International Journal of Wireless Information Networks, Vol. 13, No. 4, October 2006 (© 2006)
DOI: 10.1007/s10776-006-0037-y

Attribute Allocation and Retrieval Scheme for Large-Scale Sensor Networks

Ratnabali Biswas,[1,2] Kaushik Chowdhury[1] and Dharma P. Agrawal[1]

Wireless sensor network is an emerging technology that enables remote monitoring of large geographical regions. In this paper, we address the problem of distributing attributes over such a large-scale sensor network so that the cost of data retrieval is minimized. The proposed scheme is a data-centric storage scheme where the attributes are distributed over the network depending on the correlations between them. The problem addressed here is similar to the *Allocation Problem* of distributed databases. In this paper, we have defined the Allocation Problem in the context of sensor networks and have proposed a scheme for finding a good distribution of attributes to the sensor network. We also propose an architecture for query processing given such a distribution of attributes. We analytically determine the conditions under which the proposed architecture is beneficial and present simulation results to demonstrate the same. To the best of our knowledge, this is the first attempt to determine an allocation of attributes over a sensor network based on the correlations between attributes.

KEY WORDS: Data-centric storage; allocation problem; storage node; control node; correlation tree; aggregation tree; hard threshold; soft threshold; update epoch

1. INTRODUCTION

Sensor networks are revolutionizing remote monitoring applications because of their ease of deployment, ad hoc connectivity and cost effectiveness. Such networks might be expected to serve multiple applications simultaneously. For example, given a geographical area, the user might want to deploy a sensor network that assists in ecosystem monitoring, weather monitoring, precision agriculture, etc. For each application the user would want the network to sense specific physical attributes (e.g. humidity, temperature, light, etc. for weather monitoring application; temperature, light, presence of chemicals etc. for precision agriculture application and so on.) and consequently would expect the sensor network to respond to some user-defined queries. Since such large-scale sensor networks would be expected to serve a substantial number of queries simultaneously for several applications, the number of attributes sensed by the network would also be substantial. Since a major resource constraint of sensor networks is the limited battery life of sensor nodes, protocols designed for sensor networks should minimize energy consumption of the network. At the node level, data communication is the dominant component of energy consumption, and hence sensor network protocols are geared towards reducing communication in the network [1].

The proposed Data-Centric Attribute Allocation and Retrieval (DCAAR) scheme is designed for minimizing the communication cost involved in data retrieval from large-scale sensor networks. The proposed scheme is a data-centric storage scheme where data is stored at a particular network location and queries for that data are directly routed to that location without flooding the network. As is true for all data-centric storage schemes [2], the proposed scheme is not always the method of choice, but is

[1] OBR Research Center for Distributed and Mobile Computing, Department of ECECS, University of Cincinnati, Cincinnati, OH 45221-0030, USA.
[2] E-mail: biswasr@ececs.uc.edu

preferable under certain conditions. Thus, using both analysis and simulation studies, we have determined the conditions under which the DCAAR scheme should be preferred. Future sensor networks are expected to support several protocols, with middleware software allowing a user to select the preferred protocol as per his requirements.

The rest of the paper is organized as follows. Section 2 lists some related research in the area of data storage in sensor networks. Section 3 defines the problem that this paper addresses, while Section 4 presents the proposed architecture to demonstrate how a distribution of attributes can be used to facilitate query processing in sensor networks. The methodology for determining a good distribution of attributes is presented in Section 5. Section 6 determines analytically the conditions under which the DCAAR scheme should be preferred. The simulation results are discussed in Section 7, while Section 8 concludes the paper.

2. RELATED WORK

There have been different approaches for storing data in sensor networks. Earlier sensor network systems stored sensor data externally at a remote base station (external storage) or locally at the nodes which generated them (local storage). Recently there has been a paradigm shift so that a technique called *data-centric storage* (DCS) now allows events to be stored at specific rendezvous points within the network that queries can access directly. Shenker et al. [2] proposed the DCS scheme and have shown that DCS outperforms other approaches such as external storage (ES) and local storage (LS) under certain circumstances. However, data-centric storage brings forth the challenge of storing the sensed data in a manner that assists in its retrieval for query processing. Furthermore, a query processing architecture needs to be defined for retrieving the stored data. Also, the data stored at specific rendezvous points needs to be updated in response to environmental changes. Thus, an efficient update mechanism needs to be designed such that the application quality-of-service requirements are maintained. Ratnasamy et al. [3] have proposed Geographic Hash Table (GHT) as a specific solution to achieve DCS in sensor networks. GHT hashes user-defined events (e.g. earthquakes, animal sightings) into geographic coordinates. GHT is inspired by Internet-scale Distributed Hash Table (DHT) systems such as Chord [4] and CAN [5]. Thus GHT hashes event names into geographic locations and stores the event at the sensor node closest to the hashed location. Greenstein et al. [6] have further extended the DCS architecture by designing DIFS, a spatially distributed index to efficiently support range queries (i.e. queries where only events with attributes in a certain range are desired). DIFS builds on top of GHT and constructs a multiply-rooted hierarchical index where nodes store event information for a particular range of values detected within a particular geographical region. Higher-level nodes cover smaller ranges detected within large geographic regions, while lower-level nodes cover wider range of values from within a smaller geographic region. In a similar work, Li et al. [7], have built a distributed index (DIM) for multidimensional data. DIM supports multi-dimensional range queries such as "List all events whose temperature lies between 50° and 60°, and whose light levels lie between 10 and 15″. In another work, Ghose et al. [8] have proposed Resilient Data-Centric Storage (R-DCS) as a method to achieve scalability and resilience by replicating data at strategic locations in the sensor network.

In this paper, we also propose a data-centric storage scheme for distributing attributes over a sensor network depending on the correlations between attributes and hence translate the allocation problem of distributed databases to the context of sensor networks. The proposed approach differs from the existing data-centric approaches in that it attempts to distribute attributes instead of specific user-defined events. We reason as follows. If an attribute is included in many events, then the values for that attribute have to be replicated and stored at different places in the network for each individual event. In our proposed approach, the attribute need not be replicated at multiple places and hence saves communication cost involved in storing and replicating data. Instead every attribute is stored at a predefined location within the network such that attributes that are a part of the same query are stored near each other to facilitate data retrieval. Thus it is similar to the Allocation Problem of distributed databases where a set of attributes need to be distributed over a number of sites, such that, in serving a predefined set of queries the communication between sites is minimized. In a similar manner, in this paper we present a scheme for storing attributes in the sensor network such that the communication involved in serving a predefined set of user queries can be minimized. Since all values of a particular attribute are stored at one geographical

location, the proposed architecture easily supports range queries. Also storing attributes belonging to the same query near each other, facilitates efficient querying of multidimensional data.

3. THE ALLOCATION PROBLEM

Sensor networks can be envisioned as a large distributed database where the sensor nodes generate named data against user-specified queries. Hence, the *Allocation Problem* of distributed databases can be redefined in the context of sensor networks as follows. *"Assume that there are a set of sensed attributes* $A = \{A_1, A_2, \ldots, A_m\}$ *and a network S of sensor nodes which have to serve a set of queries* $Q = \{Q_1, Q_2, \ldots, Q_q\}$. *The allocation problem involves in finding the optimal distribution of A to S."* Here, the optimality can be defined with respect to minimal energy consumption and hence minimal communication required in serving the set of queries Q. This is similar to the Allocation Problem of distributed databases which can be stated as "Given a relation R (i.e. a set of attributes), a set of queries Q, and a set of sites S where queries in Q run, the allocation problem involves finding the optimal distribution of R to S that minimizes the cost of evaluating Q". For databases, the Allocation Problem involves finding disjoint fragments (group of attributes which are accessed together) that are distributed at independent sites. On the other hand, for sensor networks all the attributes are distributed over the same two-dimensional geographical area, such that the communication involved in dissemination of query and retrieval of data for the set of queries Q is minimized.

As in the case of distributed databases, some statistics about query runs (e.g. query access frequency, attribute affinity) are required to solve the allocation problem. Thus, it is assumed that a set of priorities $\{p_1, p_2, \ldots, p_q\}$ are available for the queries $\{Q_1, Q_2, \ldots, Q_q\}$. The priority p_i of a query Q_i could depict either the probability with which query Q_i is issued per unit time or the frequency of tuples generated for query Q_i per unit time or a combination of both. These values may be set by the system designer to specify the relative priorities of queries and are normalized such that $\sum_{i=1}^{q} p_i = 1$. Each query Q_i can be modeled by the set of attributes included in the query i.e., $Q_i = \{A_{i_1}, A_{i_2}, \ldots, A_{i_k}\}(A_{i_p} \in \{A_1, A_2, \ldots, A_m\}, p = 1, \ldots, k)$. The sensor nodes are assumed to be uniformly deployed over a rectangular region. Thus, the objective is to distribute the m attributes $\{A_1, A_2, \ldots, A_m\}$ over this rectangular field of sensor nodes. To do so, the rectangular area is split into a grid G of size $|\sqrt{m}| \times |\sqrt{m}|$ and each attribute A_i is allocated to a particular grid cell. The allocation problem thus reduces to finding an optimal distribution of m attributes over a $|\sqrt{m}| \times |\sqrt{m}|$ grid. Let $f{:}A \rightarrow G$ be an optimal distribution function. Thus $f(A_k) = (x_k, y_k), x_k \in \{1, \ldots, |\sqrt{m}|\}, y_k \in \{1, \ldots, |\sqrt{m}|\}$ implies that attribute A_k should be stored in grid cell (x_k, y_k) of the deployed rectangular sensor network. The function f then attempts to minimize the following two costs for each query $Q_i, i = 1, \ldots, q$:

- *Query Evaluation Cost* C_E^i: This is the cost of aggregating resultant tuples for all the attributes $\{A_{i_1}, A_{i_2}, \ldots, A_{i_k}\}$ specified in the query Q_i. For minimizing this cost, attributes belonging to the same query should be placed near each other in the grid, i.e. the cost of the minimum spanning tree joining the grid cells $\{(x_{i_1}, y_{i_1}), (x_{i_2}, y_{i_2}), \ldots, (x_{i_k}, y_{i_k})\}$ should be minimized.
- *Query Access Cost* C_A^i: This includes the cost of query dissemination from the sink and result delivery to the sink. In order to minimize this cost, the query attributes should be placed close to the sink.

Thus, the function f should minimize the total query evaluation cost $C_E(\sum_{i=1}^{q} C_E^i)$ and the total query access cost $C_A(\sum_{i=1}^{q} C_A^i)$ for the set of queries Q.

This distribution f should be made available to all the nodes in the network so that every node is aware about where every attribute is stored in the network. The method of determining the function f is explained in Section 5. Since the proposed method for computing the function f is topology independent, each sensor node may compute f independently. However, to reduce the overhead involved in performing the same redundant computation all over the network, the function f may be computed at the sink and then sent to all the nodes in the network. For now, let us assume that such a function f is available to all the nodes in network and analyze how it facilitates query processing in a large sensor network.

4. PROPOSED ARCHITECTURE

As mentioned in Section 3, the entire network is divided into a $|\sqrt{m}| \times |\sqrt{m}|$ grid. Each sensor node is

assumed to be location-aware [9,10] and hence can determine which grid-cell it belongs to. The number of sensor nodes in each grid cell depends on the density and distribution of nodes in the sensor network. Given the function f (as computed in Section 5), each attribute A_i is then assigned a grid cell $f(A_i)$ of the network. The grid cell $f(A_i)$ is responsible for storing values of the attribute A_i. Thus, even though the attribute A_i is sensed all over the network (assuming complete sensing coverage), the values of the attribute are stored in the grid cell $f(A_i)$ and every node in the network knows in which grid cell each attribute is stored.

The grid cell $f(A_i)$ responsible for storing attribute A_i, is referred to as the zone for attribute A_i and denoted by Z_i. Only some of the nodes in the zone Z_i actually store the values for the attribute A_i. These nodes are called the *storage nodes* for A_i. The number of storage nodes required for an attribute A_i depends on the amount of data values corresponding to attribute A_i and the storage capacity of each sensor node. To facilitate data retrieval from these storage nodes, one node in every zone is designated as the *control node*. The control node is responsible for fetching data from the storage nodes using specialized indexes that it maintains. Some of the remaining nodes in a zone may store redundant or summary information about attribute values to provide fault tolerance and are hence called the *replica nodes*. Figure 5 shows the proposed architecture where the control node is the node nearest to the center of each zone.

4.1. Processing of Queries

The steps involved in serving a query are as follows:

- Sink gets a query $Q_i = \{A_{i_1}, A_{i_2}, \ldots, A_{i_k}\}$ whose attributes are stored in the zones $\{Z_{i_1}, Z_{i_2}, \ldots, Z_{i_k}\}$.
- The sink computes the optimal route for disseminating the query to the zones $\{Z_{i_1}, Z_{i_2}, \ldots, Z_{i_k}\}$ storing the query attributes. The optimal route for retrieving the attributes $\{A_{i_1}, A_{i_2}, \ldots, A_{i_k}\}$ would essentially be the minimum spanning tree joining sink with the zones $\{Z_{i_1}, Z_{i_2}, \ldots, Z_{i_k}\}$. The complexity involved in calculating this route is minimal (any minimum spanning tree algorithm may be used) since a query usually contains a limited number of attributes.
- The computed route is used for routing the query and routing the resultant data tuples back to the sink. Suppose the sink sends the query to zone Z_{i_j}.
- The control node of zone Z_{i_j} uses its stored indexes to retrieve relevant values of A_{i_j} from the storage nodes.
- While routing the data to the sink, the control nodes of the respective storage zones use aggregation [11,12] and in-network processing mechanisms [13,14] to further reduce the amount of data transfer in the network.

4.2. Updating of Stored Data

The proposed architecture achieves efficiency in query processing at the cost of maintaining updated values of all the attributes in the zones where they are stored. To reduce the control overhead involved in this maintenance, a soft threshold and timer-based scheme may be used as follows:

- Suppose a sensor node in zone Z_i senses an attribute A_k. Suppose the soft threshold for attribute A_k is ϕ_k. If the difference between the previously sensed value and the current value of A_k is more than ϕ_k, the current value needs to be reported to the storage zone Z_k housing A_k. However, instead of sending update messages for every individual node fluctuation, the sensor node first sends an update message to the control node of its own zone Z_i.
- The control node of Z_i waits for a predefined time interval T_k. All the update messages for A_k that the control node receives from nodes in its own grid cell during time T_k are then combined into *aggregate update message(s)* and sent to zone Z_k housing A_k. The duration T_k is called the *update epoch* for attribute A_k.

Increasing the soft threshold and the update epoch reduces the frequency with which the attributes are updated. Reducing the number the update messages in turn reduces the communication overhead involved in maintaining updated values of the attributes in their respective storage zones. However, increasing the soft threshold and update epoch also reduces the probability of the zone Z_k having updated values of A_k. This in turn, reduces the probability that queries accessing A_k would receive the current values for A_k as sensed by the network which might be a serious problem for critical data. Thus, depending on how critical an attribute is, a

suitable soft threshold and update epoch may be chosen to balance the tradeoff between energy-efficiency and accuracy.

Note that all the messages (viz. query message, update message or resultant data tuple) can be routed in the network using geographic routing protocols like GPSR [15].

4.3. Advantages and Limitations

As with all data-centric storage schemes, the proposed DCAAR scheme obviates the need for flooding queries in the network. The user response time for queries is also minimized. Moreover, DCAAR minimizes both the query access cost and the query evaluation cost. In addition to these benefits, having all values of an attribute at one place provides helpful global context for evaluating local data. For example, sensed temperature values could be compared against the average temperature value of the network to detect fires or other local temperature spikes. Furthermore, user-defined parameters like soft threshold and update epoch allow the user to tune the performance of the system as per his requirements.

The DCAAR scheme works well as long as the overhead of sending update messages to storage zones does not supersede the advantage of minimizing the query cost. That is, if the attributes are such that they have very frequent fluctuations or fluctuations continually occur all over the network while the queries are not frequent enough, then more energy may be expended in proactively maintaining updated values in storage zones. However, since most real-life physical phenomena are localized, the fluctuations can be considered to be mostly local and not too many. Consequently, the DCAAR scheme should perform well in most real-life situations.

5. ATTRIBUTE ALLOCATION METHODOLOGY

Having discussed the benefits of distributing attributes over a sensor network and how it assists in efficient query processing, this section focuses on the methodology for determining a good distribution of attributes such that the total cost of serving a set of user-defined queries is minimized. The methodology has two phases. In the first phase, the query priorities are used to determine correlations between each pair of attributes. In the second phase the correlations are used to determine the distribution of attributes to the rectangular sensor network. To illustrate the proposed methodology, consider an example set of queries as listed in Table I.

5.1. Phase 1: Determining Correlations

Query priorities can be used to determine correlations between attributes. If a pair of attributes is a part of a high priority query, then the attributes may be considered to have a high correlation between them since they would be accessed together very frequently. Similarly, if a pair of attributes is never accessed together in the same query, the attributes may be considered to not have any correlation between them. The distribution function should then store attributes with higher correlations closer to each other. The correlations between all the attributes can be represented by a tree of attributes where the edge weights between a pair of attributes represent the correlation between them. Using this data structure and its represented correlations, the distribution of attributes can be determined. Also attributes that are accessed more frequently should be stored closer to the sink. The individual access probability $P(A_i)$ of an attribute A_i can be computed as follows:

$$P(A_i) = \sum_{j=1}^{q} P(Q_j) P(A_i|Q_j), \quad \forall i = 1, \ldots, m,$$

where

$$P(Q_i) = p_j, \quad \forall j = 1, \ldots, m \text{ and}$$
$$P(A_i|Q_j) = \begin{cases} 1, & A_i \in Q_j \\ 0, & A_i \notin Q_j \end{cases} \quad (1)$$

Table I. List of queries in ascending order of priorities

Q_i	p_i	A_i	Q_i	p_i	A_i
Q_{30}	0.0001	A_6	Q_{10}	0.0044	A_{20}
Q_{29}	0.0001	A_{12},A_{19}	Q_6	0.0052	A_7
Q_{24}	0.0001	A_{10},A_{16}	Q_8	0.0063	A_5,A_8,A_{11}
Q_{27}	0.0001	A_{12},A_{18}	Q_{22}	0.0117	A_1,A_6,A_{11},A_{16}
Q_{18}	0.0001	A_4,A_8,A_{12},A_{19}	Q_{26}	0.0136	A_{14},A_{20},A_6
Q_{12}	0.0001	A_{13},A_{16}	Q_{14}	0.0161	A_3,A_7
Q_{23}	0.0005	A_1,A_6,A_{11}	Q_4	0.0185	A_5,A_7,A_9
Q_{15}	0.0005	A_{11}	Q_{16}	0.0189	A_{15}
Q_{17}	0.0007	$A_{19},A_3,\ A_7,A_{11},A_{15}$	Q_2	0.0246	$A_8,A_{16},A_4,A_{12},A_{20}$
Q_{28}	0.0007	A_5	Q_{13}	0.0510	A_{19}
Q_{21}	0.0008	A_{16}	Q_{25}	0.1017	A_8
Q_{20}	0.0010	A_8,A_{13}	Q_{19}	0.1179	A_{16},A_{20},A_4
Q_3	0.0011	A_9,A_{18},A_7,A_{20}	Q_5	0.1185	A_6,A_{16}
Q_{11}	0.0015	A_4,A_7,A_{10}	Q_1	0.1994	A_1,A_3,A_5,A_7,A_9
Q_9	0.0015	A_{14},A_{17}	Q_7	0.2833	A_2,A_{15},A_{17},A_{19}

Table II. List of attributes with access probabilities

A_i	$P_i(A_i)$	A_i	$P_i(A_i)$	A_i	$P_i(A_i)$
A_1	0.2116	A_8	0.1337	A_{15}	0.3029
A_2	0.2833	A_9	0.2190	A_{16}	0.2737
A_3	0.2162	A_{10}	0.0016	A_{17}	0.2848
A_4	0.1441	A_{11}	0.0197	A_{18}	0.0012
A_5	0.2249	A_{12}	0.0249	A_{19}	0.3352
A_6	0.1444	A_{13}	0.0011	A_{20}	0.1616
A_7	0.2425	A_{14}	0.0151		

Table II gives the individual access probabilities for the attributes involved in the queries listed in Table I. To represent the relative ordering of attributes with respect to their individual access probabilities of attributes, attributes are represented in the form of a heap so that if attribute A_i is parent of A_j then $P(A_i) \geq P(A_j)$. This heap-like data structure is referred to as the *correlation tree* and it gives a synoptic view of all the correlations between attributes as also the relative ordering of attributes with respect to their individual access probabilities and thereby assists in determining a good distribution function f.

To create a correlation tree for a given set of queries, to start with each query is represented as a tree of depth 1 and then these individual query trees are combined to form a comprehensive correlation tree. Figure 1 shows the query tree corresponding to query Q_7 listed in Table I. Since the correlation tree should have a heap-like structure, the attribute in the query having the highest access probability is made the root of the corresponding query tree (refer to access probabilities listed in Table II). Hence attribute A_{19} becomes the root for query tree for Q_7. Also as mentioned before, the edge weights depict the correlation between the attributes joined by the edge. For the initial query tree, the edge weights are simply the query priorities. Such a query tree is created for every given query. These query trees are combined to form the final correlation tree. This is done iteratively by selecting query trees in ascending order of their associated query priorities.

The process is illustrated using the set of queries listed in Table I. First the query tree corresponding to Q_{30} is selected and set to be the initial partial correlation tree. Next the query tree corresponding to query Q_{29} is combined with the partial correlation tree. The process of combining query trees continues using usual tree union algorithms, reinforcing edge weights as required. However, special consideration is required when an attribute has different parents in the query tree to be added and the partial correlation tree, respectively. To illustrate this, consider the partial correlation tree after query tree for query Q_{23} has been added (Figure 2). While attempting to add query tree for Q_{17}, it is found that attribute A_{11} has attribute A_{19} as parent in the query tree but attribute A_1 as parent in the partial correlation tree. Thus, it needs to be decided as to which attribute should be parent of A_{11} in the new correlation tree. The attribute with which A_{11} has higher correlation should be chosen, because the second stage of the methodology allocates attributes to grid cells such that attributes are stored closer to their parent attribute. Also, since every child attribute is stored near its parent, the sibling attributes also end up being stored fairly close to each other in the grid. Thus the other parent is made a sibling of the attribute as shown in Figure 3. A triplet of values (.0005,1,11) called the virtual weight is assigned to both attributes A_1 and A_{11} to signify the correlation that attribute A_{11} has with attribute A_1 even though they do not share a parent–child relationship in the tree. Also note that attribute A_{11} could be made a

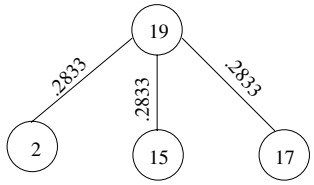

Fig. 1. Query tree for query Q_7.

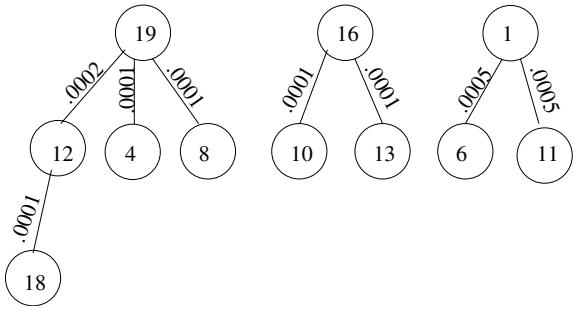

Fig. 2. Partial correlation tree after adding query Q_{23}.

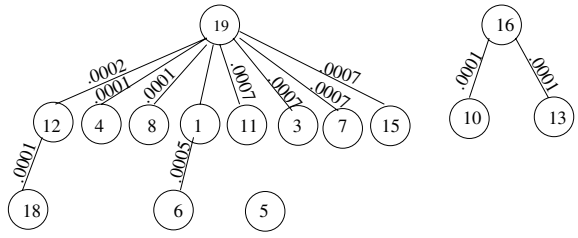

Fig. 3. Partial correlation tree after adding query Q_{17}.

child of attribute A_{19} because $P(A_{11}) < P(A_{19})$. If A_{19} had a higher individual probability, then the position of A_{11} in the tree had to be adjusted using usual heap creation algorithm.

If any edge has to be deleted in the process, a cost-benefit analysis is performed to ensure that the benefit > cost, where cost and benefit are defined as follows,

Cost = Sum of effective weights of deleted edges

Benefit = Sum of effective weights of new edges where effective weight $w'_T(A_i, A_j)$ of an edge (A_i, A_j) is,

$$w'_T(A_i, A_j) = w_T(A_i, A_j) + \Sigma\{\text{virtual weights of } A_j\}$$

and value of a virtual weight (v, A_y, A_z) is,

$$(v, A_y, A_z) = \begin{cases} v & \text{if } A_y \text{ and } A_z \text{ are siblings} \\ 0 & \text{otherwise} \end{cases}.$$

The final correlation tree for the set of queries in Table I is shown in Figure 4. The virtual weights have not been shown to preserve clarity of the figure.

5.2. Phase 2: Allocating Attributes

Once the correlation tree has been constructed, it can be used to determine the distribution of attributes to the grid such that the more frequently accessed attributes are closer to the sink and the attributes with higher correlations are stored closer to each other. The sink is assumed to be in any random location of the network. Hence, more frequently accessed attributes are stored as close to the centre of the grid since the centre is the position that is easily accessible from any random position in the region. The attribute having the maximum access probability is allocated to the central-most grid cell. Then, attributes are chosen iteratively in descending order of their access probabilities and grid cells for storing them are determined using the correlation tree. If the attribute does not have a parent in the correlation tree, it is allocated the grid cell nearest to the centre. However, there may be multiple available grid cells at the same distance from the centre. In that case, the optimal grid cell is the one for which adjacent already-allocated attributes have the least number of unassigned children. The justification for choosing such a cell is that, if a cell C is surrounded with attributes that have more number of unassigned children attributes, then it would be preferable to leave C for those unassigned children attributes when other options are available. Now consider the case where the correlation that an attribute has with its parent is the same as its own access probability. This implies that any query that accesses it also accesses its parent. In that case, the attribute should be stored close to its parent. However, there may be more than one cell at the same distance from its parent. In this case, the optimal grid cell is the one farthest from the

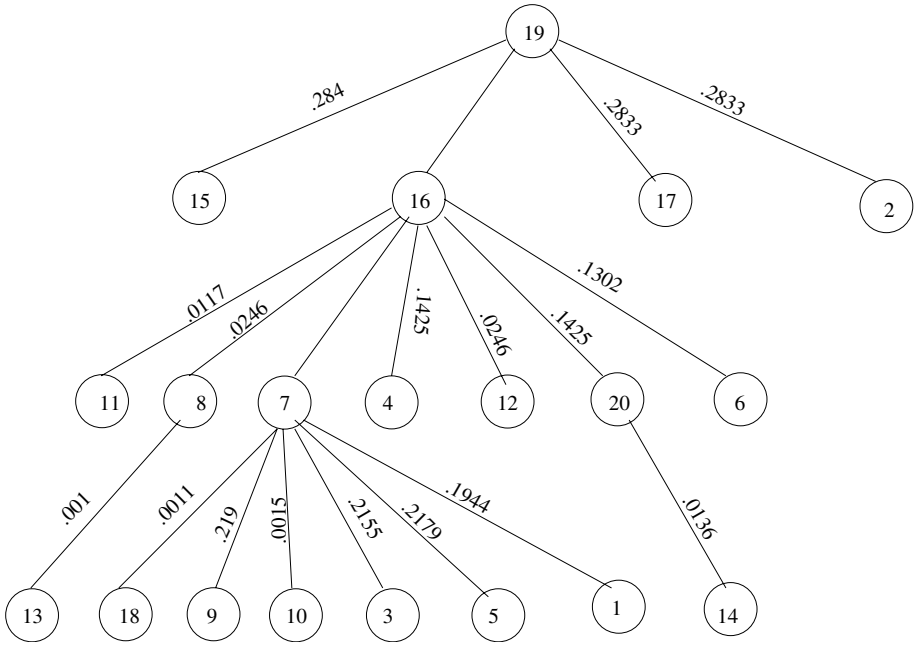

Fig. 4. Correlation tree.

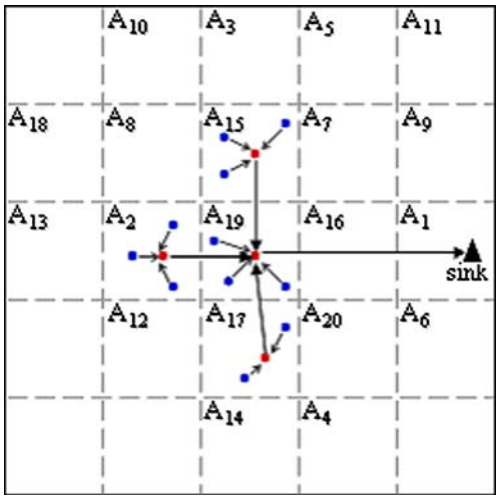

Fig. 5. Allocation of attributes to grid.

centre. This can be justified by the fact that the available cells near the centre are left for attributes that need to be stored close to the centre. Finally, consider the case where the attribute has correlations with multiple attributes. In such a situation, the DCAAR scheme attempts to allocate the attribute to an available grid cell, such that the distance between storage zones of attributes is inversely proportional to the respective correlation value.

Figure 5 shows the allocation of attributes to the grid with the assistance of the correlation tree of Figure 4. Note that attribute A_{19} has highest individual probability and hence is in the centre. Attributes A_{15}, A_{17}, A_2 and A_{16} are then selected in descending order of probabilities and stored near A_{19}. Attribute A_7 is then stored near its parent A_{16} while A_5 is stored near its parent A_7 and so on. Also note that query Q_7 that has the highest priority has all its attributes A_2, A_{15}, A_{17} and A_{19} stored near the centre adjacent to each other. Some of the grid cells in Figure 5 are empty since the number of attributes in our example query set (Table I) is less than the number of grid cells. These empty unassigned grid cells can be used later for fault tolerance and load balancing.

6. ANALYSIS

To analyze the performance of the DCAAR scheme, we study the energy consumed in querying the network and maintaining updated values in storage zones. As mentioned in Section 2, the existing data-centric storage schemes store user-defined events instead of individual attributes. Thus we compare the energy consumption of DCAAR with that of TAG [12]. In TAG, individual attribute values are stored locally at nodes that generate them and an aggregation tree spanning all the sensor nodes is formed in a distributed manner to perform in-network aggregation while delivering query results to the sink. For both DCAAR and TAG, it is assumed that the query selection predicate specifies a hard threshold Φ for a query attribute, and only those attribute readings greater than the hard threshold Φ are reported to the sink. Furthermore, as mentioned in Section 4.2, update messages are sent in the DCAAR scheme only when the soft threshold ϕ is crossed. The sink is assumed to be at the centre of the network as shown in Figure 6. It is also assumed that the communication between nodes of a grid cell and their respective control nodes, the sink and the control nodes, as well as nodes in different levels of the aggregation tree all pack available data in the fewest possible packets. The parameters used in the analysis are listed in Table III.

6.1. Energy Cost of Aggregation Tree (AT)

The queries are propagated downwards till every leaf node in the AT is reached. The AT is rooted at the sink and thus spreads outwards from the center of the region. For best case performance, the AT is assumed to span the region uniformly. We divide the entire area into n concentric square regions, each of width r as shown in Figure 6a. The nodes present in the ith concentric region are assumed to be at a depth i in the AT. Thus n is the depth of the tree where $2nr = L$.

Table III. Parameters used in analysis

N_{total}	Total number of nodes in the network
$A(i)$	Area of sub-region at depth i
r	Transmission radius
L	Length of a side of the deployed region
Φ	Hard threshold
ϕ	Soft threshold
ξ	Ratio of information generated by a single node to size of data packet
P_{tx}	Transmission cost of a single packet
d	Length of a side of the sub-region
N	Nodes reporting a reading $> \Phi$
N'	Nodes reporting a reading $> \phi$
D(sk,stg)	Distance between sink and storage region
μ	Frequency of attribute value crossing Φ
μ'	Frequency of attribute value crossing ϕ
f_Q	Query frequency

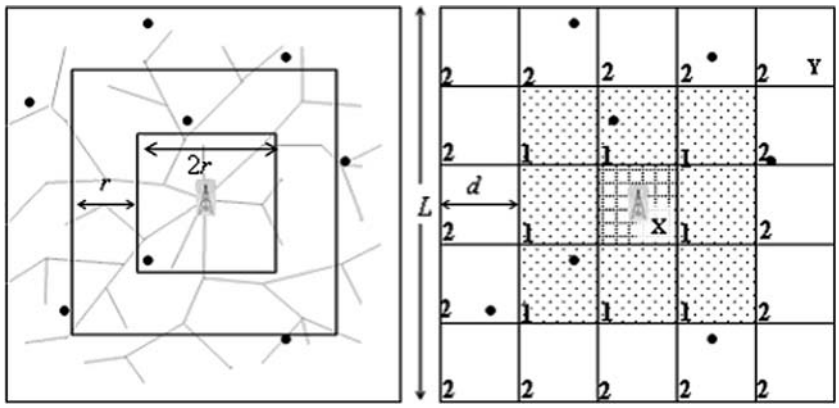

Fig. 6. Aggregation tree vs. DCAAR.

Area of region i is $A(i) = 4(2i-1)r^2$, $i = 1, 2, 3, \ldots, n$.

The probability that a node lies in region i is given by $p(i) = \frac{A(i)}{L^2}$. Thus, out of N nodes, the number of nodes in the ith region is given by $Np(i)$. Consequently, the energy cost incurred at each level is $E(i) = I(i) \times \xi \times P_{tx}$ and the total energy is given by $E_{\text{Data(AT)}} = \sum_{i=1}^{n} E(i)$. We now calculate $I(i)$, the total number of packets transmitted at level i of the aggregation tree. This includes the total number of packets generated at level i as well as the packets arriving at level $i + 1$. Thus a recursive relation is obtained as follows as:

$$I(n) = 4N(2n-1)\left(\frac{r}{L}\right)^2,$$

$$I(n-1) = 4N(2n-3)\left(\frac{r}{L}\right)^2 + I(n), \ldots,$$

and

$$I(1) = 4N\left(\frac{r}{L}\right)^2 + I(2)$$

Thus the total incurred communication cost,

$$E_{\text{Data(AT)}} = \left[\sum_{i=1}^{n} I(n)\right] \times \xi \times P_{tx}$$

Through algebraic manipulation,

$$I(1) = 4N\left(\frac{r}{L}\right)^2 (n)^2, \quad I(2) = 4N\left(\frac{r}{L}\right)^2 [(n)^2 - 1], \ldots,$$

$$I(i) = 4N\left(\frac{r}{L}\right)^2 [(n)^2 - (i-1)^2],$$

Therefore,

$$\left[\sum_{i=1}^{n} I(n)\right] \times \xi \times P_{tx} = 4N\left(\frac{r}{L}\right)^2 \xi P_{tx}\left[n^3 - \frac{n(n-1)(2n-1)}{6}\right] \quad (2)$$

Since each node forwards the query exactly once, the initial cost of propagating the query down the tree is given by

$$E_{\text{Query(AT)}} = N_{\text{total}} P_{tx} \quad (3)$$

For query-driven systems, information is sent to the sink only in response to a query. Hence, total cost of query dissemination and aggregation is,

$$E_{\text{Total}}(AT)_P = f_Q\left(E_{\text{Query(AT)}} + E_{\text{Data(AT)}}\right) \quad (4)$$

The total cost incurred in event-driven systems is dependent on the number of reports generated by the system with $E_{\text{Query(AT)}} = 0$. In this case, the cost is,

$$E_{\text{Total}}(AT)_R = E_{\text{Data(AT)}}\mu$$

6.2. Energy Cost of DCAAR Scheme

Let us assume that the storage region X in Figure 6b stores the values for the desired query attribute. The squares marked '1' represent the first tier of surrounding sub-regions, those marked '2' indicate the second tier and so on. We approximate the distance from any node of the ith tier to the center of X as id and this distance is traversed in id/r hops. Thus the probability of a node lying in regions marked 1, 2, is, $p(1) = \frac{8d^2}{L^2}, p(2) = \frac{16d^2}{L^2} \ldots$ up to $\lfloor L/2d \rfloor$ tiers.

Hence, $p(i) = \frac{8id^2}{L^2}$ where $i = 1,2, \ldots, \lfloor L/2d \rfloor$

N' number of nodes report update messages to control nodes of their own grid cell. In the worst case scenario, the reporting nodes are furthest away from

the control node at a distance of d/r hops. Thus energy spent in this update is,

$$E_{\text{Update}} = N'\left(\frac{d}{r}\right)P_{tx}$$

The number of nodes reporting update of attribute values in tier i is $N'p(i)$. Energy cost associated with all control nodes in tier i reporting aggregated update message to region X is,

$$E(i) = \frac{8iN'd^2}{L^2}\left(\frac{di}{r}\right)\xi P_{tx} \tag{5}$$

Thus, in one reporting event, total energy spent by all control nodes in reporting data to the storage region hosting that attribute is given by

$$\begin{aligned} E_{\text{Data(DS)}} &= \sum_{i=1}^{L/2d} E(i) = \frac{8N'd^3}{rL^2}\xi P_{tx}\sum_{i=1}^{L/2d} i^2 \\ &= \sum_{i=1}^{L/2d} E(i) = \frac{2N'd^3}{3rL^2}\xi P_{tx} \\ &\quad [(L/d)(L/2d+1)(L/d+1)] \end{aligned} \tag{6}$$

The storage region receives queries from the sink with a frequency f_Q and incurs a query cost of $E_{\text{Query(DS)}} = \frac{D(\text{sk,stg})}{r}P_{tx}$. The consequent reply from the storage region to the sink incurs a cost of $E_{\text{Reply(DS)}} = \frac{D(\text{sk,stg})}{r}N\xi P_{tx}$

The total energy expended in our scheme is given by

$$\begin{aligned} E_{\text{Total}}(DS) =& f_Q\left(E_{\text{Query(DS)}} + E_{\text{Reply(DS)}}\right) \\ &+ \left(E_{\text{Data(DS)}} + E_{\text{Update}}\right)\mu' \end{aligned} \tag{7}$$

Thus, the DCAAR scheme should be preferred only when $E_{\text{Total}}(DS) < E_{\text{Total}}(AT)$ i.e.

$$\begin{aligned} & f_Q\left(E_{\text{Query(DS)}} + E_{Reply(DS)}\right) + \left(E_{\text{Data(DS)}} + E_{\text{Update}}\right)\mu' \\ & < \begin{cases} f_Q\left(E_{\text{Query(AT)}} + E_{\text{Data(AT)}}\right) \text{ (Query - driven)} \\ E_{\text{Data(AT)}}\mu \text{ (Event - driven)} \end{cases} \end{aligned} \tag{8}$$

To validate our models of the aggregation tree and DCAAR scheme, we simulated a testbed of 1030 nodes distributed randomly in an area of 630×630 square units. A node at (x,y) is given an initial attribute value calculated by the function $Z = 20(1 + \sin(R)/R)$, where $R = \sqrt{x^2 + y^2} + 0.5$. We allow 15 possible fluctuations in the sensed attribute, but the decision to undergo a change was taken locally at each node with probability 0.3. The magnitude of change is ± 1.8 and is a function of a uniformly distributed random variable. At the end of the simulation we obtained an average of 340 nodes crossing $\varphi = 0.7$ at each fluctuation and those that are over $\Phi = 22$ averaged at 380. The other constant simulation parameters are as described in Section 7. The calculated and observed values are summarized in Table IV and are in good agreement thereby validating our model. As predicted by Eq. 8, DCAAR has lower energy cost under these conditions and is the preferred choice.

Table IV. Energy cost analysis (mJ)

DCAAR		Aggregation tree	
Simulation	Calculated	Simulation	Calculated
19.7311	19.5864	26.351	21.073

7. SIMULATION RESULTS

Using Simjava [16], a general discrete event simulator, we have conducted extensive simulations to compare the performance of the proposed DCAAR scheme with that of the aggregation tree (AT) algorithm of TAG [12].

7.1. Simulation Environment

For simulation studies, 1030 nodes have been randomly dispersed in a square area of 630×630 units, each node having a transmission range of 40 units. Each packet of 30 bytes is transmitted over a 20 kbps channel, incurring a cost of 0.81 and 0.3 mW for transmission and reception, respectively. To investigate the performance of the DCAAR scheme, we split the area into 49 subregions, each of side 90 units. It is assumed that the communication between nodes of a grid cell and their respective control nodes, the sink and the control nodes, as well as nodes in different levels of the aggregation tree, pack available data in the fewest possible packets. We assume that the sink is at the centre of the network. To evaluate the best case performance of the proposed DCAAR scheme, the attribute queried is assumed to be stored in the central grid cell (i.e. nearest to the sink). To measure the worst case performance, the sink queries an attribute that is stored in a grid cell further away

from the centre of the network. Energy consumption has been used as the metric to compare the DCAAR scheme with the aggregation tree (AT) algorithm of TAG. For both the schemes, only those readings greater than the hard threshold are reported to the sink. Furthermore, for the proposed DCAAR scheme, the update messages are sent only when the soft threshold is breached. Experiments have been conducted for computing energy consumption against varying query rate and attribute fluctuation frequency as described in the following sections.

7.2. Effect of Query Rate on Performance

First the rate of fluctuations is kept constant and the query rate is varied (Figure 7a, b). The DCAAR scheme shows marginal performance degradation at lower query rates. As the sink injects progressively greater number of queries per unit time, the DCAAR scheme performs increasingly better than the AT algorithm. The reason is as follows. The cost of flooding the query down to the leaf level of the AT and then retrieving the information essentially requires $O(n)$ transmissions. In the proposed DCAAR scheme (and more so for the considered topology), this is accomplished in a single transmission between the control node in the storage region and the sink, as is evident from the minimal DCAAR-Query cost. However, there is a constant overhead of updating the storage region for the nodes which show a variation in the sensed attribute. This is reflected in the almost constant DCAAR-Update cost. Figure 7b shows results for the scenario where the attribute stored at a subregion far from the sink is accessed by a periodic query. Here the break-even point is reached when the query rate is 22 (unlike eight in the best case), after which DCAAR performs much better than AT.

7.3. Effect of Fluctuation Rate on Performance

In this experiment (Figure 7c, d), the rate of fluctuations has been varied, while maintaining a steady query rate of 20 queries/simulation time. It can be observed that with an increase in the number of fluctuations, the DCAAR scheme is no longer preferable to the AT scheme after the break-even point. The AT performs at a steady energy cost as nodes, while sensing the changed attribute values, see no need of reporting it unless a query message is received. The minor increase in energy cost of AT

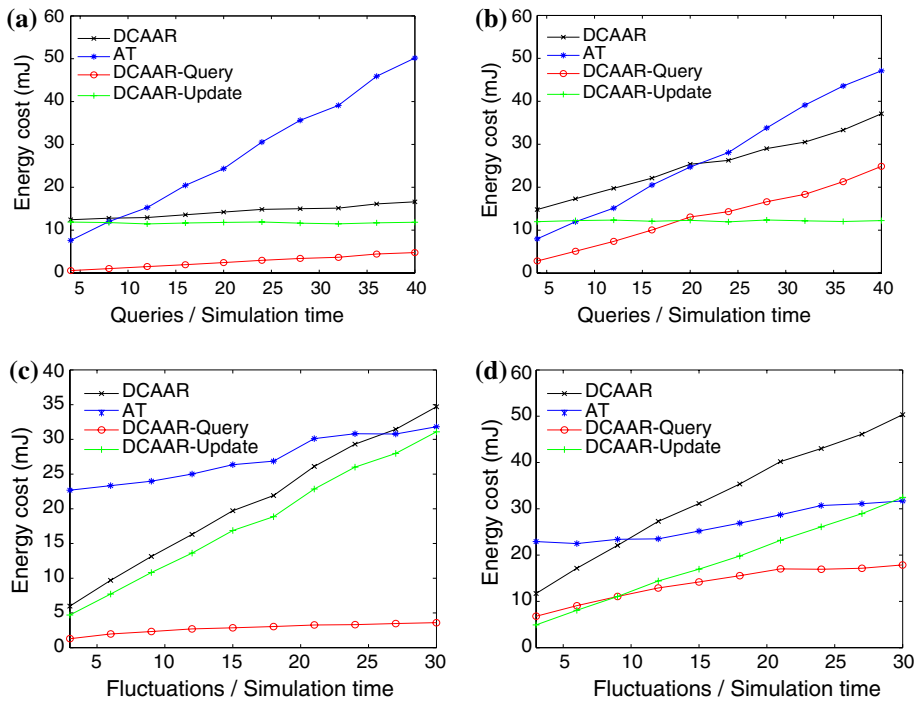

Fig. 7 (a) Effect of query rate (best case). (b) Effect of query rate (attribute far from sink). (c) Effect of fluctuation rate (best case). (d) Effect of fluctuation rate (attribute far from sink).

happens because, with increasing rate of fluctuations, the number of attribute values more than the hard threshold increase and hence more number of nodes report data to the sink. On the other hand, the rapid fluctuation and its associated cost in maintaining updated information in the storage region strains the DCAAR scheme, which explains the steady increase in DCAAR-Update cost and hence DCAAR-total cost. Even then, in the best case scenario (Figure 7c), the DCAAR scheme performs better than AT till the fluctuation rate reaches a considerable value of 26 fluctuations/simulation time.

8. CONCLUSIONS

The proposed DCAAR scheme is a data-centric storage scheme for allocating attributes to a large-scale sensor network depending on the correlations between them. In order to define the optimization criteria involved in selecting appropriate rendezvous locations for sensed data, the problem has been formulated in a manner similar to the allocation problem of distributed databases. Each user-defined query is assumed to have an associated priority, which in turn defines correlations between the sensed attributes. These correlations and priorities have been used to determine storage locations of individual attributes within the network. Also mechanisms have been developed for efficiently retrieving and processing the stored data in response to user queries. Furthermore, a soft threshold and timer-based update mechanism has been proposed to increase the energy efficiency of the system and to better cater to user requirements. Finally, using both analysis and extensive simulations, the preferable conditions for the protocol has been determined. As a part of future work, we plan to develop detailed protocols for query dissemination, data update and data retrieval. We also need to ensure that these protocols are fault-tolerant and perform load balancing.

REFERENCES

1. D. P. Agrawal, R. Biswas, N. Jain, A. Mukherjee, S. Sekhar and A. Gupta, Sensor systems: state of the art and future challenges,'' Book Chapter in *Handbook on Theoretical and Algorithmic Aspects of Ad hoc and Sensor Networks*, pp. 317–346, Auerbach Publications, 2006.
2. S. Shenker, S. Ratnasamy, B. Karp, R. Govindan and D. Estrin, Data-centric storage in sensornets, in *Proceedings of the 1st ACM SIGCOMM Workshop on Hot Topics in Networks*, Princeton, New Jersey, October 2002.
3. S. Ratnasamy, B. Karp, L. Yin, F. Yu, D. Estrin, R. Govindan and S. Shenker, GHT: A Geographic Hash Table for Data-Centric Storage, in *Proceedings of the 1st ACM International Workshop on Wireless Sensor Networks and Applications (WSNA 2002)*, Atlanta, Georgia, September 2002.
4. I. Stoica, R. Morris, D. Karger, F. Kaashoek and H. Balakrishnan, Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications, in *Proceedings of the ACM SIGCOMM 2001 Conference*, San Diego, California, August 2001.
5. S. Ratnasamy, P. Francis, M. Handley, R. Karp and S. Shenker, A scalable content-addressable network, in *Proceedings of the ACM SIGCOMM 2001 Conference*, San Diego, California, August 2001.
6. B. Greenstein, D. Estrin, R. Govindan, S. Ratnasamy and S. Shenker, DIFS: a distributed index for features in sensor networks, in *Proceedings of the 1st IEEE International Workshop on Sensor Network Protocols and Applications*, Anchorage, Alaska, May 2003.
7. X. Li, Y. J. Kim, R. Govindan and W. Hong, Multi-dimensional range queries in sensor networks, in *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems*, Los Angeles, California, November 2003.
8. A. Ghose, J. Grossklags and J. Chuang, Resilient data-centric storage in wireless ad-hoc sensor networks, in *Proceedings of the 4th International Conference on Mobile Data Management (MDM)*, Melbourne, Australia, 2003.
9. J. HighTower and G. Borreillo, Location systems for Ubiquitous Computing, in *IEEE Computer Magazine*, Vol. 34, No. 8, pp. 57–66, Aug 2001.
10. L. Doherty, K. S. J. Pister and L. E. Ghaoui, Convex position estimation in wireless sensor networks, *in Proceedings of the IEEE Infocom*, Anchorage, Alaska, April 2001.
11. C. Intanagonwiwat, R. Govindan and D. Estrin, Directed diffusion: a scalable and robust communication paradigm for sensor networks, in *Proceedings of the 6th Annual ACM/IEEE International Conference on Mobile Computing and Networking (Mobicom 2000)*, Boston, Massachusetts, August 2000.
12. S. Madden, M. J. Franklin, J. M. Hellerstein and W. Hong, TAG: a tiny aggregation service for ad-hoc sensor networks, in *Proceedings of the 5th Annual Symposium on Operating Systems Design and Implementation (OSDI)*, Boston, Massachusetts, December 2002.
13. R. Biswas, N. Jain, N. Nandiraju and D. P. Agrawal, Communication architecture for processing spatio-temporal continuous queries in sensor networks, in *Special issue of the Annals of Telecommunications on Sensor Networks*, Vol. 60, pp. 901–927, July–Aug. 2005.
14. R. Kumar, V. Tsisatsis and M. Srivatsava, ''Computation Hierarchy for In-network Processing,'' in *Proceedings of the 2nd ACM International Workshop on Wireless Sensor Networks and Applications (WSNA'03)*, San Diego, California, September 2003.
15. B. Karp and H. T. Kung, ''GPSR: greedy perimeter stateless routing for wireless sensor networks, in *Proceedings of the 6th Annual ACM/IEEE International Conference on Mobile Computing and Networking (Mobicom '00)*, Boston, Massachussets, August 2000.
16. F. Howell and R. McNab, ''Simjava: a discrete event simulation package for Java with applications in computer systems modelling, in *Proceedings of the 1st International Conference on Web-based Modelling and Simulation*, San Diego, California, January 1998.

Ratnabali Biswas is a PhD candidate in Department of Electrical and Computer Engineering and Computer Science (ECECS) in University of Cincinnati. She received a B.Sc. degree in Mathematics (Honors) from University of Calcutta, India and a Master in Computer Applications degree from Bengal Engineering College, Shibpur (India). Her research interests are in developing energy-efficient protocols for query processing in sensor networks. Her focus includes energy-efficient routing infrastructure for query processing in sensor networks, application-specific energy-efficient distributed data storage in sensor networks as well as providing link-layer quality of service in sensor networks.

Kaushik Chowdary received the B.E. degree in electronics engineering from the Veermata Jijabai Technological Institute (erstwhile affiliated to the University of Mumbai) in 2003. Currently, he is a PhD student in computer science and engineering at the University of Cincinnati. His research interests include resource provision in sensor and ad-hoc networks, distributed channel allocation and multichannel MAC protocols. He is a student member of the IEEE.

Dharma P. Agrawal is the Ohio Board of Regents Distinguished Professor of Computer Science and Engineering and the founding director for the Center for Distributed and Mobile Computing in the Department of ECECS, University of Cincinnati, OH. He has been a faculty member at the N.C. State University, Raleigh, NC (1982–1998) and the Wayne State University, Detroit (1977–1982). His current research interests include energy efficient routing, information retrieval, and secured communication in ad hoc and sensor networks, effective handoff handling and multicasting in integrated wireless networks, interference analysis in piconets and routing in scatternet and use of smart directional antennas for enhanced QoS. Thomson has recently published the second edition of his recent co-authored textbook entitled Introduction to Wireless and Mobile Systems, specifically designed for Computer Science & Engineering students. He is an editor for the Journal of Parallel and Distributed Systems, the International Journal of High Speed Computing, International Journal on Distributed Sensor Networks, Taylor and Francis Journal, International Journal of Ad Hoc and Ubiquitous Computing (IJAHUC), Interscience Publishers, 2004 and the International Journal of Ad Hoc & Sensor Wireless Networks. He has served as an editor of the IEEE Computer magazine, and the IEEE Transactions on Computers. He has been the Program Chair and General Chair for numerous international conferences and meetings. He has received numerous certificates and awards from the IEEE Computer Society. He was awarded a Third Millennium Medal, by the IEEE for his outstanding contributions. He has also delivered keynote speech for five international conferences. He also has four patents in wireless networking area. He is a Fellow of the IEEE, the ACM, the AAAS, and the WIF.