노드가 없어서 발생하는 불필요한 라우팅 비용을 줄였기 때문이다.

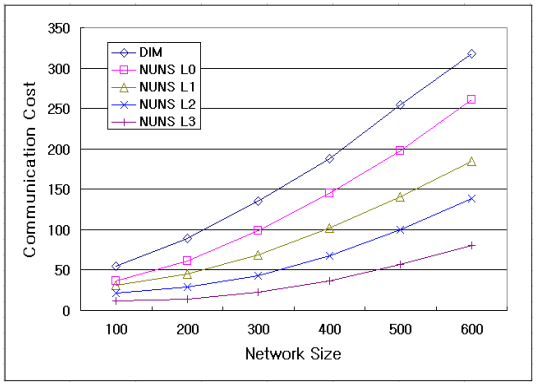
<그림 13>의 Hotspot 데이타 저장 비용도 <그림 12>의 센서 네트워크 데이타 저장 비용과 유사하게 센서 네트워크 크기가 커질수록 Partition 개수가 많은 NUNS의 데이타 저장 비용이 DIM보다 효율적으로 나타났다. 이는 센서 네트워크에서 측정되는 데이타가 Partition 별로 분산 저장되어 Hotspot의 저장 부하가 줄어들고, 각 Partition은 분할 영역 크기 차이를 최소가 되도록 센서 노드의 처리 영역인 Zone으로 분할됨으로써 처리 영역이 큰 센서 노드에 부하가 치우치는 것을 방지하였기 때문이다.

4.2 질의 처리 비용

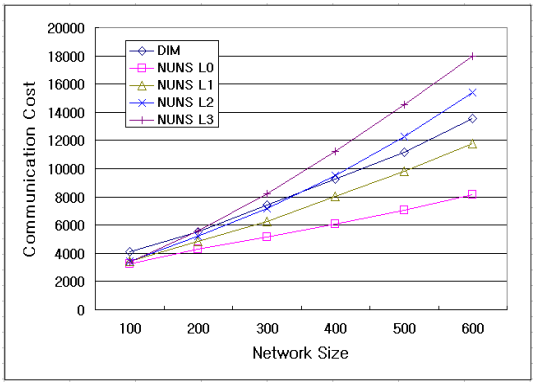
센서 네트워크에서 질의를 처리할 때 발생되는



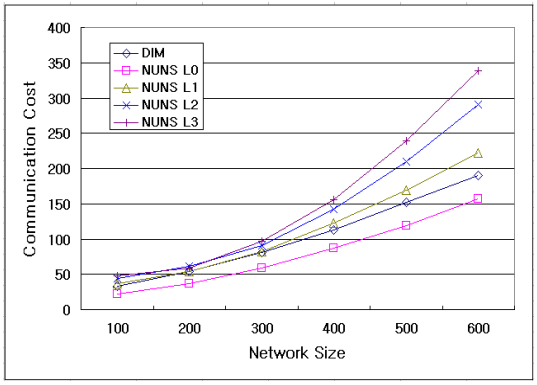
<그림 12> 센서 네트워크의 데이타 저장 비용



<그림 13> Hotspot의 데이타 저장 비용



<그림 14> 센서 네트워크의 질의 처리 비용



<그림 15> Hotspot의 질의 처리 비용

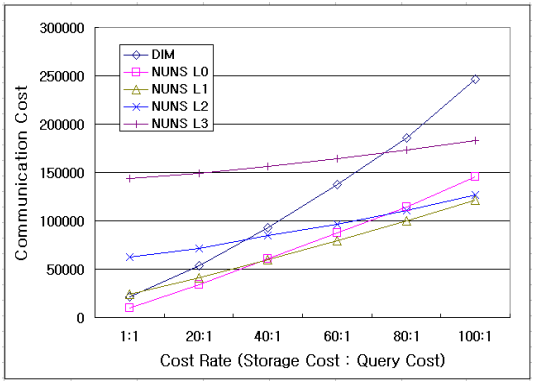
통신비용에 대해 NUNS와 DIM을 비교하였다. 실험 조건으로 센서 노드의 통신 범위를 50m로 설정하고, 센서 노드를 100개에서 600개까지 증가시키면서 두 개의 속성을 갖는 범위 질의 100개를 랜덤으로 생성하였다. 이때, 질의가 가지는 속성값의 범위는 속성이 가질 수 있는 최대 범위의 10% 이내로 하였다. <그림 14>와 <그림 15>는 각각 센서 네트워크의 질의 처리 비용과 Hotspot의 질의 처리 비용을 보여준다.

<그림 14>에서 보는 것과 같이 센서 네트워크 크기가 커질수록 Partition 개수가 1개, 2개인 NUNS의 질의 처리 비용이 각각 DIM보다 최대 40%, 15%가 효율적이었지만, Partition 개수가 4개, 8개인 NUNS의 질의 처리 비용은 DIM보다 높아지는 것으로 나타났다. 이는 NUNS가 센서 노드의 저장 부하를 줄임으로써 목적 센서 노드의 질의 결과 전송 비용을 줄였지만 질의를 모든 Partition의 목적 센서 노드로 전달해야 하기 때문에 전달할 Partition 개수가 많아질수록 질의 전달을 위한 통신비용이 높아지기 때문이다.

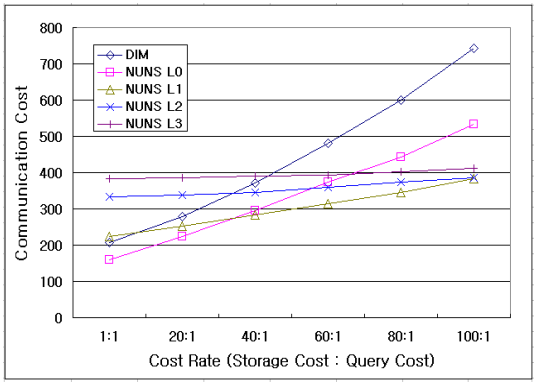
<그림 15>의 Hotspot 질의 처리 비용도 <그림 14>과 유사하게 센서 네트워크 크기가 커질수록 Partition 개수가 1개인 NUNS의 질의 비용이 DIM보다 효율적이었지만, Partition 개수가 2개, 4개, 8개인 NUNS의 질의 처리 비용은 DIM보다 높아지는 것으로 나타났다. 이는 NUNS가 Hotspot의 저장 부하를 줄임으로써 결과적으로 Hotspot의 질의 결과 전송 비용을 줄였지만 질의를 모든 Partition의 목적 센서 노드로 전달해야 하기 때문에 전달할 Partition 개수가 많아질수록 질의 전달을 위한 통신비용이 높아지기 때문이다. 따라서 데이타 중심 센서 네트워크의 에너지 효율성을 위해서는 데이타 저장 횟수와 질의 처리 횟수의 비율을 고려하여 효율적인 Partition 개수를 설정하는 것이 필요하다.

4.3 데이타 저장과 질의 처리 비율에 따른 비용

본 절에서는 데이타 저장 횟수와 질의 처리 횟수 비율에 따른 센서 네트워크의 통신비용과 Hotspot의 통신비용을 실험하였다. 즉, 센서 네트워크에서 데이타 저장 횟수와 질의 처리 횟수 비율을 1:1에서 100:1까지 변경시키면서 발생되는 통신비용에 대해 NUNS와 DIM을 비교하였다. 실험 조건으로 센서 노드의 통신 범위를 50m로 설정하고, 센서 노드는 600개를 사용하였다. 그리고 두 개의 속성을 갖는 데이타를 100개에서 10,000개까지 랜덤으로 생성하고 두 개의 속성을 갖는 범위 질의 100개를 랜덤으로 생성하였다. 이때, 질의가 가지는 속성값의 범위는 속성이 가질 수 있는 최대 범위의 10% 이내로 하였다. <그림 16>과 <그림 17>은 각각 센서 네트워크의 통신비용과 Hotspot의 통신



<그림 16> 센서 네트워크의 통신비용



<그림 17> Hotspot의 통신비용

비용을 보여준다.

<그림 16>에서 보는 것과 같이 센서 네트워크의 통신비용 측면에서 데이타 저장 횟수와 질의 처리 횟수 비율이 약 40:1까지는 Partition 개수가 1개인 NUNS가 가장 효율적이고, 데이타 저장과 질의 발생 비율이 약 40:1부터 120:1까지 Partition 개수가 2개인 NUNS가 가장 효율적인 것으로 나타났다. 그리고 데이타 저장 횟수가 질의 처리 횟수보다 더 많아질수록 Partition 개수가 많은 NUNS의 통신비용 효율성이 높아질 것으로 예상할 수 있다. 이는 NUNS에서 Partition 개수가 증가함에 따라 데이타 저장 비용은 감소되고 반대로 질의 처리 비용은 증가되기 때문에 데이타 저장 횟수가 질의 처리 횟수보다 많아질수록 데이타 저장 비용 감소폭이 질의 처리 비용 증가폭보다 상대적으로 커지기 때문이다. <그림 17>에서 보는 것과 같이 Hotspot의 통신비용도 <그림 16>의 센서 네트워크의 통신비용과 유사하게 데이타 저장 횟수가 질의 처리 횟수보다 많을수록 Partition 개수가 큰 NUNS가 효율적인 것으로 나타났다.

위 실험 결과를 통해 데이타 중심 센서 네트워크에서 Partition 개수가 1개인 NUNS가 모든 경우에서 DIM보다 에너지 효율성이 우수하다는 것을 알 수 있었고, 또한 NUNS의 Partition 개수가 많을수록 질의 처리 비용은 증가하였지만 데이타 저장 비용을 줄일 수 있다는 것을 알 수 있었다. 따라서 데이타 저장이 빈번한 데이타 중심 센서 네트워크에서는 NUNS를 통해 에너지 효율성을 높일 수 있을 것이다.

5. 결론

데이타 중심 센서 네트워크에서는 측정된 데이타의 값에 따라 데이타를 저장되는 센서 노드가 결정되기 때문에 같은 값의 데이타가 빈번하게 발생하면 이를 저장하는 센서 노드에 부하가 집중되어 빠르게 에너지가 고갈되는 문제가 발생한다. 그리고 새로운 센서 노드의 추가로 센서 네트워크가 확장되면 데이타를 측정한 센서 노드와 데이타를 저장하는 센서 노드간의 거리가 멀어지면서 데이타 저장 및 질의 처리시 통신비용이 증가하는 문제가 있다. 그러므로 데이타 중심 센서 네트워크에서는 센서 노드의 부하를 효율적으로 분산시키고 센서 네트워크의 확장에 따른 비용을 줄여 센서 네트워크의 에너지 효율성을 높이는 것이 중요하다.

이와 같은 문제를 해결하기 위해 본 논문에서는 데이타 중심 센서 네트워크에서 센서 노드의 부하를 분산시키고 센서 네트워크의 확장에 따른 비용을 줄이는 NUNS를 제안하였다. NUNS는 센서 노드의 부하를 분산시키고 센서 네트워크 확장에 따른 비용을 줄이기 위해 센서 네트워크를 비균등 크기의 Partition으로 분할하고, 각 Partition 내에서

발생한 데이타를 각 Partition 내의 센서 노드가 저장 및 관리하도록 하였다. 그리고 NUNS에서는 센서 노드의 부하 집중을 막고 불필요한 라우팅 비용을 줄이기 위해 각 Partition을 분할된 영역 크기의 차이가 최소가 되도록 센서 노드 개수만큼 비균등 크기의 Zone으로 분할하였다. 마지막으로 실험을 통해 데이타 저장이 빈번한 데이타 중심 센서 네트워크에서 NUNS가 DIM보다 에너지 효율성이 우수함을 입증하였다.

참고문헌

1. Draper. C., and Wornell, G., "Side Information Aware Coding Strategies for Sensor Networks," IEEE Journal on Selected Areas in Communications, Vol.22, No.6, 2004, pp.966-976.
2. Karp, B., and Kung, T., "GPSR: Greedy Perimeter Stateless Routing for Wireless Networks," Proc. of the 6th Annual International Conference on Mobile Computing and Networking, 2000, pp.243-254.
3. Madden, S., Franklin, M., Hellerstein, J., and Hong, W., "TAG: A Tiny Aggregation Service for Ad-hoc Sensor Networks," Proc. of the 5th Symposium on Operating System Design and Implementation, 2002, pp.131-146.
4. Li, X., Kim, Y., Govindan, R., and Hong, W., "Multi-Dimensional Range Queries in Sensor Networks," Proc. of the 1st International Conference on Embedded Networked Sensor Systems, 2003, pp.63-75.
5. Mainwaring, A., Polastre, J., Szewczyk, R., Culler, D., and Anderson, J., "Wireless Sensor Networks for Habitat Monitoring," Proc. of the 1st ACM International Workshop on Wireless Sensor Networks and Applications, 2002, pp.88-97.
6. 강홍구, 전상훈, 홍동숙, 한기준, "데이타 중심 저장 방식의 센서 네트워크를 위한 비균등 영역 분할 기법," 한국공간정보시스템학회 논문지, 8권3호, 2006, pp.105-115.
7. Ratnasamy, S., Karp, B., Yin, L., Yu, F., Estrin, D., Govindan, R., and Shenker, S., "GHT: A Geographic Hash Table for Data-Centric Storage in Sensornets," Proc. of the ACM Workshop on Sensor Networks and Applications, 2002, pp.78-87.
8. Ratnasamy, S., Estrin, D., Govindan, R., Karp, B., and Shenker, S., "Data-Centric Storage in Sensornets," Proc. of the 1st ACM SIGCOMM Workshop on Hot Topics in Networks, Vol.33, No.1, 2003, pp.137-142.
9. Ratnasamy, S., Karp, B., Shenker, S., Estrin, D., Govindan, R., Yin, L., and Yu, F., "Data-Centric Storage in Sensornets with GHT, A Geographic Hash Table," Proc. of the Mobile Networks and Applications, Vol.8, No.4, 2003, pp.427-442.
10. Greenstein, B., Estrin, D., Govindan, R., Ratnasamy, S., and Shenker, S., "DIFS: A Distributed Index for Features in Sensor Networks," Proc. of the 1st IEEE International Workshop on Sensor Network Protocols and Applications, 2003, pp.163-173.
11. Sharifzadeh, M., and Shahabi, C., "Supporting Spatial Aggregation in Sensor Network Databases," Proc. of the 12th ACM International Symposium on Advances in Geographic Information Systems, 2004, pp.166-175.
12. 이해준, 고양우, 이동만, "센서 네트워크를 위한 홉간 순서 번호를 이용하는 신뢰성 있는 데이타 전송 지원 기법," 한국정보과학회 학술발

표 논문집, 32권2호, 2005, pp.178-180.
13. 임용훈, 정연돈, 김명호, "데이타 기반 센서 네트워크에서 다차원 영역 질의를 위한 동적 데이타 분산," 한국정보과학회 논문지, 33권1호, 2006, pp.32-41.
14. 강홍구, 김정준, 홍동숙, 한기준, "센서 네트워크에서 동적 영역 분할을 이용한 다차원 범위 질의 인덱스," 한국정보과학회 학술발표 논문집, 31권2호, 2006, pp.52-54.

강홍구
2002년 건국대학교 컴퓨터공학과(공학사)
2004년 건국대학교 대학원 컴퓨터공학과(공학석사)
2004년~현재 건국대학교 대학원 컴퓨터공학과 박사과정
관심분야 : 공간 데이타베이스, GIS, LBS, USN, 센서 데이타베이스

김정준
2003년 건국대학교 컴퓨터공학과(공학사)
2005년 건국대학교 대학원 컴퓨터공학과(공학석사)
2005년~현재 건국대학교 대학원 컴퓨터공학과 박사과정
관심분야 : 공간 메인 메모리 데이타베이스, GIS, LBS, 텔레매틱스

한기준
1979년 서울대학교 수학교육학과(이학사)
1981년 한국과학기술원(KAIST) 전산학과(공학석사)
1985년 한국과학기술원(KAIST) 전산학과(공학박사)
1990년 Stanford 대학 전산학과 Visiting Scholar
1985년~현재 건국대학교 컴퓨터공학부 교수
2000년~2002년 한국정보과학회 데이터베이스연구회 운영위원장
2004년~2006년 한국공간정보시스템학회 회장
2004년~현재 한국정보시스템감리사협회 회장
관심분야 : 공간 데이타베이스, GIS, LBS, 텔레매틱스, 정보시스템 감리