Attribute Allocation in Large Scale Sensor Networks

Ratnabali Biswas Kaushik Chowdhury Dharma P. Agrawal
OBR Research Center for Distributed and Mobile Computing, Dept. of ECECS,
University of Cincinnati, Cincinnati, OH 45221-0030
{biswasr,kaushir,dpa}@ececs.uc.edu

ABSTRACT

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. Finally, we present simulations results to illustrate the conditions under which our proposed architecture is beneficial. To the best of our knowledge, this is the first attempt to study the attribute allocation problem of distributed databases in the context of sensor networks.

Categories and Subject Descriptors

C.2.1 [**Computer Communication Networks**]: Network Architecture and Design; C.3 [**Computer Systems Organization**]: Special Purpose and Application-based Systems.

General Terms

Algorithms, Performance, Design, Experimentation.

Keywords

Allocation problem, Data-centric storage, Query dissemination, Data retrieval, Aggregation, In-network processing, Minimum spanning tree, Correlation tree, Hard threshold, Soft threshold.

1. INTRODUCTION

A wireless sensor network is a dense network of a large number of low cost miniature wireless sensor nodes driven by a limited battery resource. At the node level, data communication is the dominant component of energy consumption, and hence protocol design for sensor networks is geared towards reducing communication in the network. In this paper, we have designed a scheme for determining a distribution of attributes in a sensor network that minimizes the communication cost involved in querying the network.

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. The proposed scheme is designed for minimizing cost of data retrieval from large-scale sensor networks serving such real-life scenarios.

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 and gives an outline of how such 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 4. The simulation results are discussed in Section 5, while Section 6 concludes the paper with future research agenda.

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" now allows events to be stored at specific rendezvous points within the network that queries can access directly. Shenker et al. [7] proposed the DCS scheme and have shown that DCS outperforms other approaches such as External Storage (ES) and Local Storage (LS) under certain circumstances. Ratnasamy et al. [6] have also proposed a Geographic Hash Table (GHT) to hash events into geographic coordinates. In DIFS [2], Greenstein et al. have designed a spatially distributed index to facilitate range searches over attributes, while Li et al. [5], have built a distributed index (DIM) for multidimensional range queries of attributes.

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 [2][5][6] 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.

3. DISTRIBUTED ATTRIBUTE STORAGE

Sensor networks can be envisioned as a large distributed database where the sensor nodes generate named data against user-specified queries [1]. In this section we define the Allocation Problem of distributed databases in the context of sensor networks and illustrate how our proposed approach assists in query processing.

3.1 Problem Definition

Assume that there are a set of sensed attributes $A = \{A_1, A_2, ..., A_m\}$ and a network S of sensor nodes which have to serve a set of queries $Q = \{Q_1, Q_2, ..., Q_q\}$. We attempt to find a distribution of A to S such that the energy consumed in serving the set of queries Q is minimized. 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 minimize the cost of evaluating Q". For databases, the Allocation Problem involves finding disjoint fragments (group of attributes which are accessed together) that are distributed in 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 case of distributed databases, some statistics about query runs (e.g. query access frequency, attribute affinity) are required to solve the allocation problem. We thus assume that a set of priorities $\{p_1, p_2, \mathrm{K}, p_q\}$ for the queries $\{Q_1, Q_2, \mathrm{K}, Q_q\}$ is given. 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 modelled by the set of attributes included in the query i.e. $Q_i = \{A_{i_1}, A_{i_2}, \mathrm{K}, A_{i_k}\}$ $(A_{i_p} \in \{A_1, A_2, \mathrm{K}, A_m\}, p = 1,...,k)$. We also assume that the sensor nodes are uniformly deployed over a rectangular region. We thus intend to distribute the m attributes over this rectangular field of sensor nodes so that the cost querying the network is minimized. To do so, we split the rectangular area into a grid G of size $\lceil\sqrt{m}\rceil \times \lceil\sqrt{m}\rceil$ and each attribute A_i is allocated to a particular grid cell. Let $f : A \rightarrow G$ be the mapping corresponding to the optimal distribution. In order to minimize the total cost of serving the set of queries Q, the mapping f should minimize the communication involved in disseminating the query and aggregating the resultant tuples for each query. To minimize the cost involved in evaluating a query (i.e. aggregating resultant tuples), the attributes belonging to the same query should be stored near each other in the grid. Also to minimize the communication involved in disseminating the query from the sink and reporting the results back to the sink, the attributes should be stored as close to the sink as possible. We thus need to determine a good distribution f for a given set of queries and this distribution f should be made available to all the nodes in the network so that every node is aware as to where every attribute is stored in the network. The method of determining the function f is explained in Section 4. Since the proposed method for computing 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, we 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.

3.2 Query Processing

As mentioned in Section 3.1, the entire network is divided into a $\lceil\sqrt{m}\rceil \times \lceil\sqrt{m}\rceil$ grid. We assume that each sensor node is location-aware [10][11] 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 a distribution function f (as computed in Section 4), each attribute A_k is then assigned a grid cell $f(A_k)$ of the network. The grid cell $f(A_k)$ is responsible for storing values of the attribute A_k. Thus even though the attribute A_k is sensed all over the network (assuming complete sensing coverage), the values of the attribute are stored in the grid cell $f(A_k)$ and every node in the network knows in which grid cell each attribute is stored.

Let us now consider the attribute A_k and the grid cell $f(A_k)$ where it should be stored. Only some of the nodes in the grid cell $f(A_k)$ actually store the values for the attribute A_k. These nodes are called the *storage nodes* for A_k. The number of such nodes required for an attribute A_k depends on the amount of data values corresponding to attribute A_k as also the storage capacity of each sensor node. To facilitate data retrieval from these storage nodes, one node in every grid cell is appointed as the *control node*. The control node is responsible for fetching data from the storage nodes using specialized indexes that it maintains. To disseminate a query Q_i, the sink node first determines the nearest grid cell housing any of the attributes in the query and sends the query to that grid cell. On receiving the query, the control node of the corresponding grid cell then uses its stored indexes to retrieve the required values of the stored attribute from the respective storage nodes. It then computes the optimal route of disseminating the query to the grid cells storing the remaining attributes of the query. The optimal route for retrieving the remaining attributes belonging to Q_i would essentially be the minimum spanning tree joining the grid cells that store the attributes in Q_i. The complexity involved in calculating this route is minimal (any minimum spanning tree algorithm may be used) since a query will not contain a large number of attributes. The computed route is then used for routing the query as well as routing the resultant data tuples back to the sink. While routing the data back to the sink, the control nodes of the grid cells housing the respective attributes use aggregation schemes [3][4] and in-network processing mechanisms [8] to further minimize the amount of data transfer in the network.

The efficiency in query processing and information retrieval using such an underlying architecture is achieved at the cost of maintaining updated values of all attributes in the respective grid cells where they are stored. To reduce the overhead involved in this maintenance a soft threshold scheme may be used. Whenever a sensor node senses an attribute, it determines whether the difference between the previously sensed value and the new value is more than a predefined soft threshold. If the difference is greater than the soft threshold, the new value needs to be reported to the grid cell housing the corresponding attribute. 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 grid cell. The control node then waits for a predefined time interval called the update epoch T_k for the sensed attribute A_k. All the update messages for the attribute A_k that the control node receives from nodes in its own grid cell during the epoch T_k are then combined into *aggregate update message(s)* and sent to the grid cell $f(A_k)$ housing 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 the updated values of the attributes in the grid cells where they are housed. However increasing the soft threshold and update epoch also reduces the probability of the grid cell $f(A_k)$ having updated values of A_k as sensed in the network. 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 reduce the update overhead.

This section described how queries can be processed once the attributes are stored in the network in a distributed manner. All the underlying routing of query and data messages can be done using geographical routing schemes like GPSR [9]. In the next section we present a heuristic for obtaining a good distribution of attributes over a sensor network given a set of queries along with their associated priorities.

4. ATTRIBUTE ALLOCATION METHODOLOGY

Having discussed the benefits of distributing attributes over a sensor network and how it assists in efficient query processing, we now focus 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, let us consider an example set of queries as listed in Table 1.

Table 1. List of Queries in ascending order of priorities

Q_i	P_i	A_i	Q_i	P_i	A_i
Q_{30}	.0001	A_6	Q_{10}	.0044	A_{20}
Q_{29}	.0001	A_{12},A_{19}	Q_6	.0052	A_7
Q_{24}	.0001	A_{10},A_{16}	Q_8	.0063	A_5,A_8,A_{11}
Q_{27}	.0001	A_{12},A_{18}	Q_{22}	.0117	A_1,A_6,A_{11},A_{16}
Q_{18}	.0001	A_4,A_8,A_{12}, A_{19}	Q_{26}	.0136	A_{14},A_{20},A_6
Q_{12}	.0001	A_{13},A_{16}	Q_{14}	.0161	A_3,A_7
Q_{23}	.0005	A_1,A_6,A_{11}	Q_4	.0185	A_5,A_7,A_9
Q_{15}	.0005	A_{11}	Q_{16}	.0189	A_{15}
Q_{17}	.0007	A_{19},A_3,A_7, A_{11},A_{15}	Q_2	.0246	A_8,A_{16},A_4, A_{12},A_{20}
Q_{28}	.0007	A_5	Q_{13}	.0510	A_{19}
Q_{21}	.0008	A_{16}	Q_{25}	.1017	A_8
Q_{20}	.0010	A_8,A_{13}	Q_{19}	.1179	A_{16},A_{20},A_4
Q_3	.0011	A_9,A_{18},A_7, A_{20}	Q_5	.1185	A_6,A_{16}
Q_{11}	.0015	A_4,A_7,A_{10}	Q_1	.1994	A_1,A_3,A_5, A_7,A_9
Q_9	.0015	A_{14},A_{17}	Q_7	.2833	A_2,A_{15},A_{17}, A_{19}

4.1 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 we consider the attributes 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:

$$P(A_i) = \sum_{j=1}^{q} P(Q_j) P(A_i \mid Q_j), \forall i = 1, \mathrm{K}\ , m’$$

where $P(Q_j) = p_j, \forall j = 1, \mathrm{K}\ , m$ and $P(A_i \mid Q_j) = \begin{matrix} 1, A_i \in Q_j \\ 0, A_i \notin Q_j \end{matrix}$

Table 2 gives the individual access probabilities for the attributes involved in the queries listed in Table 1. To represent the relative ordering of attributes with respect to their individual access probabilities of attributes, we can further represent the attributes in the form of a heap so that if attribute A_i is parent of A_j, then $P(A_i) > P(A_j)$. We call this heap-like data structure 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*.

Table 2. List of Attributes and their access probabilities

A_i	P_i	A_i	P_i	A_i	P_i
A_1	.2116	A_8	.1337	A_{15}	.3029
A_2	.2833	A_9	.2190	A_{16}	.2737
A_3	.2162	A_{10}	.0016	A_{17}	.2848
A_4	.1441	A_{11}	.0197	A_{18}	.0012
A_5	.2249	A_{12}	.0249	A_{19}	.3352
A_6	.1444	A_{13}	.0011	A_{20}	.1616
A_7	.2425	A_{14}	.0151		

To create a correlation tree for a given set of queries, we begin by representing each query as a tree of depth 1 and then combine these individual query trees to form a comprehensive correlation tree. Figure 1 shows the query tree corresponding to query Q_7 listed in Table 1. Since the correlation tree should have a heap-like structure, the attribute in the query having the maximum access probability is made the root of the corresponding query tree (refer to access probabilities listed in Table 2). 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 then need to be combined to form the final correlation tree. This is done iteratively by selecting query trees in ascending order of their associated query priorities.

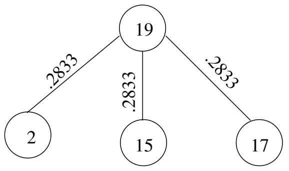

Figure 1. Query Tree for query Q_7

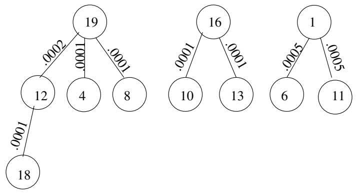

Figure 2. Partial Correlation after adding query Q_{23}

We illustrate the process using the set of queries listed in Table 1. For our example, 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, let us consider the partial correlation tree after query tree for query Q_{23} has been added (refer Figure 2). On 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. We thus need to decide which attribute should be parent of A_{11} in the new correlation tree. We choose the attribute with which A_{11} has higher correlation. This is done because in the second stage of the methodology, the attributes would be allocated grid cells such that they 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 we make the other parent as a sibling of the attribute as shown in Figure 3.

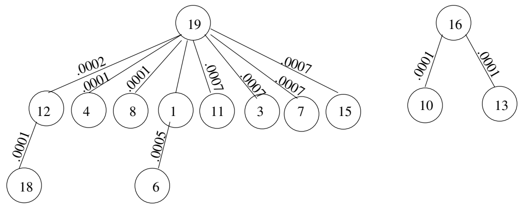

Figure 3. Partial Correlation after adding query Q_{17}

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 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,

$(v,A_y,A_z) = v$ if A_y and A_z are siblings

= 0 otherwise

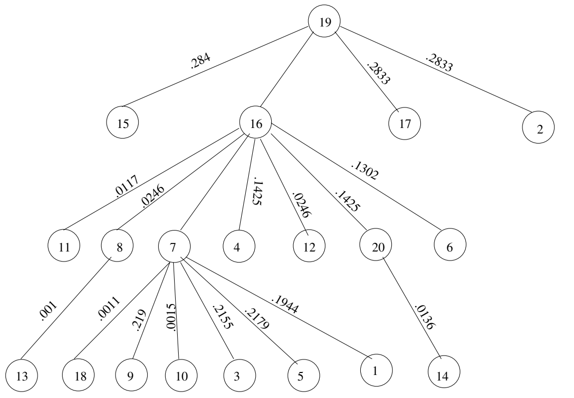

Figure 4. Correlation Tree

The final correlation tree for the set of queries in Table 1 is shown in Figure 4. The virtual weights have not been shown to ensure clarity of the figure.

4.2 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 more frequently accessed attributes are closer to the sink and the attributes with higher correlations are stored closer to each other. In this paper, we assume that the sink can 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 most easily accessible from any random position in the rectangular 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 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 let us 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 centre. The justification is that the available cells near the centre are left for attributes that need to be stored close to the centre. Finally we consider the case where the attribute has correlations with multiple attributes. In such a situation we try to allocate the attribute to an available grid cell, such that distance of the grid from the attribute with which it has correlation is inversely proportional to its correlation value.

	A_{10}	A_3	A_5	A_{11}
A_{18}	A_8	A_{18}	A_7	A_9
A_{13}	A_2	A_{19}	A_{16}	A_1
	A_{12}	A_{17}	A_{20}	A_6
		A_{14}	A_4	

Figure 5. Allocation of attributes to grid

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 also its attributes A_2, A_{15}, A_{17} and A_{19} stored near the centre adjacent to each other. Also note that some of the grid cells are empty since the number of attributes in our example query set (Table 1) is less than the number of grid cells. These empty unassigned grid cells can be used later for fault tolerance and load balancing. The simulation results demonstrating the enhanced performance of our scheme are given in the next section.

5. SIMULATION RESULTS

We have implemented our proposed scheme and conducted our simulations using Simjava [12], a general discrete event simulator. To compare the performance of our proposed scheme, we choose the aggregation algorithm TAG [4] in which the sensor nodes form a spanning tree in a distributed manner. The parameters used in our study are summarized in Table 3. We have 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 our proposed scheme, we assume that the attribute queried is stored in the central grid cell (i.e. nearest to the sink). To measure the worst case performance, we consider the situation when the sink (which is at the centre of the region) queries an attribute that is stored in a grid cell further away from the centre of the network. We use energy consumption as the metric to compare our proposed scheme with the aggregation tree algorithm of TAG. To measure energy consumption for both the schemes, we assume that only those readings greater than the *hard threshold* are reported to the sink. Further for our proposed scheme we assume that the update messages are sent only when the 'soft' threshold is breached. We have conducted experiments for computing energy consumption against querying rate and attribute fluctuation frequency as described in the following sections.

Table 3. Simulation Parameters

Total number of nodes	1030
Dimensions of the deployed area	630×630 meters
Transmission radius of a node	40 meters
Data packet size	30 bytes
Channel bandwidth	20 kbps
Transmission power	0.81 mW
Reception power	0.3 mW
Number of subregions	49
Length of each subregion	90 meters
Hard threshold	22
Soft Threshold	0.6
Probability of fluctuation	0.3
Magnitude of change	± 1.8

5.1 Test # 1: Varying the query rate

We first keep the rate of attribute fluctuations as constant and vary the query dissemination rate. Figure 6(a) shows results for the best case scenario where the attribute stored at the central grid cell (CR) is accessed by a periodic query. We observe that our scheme shows marginal performance degradation at lower query rates but as the number of queries injected per unit time increases, our scheme performs significantly better than the aggregation tree (AT) algorithm. We reason this as follows. The cost of flooding the query down to the leaf level of the AT and then retrieving the information requires *O(n)* transmissions. On the other hand, in our

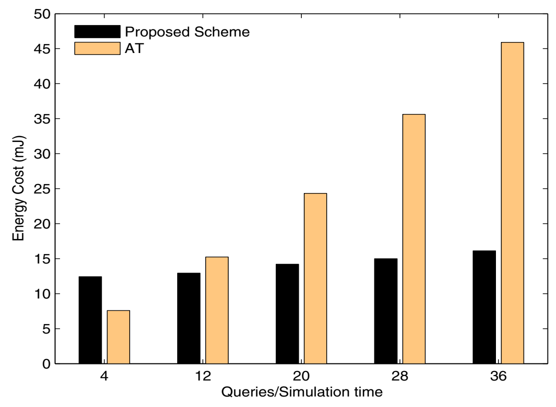

Figure 6(a). Varying Query rate (Best Case)

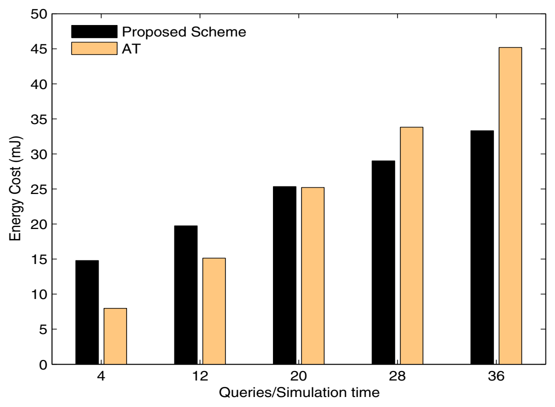

Figure 6(b). Varying Query Rate (Attribute far from sink)

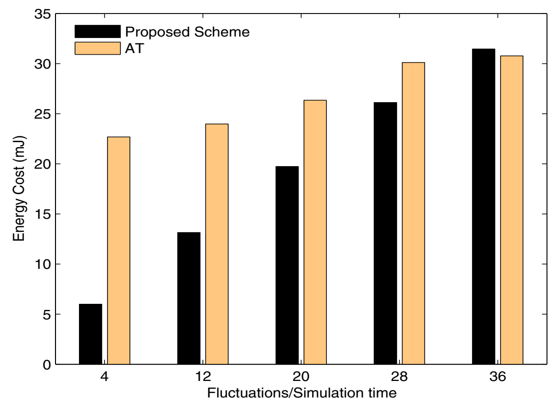

Figure 7(a). Varying Fluctuation rate (Best Case)

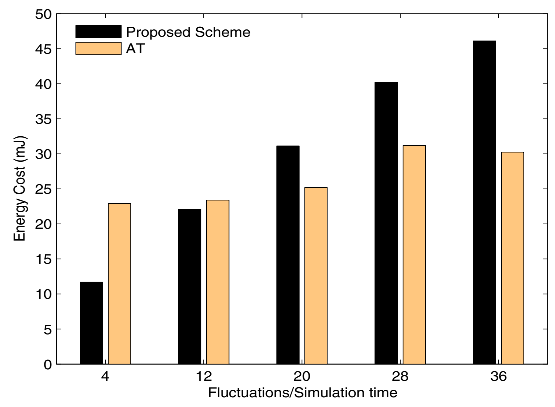

Figure 7(b). Varying Fluctuation Rate (Attribute far from sink)

proposed scheme (and more so for our considered topology), this is accomplished in a single transmission between the control node in the CR and the sink. Thus we reason that that the querying cost is minimized in this configuration. There is however, a constant overhead of updating the storage region based on the nodes which show a variation in the sensed attribute. We note that the AT is preferred when the rate of query injection is less. From figure 6(a), we see that when the query rate is at about 8 queries/simulation time, both the schemes show equal performance for the best case scenario. Figure 6(b) shows results for the scenario where the attribute stored at a grid cell far from the sink is accessed by a periodic query. Here the break-even point is reached when the rate of query is 22 queries/simulation time, after which our scheme performs much better than AT.

5.2 Test # 2: Varying the fluctuation rate

In this experiment (Figures 7(a) and 7(b)), we vary the rate of attribute value fluctuations while keeping a steady query rate of 20 queries/simulation time. We observe that with an increase in the number of fluctuations per unit time, the proposed scheme is no longer preferable to the AT scheme beyond the break-even point. The AT performs at a steady energy cost as nodes do not report until a query message is disseminated. The minor increase in energy cost of AT happens because, with increasing rate of fluctuations, more number of nodes have values greater than the hard threshold and hence send reports back to the sink. On the other hand, the rapid fluctuations result in steady increase in the number of update messages sent from the node detecting fluctuation to the control node of its own cell as well as the aggregated update messages sent to the control node of the grid cell where the attribute is stored. Even then, in the best case scenario (Figure 7(a)), our scheme performs better than AT till a considerable value of 34 fluctuations /simulation time is reached. In the worst case scenario (Figure 7(b)), this point is reached at 10 fluctuations /simulation time. The energy consumed by the AT is seen to be almost constant at 23-30 mJ, independent of the frequency of fluctuations.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we have proposed a scheme for determining a distribution of attributes to a large-scale sensor network based on the correlations between them. Our scheme obviates the need for queries to be flooded in the network and minimizes the average user response time. Moreover our proposed scheme 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, the sensed temperature values could be compared against the average temperature value of the network to detect fires or other local temperature spikes. Also, having user-defined parameters like soft threshold and update epoch allows the user to tune the performance of the system as per his requirements. The proposed scheme works well as long as the overhead of sending update messages does not supersede the advantage of minimizing the query cost. Since most real-life physical phenomena are localized, the fluctuations can be considered to be mostly local and not too many. Consequently the proposed scheme should perform well in most real-life situations.

As part of future work, we plan to develop detailed protocols for query dissemination, data updating and retrieval. We also need to ensure that these protocols are fault-tolerant and perform load balancing.

7. REFERENCES

[1]. Govindan, R., Hellerstein, J. M., Hong, W., Madden, S., Franklin, M. and Shenker, S. *The Sensor Network as a Database*. Technical Report 02-771, USC Computer Science Department, 2002.

[2]. Greenstein, B., Estrin, D., Govindan, R., Ratnasamy, S. and Shenker, S. DIFS: A Distributed Index for Features in Sensor Networks. In *Proceedings of the First IEEE International Workshop on Sensor Network Protocols and Applications*. May 2003, 163-173.

[3]. Intanagonwiwat, C., Govindan, R. and Estrin, D. Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks. In *Proceedings of the Sixth Annual ACM/IEEE International Conference on Mobile Computing and Networking (Mobicom 2000).* Boston, Massachusetts, August 2000.

[4]. Madden, S., Franklin, M. J., Hellerstein, J. M. and Hong, W. TAG: a Tiny AGregation Service for Ad-Hoc Sensor Networks. In *Proceedings of the 5th Annual Symposium on Operating Systems Design and Implementation (OSDI).* Boston, MA, December 2002.

[5]. Li, X., Kim, Y. J., Govindan, R. and Hong, W. Multi-Dimensional Range Queries in Sensor Networks. In *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems.* Nov. 2003.

[6]. 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 *Proceedings of the First ACM International Workshop on Wireless Sensor Networks and Applications (WSNA 2002).* Atlanta, GA, September, 2002.

[7]. Shenker, S., Ratnasamy, S., Karp, B., Govindan, R. and Estrin, D. Data-centric Storage in Sensornets. *ACM SIGCOMM Computer Communication Review*, Volume 33 Issue 1, January 2003.

[8]. Kumar, R., Tsiatsis, V. and Srivastava, M. B. Computation Hierarchy for In-network Processing. In *Proceedings of 2nd ACM International Conference on Wireless Sensor Networks and Applications*. San Diego, CA, September 2003.

[9]. Karp, B. and Kung, H.T. GPSR: Greedy Perimeter Stateless Routing for Wireless Networks. In *Proceedings of Sixth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM).* Boston, Massachusetts, August 2000.

[10]. Doherty, L., Pister, K. S. J. and Ghaoui, L. E. Convex Position Estimation in Wireless Sensor Networks. In *Proceedings of the IEEE Infocom.*. Alaska, April 2001, 1655–1663.

[11]. HighTower, J. and Borreillo, G. Location systems for Ubiquitous Computing. *IEEE Computer, Vol. 34* (Aug 2001), 57-66.

[12]. Howell, F. and McNab, R. Simjava: A Discrete Event Simulation Package for Java with Applications in Computer Systems Modelling. In *Proceedings of the First International Conference on Web-based Modelling and Simulation.* Society for Computer Simulation, San Diego, CA, Jan 1998.