# Data-Centric Attribute Allocation and Retrieval (DCAAR) Scheme for Wireless Sensor Networks

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

*Abstract* – **Wireless sensor networks have enabled information gathering from a large geographical region and present unprecedented opportunities for a broad spectrum of monitoring applications. In this paper, we propose a data-centric storage scheme to determine a distribution of attributes over a large-scale sensor network such that the cost of retrieving data is minimized. 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.**

***Keywords – Allocation problem, data-centric storage, aggregation, in-network processing, minimum spanning tree, heap, hard threshold, soft threshold, query-driven, event-driven***

## I. INTRODUCTION

Recent technological advances have enabled distributed information gathering from a given region by deploying a large number of networked tiny wireless sensor nodes. However the small form factor of these nodes limits the size of the battery or the total power available with each sensor node. Since data communication is the dominant component of energy consumption at the node level, protocol design for sensor networks is geared towards reducing communication in the network. Thus we propose a data-centric storage scheme for storing attributes in a sensor network such that communication involved in querying the network can be minimized. A data-centric storage scheme [1][6][7][8] allows events to be stored at specific rendezvous points within the network that queries can access directly. In this paper, we have proposed the DCAAR scheme that determines an allocation of attributes depending on the correlations between them. We have analytically 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 II describes a general scenario where the proposed scheme can be used and lists the basic assumptions for the scheme. Section III defines the allocation problem and gives an outline of the proposed DCAAR scheme. The detailed algorithms are given in Section IV. Section V determines analytically the conditions under which the DCAAR scheme should be preferred. The simulation results are discussed in Section VI. Section VII lists other data-centric storage schemes, while Section VIII concludes the paper with future research agenda.

## II. CONTEXT

In this section we first present a general scenario where the DCAAR scheme can be applied and list some basic assumptions about the sensor networks being considered.

### A. A General Scenario

We have designed a scheme that can be employed for large scale sensor networks of the future. For example, given a geographical area, the user might want to deploy a sensor network that simultaneously serves multiple applications like precision agriculture, weather monitoring, ecosystem monitoring etc. For each application the user might 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 might expect the sensor network to respond to specific 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 DCAAR scheme is designed for minimizing cost of data retrieval from large-scale sensor networks serving such real-life scenarios.

### B. Basic Assumptions

In a sensor network, the nodes which issue user queries are called sink nodes. We have assumed that the sink node could be at any random position in the network. Queries issued by a sink may be classified into two broad categories:

- *Location-based query* is used when the user is interested in attribute values from only a particular region of the network and hence the query is routed directly to that region.
- *Attribute-based query* is used when the user is interested in data satisfying some attribute-based selection criteria. Such queries are flooded in the entire network and the relevant nodes route their data to the user using aggregation schemes [4][5].

The motivation behind the DCAAR scheme is to obviate the need for communication-extensive flooding in case of attribute-based queries. The sensor nodes are assumed to be location-aware [3]. We also assume that the sensor network has been deployed on a rectangular region and the nodes are aware of the geographical boundaries of the network.

## III. The Allocation Problem

Sensor networks can be envisioned as a large distributed database where the sensor nodes generate named data against user-specified queries. In this section, we define the Allocation Problem of distributed databases in the context of sensor networks and present our proposed architecture.

### A. Problem Formulation

We define the Allocation Problem for 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 $Q$. We propose the DCAAR scheme as a heuristic solution for the sensor network allocation problem.

We formulate the allocation problem as follows. Let $Q = \{Q_1, Q_2, \ldots, Q_q\}$ be a set of user-defined queries. We assume that a set of priorities $\{p_1, p_2, \ldots, p_q\}$ for the queries $\{Q_1, Q_2, \ldots, Q_q\}$ 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 $Q_k$ can be modeled by the set of attributes included in the query i.e. $Q_i = \{A_{i_1}, A_{i_2}, \ldots, A_{i_k}\}$. As mentioned in Section II.B., the sensor nodes are assumed to be uniformly deployed over a rectangular region. Thus our objective is to distribute the $m$ attributes over this rectangular field of sensor nodes. We do so by splitting 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. The allocation problem thus reduces to finding an optimal distribution of $m$ attributes over a $\lceil\sqrt{m}\,\rceil \times \lceil\sqrt{m}\,\rceil$ grid. Let $f : A \rightarrow G$ be an optimal distribution function. Thus $f(A_k) = (x_k, y_k), x_k \in \{1, \ldots, \lceil\sqrt{m}\,\rceil\}, y_k \in \{1, \ldots, \lceil\sqrt{m}\,\rceil\}$ 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 total query access cost $C_A$ ($\sum_{i=1}^{q} C_A^i$) for the set of queries $Q$.

### B. DCAAR Scheme

To determine a good distribution function $f : A \rightarrow G$ we employ a two-stage heuristic. Note that queries and their associated priorities indirectly define correlations between attributes. For example, if a query with a very high priority includes attributes $A_i$ and $A_j$ we can say that $A_i$ and $A_j$ have a high correlation since they are accessed together very frequently. Attributes with higher correlations should be placed closer to each other for minimizing the cost $C_E$. Similarly by placing more frequently accessed attributes closer to the center (since it is equally likely for the sink to be in any position), the cost $C_A$ can be minimized. To identify the frequently accessed attributes, we calculate the individual access probabilities $P(A_i)$ of attributes as follows:

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

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

The first stage of the DCAAR scheme creates a heap-like data structure called the *correlation tree* that provides a holistic view of all the attributes in terms of their access probabilities and correlations. The second stage uses this correlation tree to determine an optimal distribution of the attributes to respective cells of the grid. As an illustration, Table I lists a set of given queries $Q_i$ along with their priorities. The corresponding correlation tree and the optimal allocation of attributes are shown in Fig. 1. Note that since query $Q_7$ has the highest priority, attributes $A_2$, $A_{35}$, $A_{37}$, and $A_{39}$ have been allocated storage cells nearest to the center of the grid and adjacent to each other. The algorithms for determining the function $f$ are described Section IV. Since the algorithms are 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 present our proposed architecture for query processing.

### C. Proposed Architecture

Every sensor node is assumed to be aware of its own location as well as the geographical boundaries of the sensor network. Hence every sensor node can identify which grid cell it belongs to. We call the grid cell responsible for storing attribute $A_i$ as the zone for attribute $A_i$ and denote it by $Z_i$. The number of nodes belonging to a zone $Z_i$, denoted by $N(Z_i)$, depends on the density and distribution of the sensor nodes over the network. Some of the nodes belonging to each zone have special functionalities apart from sensing and routing. Based on the functionalities assigned to them, these nodes can be classified into 3 types viz. control node, storage nodes and replica nodes. The storage nodes of a zone $Z_i$ are responsible for storing the values of the attribute $A_i$. The number of storage nodes required in a zone $Z_i$ is denoted by $N(A_i)$ and depends on the total amount of data values corresponding to attribute $A_i$ and the memory capacity of each sensor node. For every zone $Z_i$, there is one control node that is responsible for fetching data from the storage nodes using specialized indexes that it maintains. Apart from the control nodes and storage nodes, some of the other nodes in a zone may store redundant or summary information about attribute values to provide fault tolerance and are hence called the replica nodes. Fig. 1(b) shows the proposed architecture where the control node is the node nearest to the center of each zone.

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 determines which zone is nearest to itself, say $Z_{i_j}$, and 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.
- The control node of zone $Z_{i_j}$ also computes the optimal route of disseminating the query to the zones storing the remaining 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 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 will not contain a large number of attributes.
- The computed route is used for routing the query and routing the resultant data tuples back to the sink. While routing the data to the sink, the control nodes of the respective storage zones use aggregation [4][5] and in-network processing mechanisms [9] to further reduce the amount of data transfer in the network.

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 this maintenance overhead, a soft threshold scheme may be used:

- Suppose a sensor node in zone $Z_i$ senses an attribute $A_k$. Suppose the soft threshold for attribute $A_k$ is $\varphi_k$.

  If the difference between the previously sensed value and the current value of $A_k$ is more than $\varphi_k$, 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 $Z_i$.

TABLE I. LIST OF QUERIES $Q_i$ AND ASSOCIATED PRIORITIES $p_i$

| $Q_i$ | $p_i$ | $A_{i_k}$ | $Q_i$ | $p_i$ | $A_{i_k}$ | $Q_i$ | $p_i$ | $A_{i_k}$ |
|---|---|---|---|---|---|---|---|---|
| $Q_1$ | .1313 | $A_1$,$A_3$,$A_5$,$A_7$,$A_9$ | $Q_{13}$ | .0510 | $A_{39}$ | $Q_{25}$ | .1017 | $A_8$ |
| $Q_2$ | .0082 | $A_{11}$,$A_{13}$,$A_{15}$,$A_{17}$ | $Q_{14}$ | .0161 | $A_3$,$A_7$ | $Q_{26}$ | .0136 | $A_{14}$,$A_{20}$,$A_{26}$ |
| $Q_3$ | .0003 | $A_{19}$,$A_{22}$,$A_{23}$ | $Q_{15}$ | .0005 | $A_{11}$ | $Q_{27}$ | .0001 | $A_{32}$,$A_{38}$ |
| $Q_4$ | .0170 | $A_{25}$,$A_{27}$,$A_{29}$ | $Q_{16}$ | .0189 | $A_{15}$ | $Q_{28}$ | .0007 | $A_5$ |
| $Q_5$ | .0005 | $A_{31}$,$A_{33}$ | $Q_{17}$ | .0007 | $A_{19}$,$A_{23}$,$A_{27}$,$A_{31}$,$A_{35}$ | $Q_{29}$ | .0001 | $A_{12}$,$A_{19}$ |
| $Q_6$ | .0017 | $A_{35}$ | $Q_{18}$ | .0001 | $A_4$,$A_8$,$A_{12}$,$A_{39}$ | $Q_{30}$ | .0001 | $A_{26}$ |
| $Q_7$ | .2833 | $A_2$,$A_{35}$,$A_{37}$,$A_{39}$ | $Q_{19}$ | .1179 | $A_{16}$,$A_{20}$,$A_{24}$ | $Q_{31}$ | .0681 | $A_{34}$ |
| $Q_8$ | .0063 | $A_5$,$A_8$,$A_1$ | $Q_{20}$ | .0010 | $A_{28}$,$A_{33}$ | $Q_{32}$ | .0164 | $A_8$,$A_{16}$,$A_{24}$,$A_{32}$,$A_{40}$ |
| $Q_9$ | .0015 | $A_{14}$,$A_{17}$ | $Q_{21}$ | .0008 | $A_{36}$ | $Q_{33}$ | .0008 | $A_9$,$A_{18}$,$A_{27}$,$A_{40}$ |
| $Q_{10}$ | .0044 | $A_{20}$ | $Q_{22}$ | .0117 | $A_1$,$A_6$,$A_{11}$,$A_{16}$ | $Q_{34}$ | .0015 | $A_6$,$A_{16}$,$A_{36}$ |
| $Q_{11}$ | .0015 | $A_4$,$A_{27}$,$A_{30}$ | $Q_{23}$ | .0005 | $A_{21}$,$A_{26}$,$A_{31}$ | $Q_{35}$ | .1180 | $A_{26}$,$A_{36}$ |
| $Q_{12}$ | .0001 | $A_{33}$,$A_{36}$ | $Q_{24}$ | .0001 | $A_{10}$,$A_{36}$ | $Q_{36}$ | .0035 | $A_7$ |

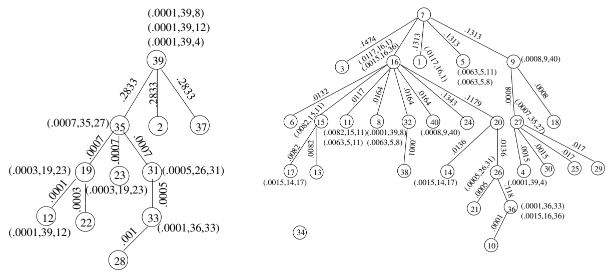


Figure 1(a). Correlation Tree corresponding to queries in Table I.

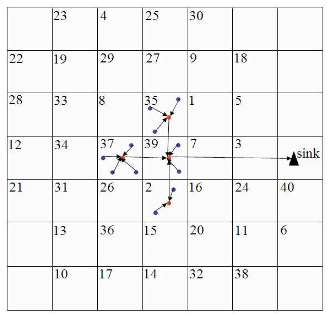


Figure 1(b). Allocation of attributes to grid

- 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 $A_k$.

All the messages (viz. query message, update message or resultant data tuple) can be routed in the network using geographic routing protocols like GPSR [2].

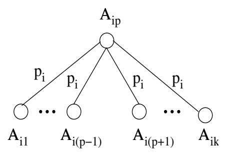


Figure 2. Shows initial tree for query $Q_i = \{A_{i_1} A_{i_2} \ldots A_{i_k}\}$.

### D. *Advantages and Limitations*

As with all data-centric storage schemes, the DCAAR scheme obviates the need for flooding queries in the network. The user response time for queries is also minimized. Moreover DCAAR minimizes both query access cost and 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 allows 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 but 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.

## IV. Algorithms

In this section we present the algorithms *CreateCorrelationTree* and *AllocateGrid* that have been employed in the two stages of the DCAAR scheme.

### A. *Algorithm CreateCorrelationTree*

Input: List of queries $Q_i$ and associated priorities $p_i$
Output: Correlation Tree/ Forest

This is the first stage of the heuristic which creates a heap-like data structure called the correlation tree to represent the correlations between attributes as well as the relative ordering of attributes in terms of their access probabilities. The initialization phase of the algorithm constructs a tree for each query $Q_i$ as shown in Fig. 2. The query attribute with highest access probability is made the root of the query tree. Thus for query $Q_i$ in Fig. 2, $A_{i_p}$ has highest probability i.e. $P(A_{i_p}) = \max\{P(A_{i_1}), P(A_{i_1}), \ldots, P(A_{i_k})\}$. The weight of an edge in tree $T$ joining the attributes $A_i$ and $A_j$ is denoted by $w_T(A_i,A_j)$ and depicts the probability of $A_i$ and $A_j$ being in the same query. At each iterative phase, a query tree $T$ is chosen from this initial list in ascending order of query priorities, and combined with the partial solution *PS* (i.e. the partial correlation tree/forest at that iteration phase) as shown in the pseudocode. The trees are combined using usual tree union algorithms. The case that requires special attention is when an attribute $A_x$ has different parents in *PS* and *T* and thus algorithm *AdjustCorrelationTree* is invoked.

```
PSEUDOCODE : CreateCorrelationTree
Sort query trees in ascending order of their query probabilities
Choose query tree T = query tree with lowest query probability
PS = T
 // initial partial solution = query tree with lowest query probability
For ( each remaining query tree T from sorted list of query trees ) do
{     If  ( (attributes of T) $\cap$ (attributes in PS) = $\phi$) then
        {  // no attribute of T is present in PS
           PS = PS +T
           // add query tree T to existing partial solution PS  }
      Else
      {   // some of the attributes in T are present in partial solution PS
           $A_p$ = root of T
           If  ( $A_p \notin$ PS ) then  // $A_p$ is not present in partial solution PS
           {    PS = {$A_p$} + PS  // add $A_p$ to PS as a single node tree  }
           For  ( each child node $A_x$ in T ) do
             {    If  ( $A_x \notin$ PS ) then
                   {   Add $A_x$ as child of $A_p$ in PS
                       Set $w_{PS}(A_p,A_x)$= $w_T(A_p,A_x)$  }
                 Else  // if $A_x$ is present in PS
                 {     If  ( $A_x$ has no parent in PS ) then
                        {  Set $A_x$ as child of $A_p$ in PS
                           Set $w_{PS}(A_p,A_x)$ = $w_T(A_p,A_x)$  }
                       Else  // $A_x$ has parent in PS
                       {    If  ( parent of $A_x$ in PS is also $A_p$ ) then
                             {   $w_{PS}(A_p,A_x)$= $w_{PS}(A_p,A_x)$+$w_T(A_p,A_x)$.
                                 // increase corresponding edge weight  }
                            Else  // $A_x$ has different parent in PS
                            {     $A_q$ = parent of $A_x$ in PS
                                  Call PS = AdjustCorrelationTree(PS,T,$A_x$,$A_p$,$A_q$)  }
}   }    }    }    }
Return PS  // PS is the final correlation tree/forest
```

### B. *Algorithm AdjustCorrelationTree*

Input: *PS* - Partial solution, *T* - Query tree to be added, $A_x$ - Attribute present in both *PS* and *T*, $A_p$ - Parent of $A_x$ in *T*, $A_q$ - Parent of $A_x$ in *PS*.
Output: *PS* – New partial solution

This algorithm makes the parent ($A_p$ or $A_q$) that has a higher correlation (i.e. edge weight) with $A_x$ as the parent of $A_x$ in the new partial solution, while the other parent is made a sibling of $A_x$. We reason as follows. Algorithm *AllocateGrid* always stores an attribute as close to its parent as possible and thus $A_x$ is made child of the parent with which it has higher correlation. Also since algorithm *AllocateGrid* stores every child attribute near its parent, sibling attributes also end up being stored fairly close to each other in the grid. This justifies making the other parent a sibling of $A_x$ in the new partial solution. A triplet of values called the virtual weight is assigned to both attributes to signify that they have a correlation even though they do not share a parent-child relationship in the tree (Lines 9,17). Note that the heap property of the correlation tree should be preserved at all times i.e. if attribute $A_i$ is the parent of $A_j$, then $P(A_i) \geq P(A_j)$ (Lines 4,12). If necessary, the algorithm *Bubble* (similar to a usual heap-creation algorithm) is invoked to adjust the tree so as to maintain the heap property (Lines 7,15). Before deleting any edge, a cost-benefit analysis is performed to ensure that benefit > cost, where cost and benefit are defined as follows,

Cost = Sum of effective weights of deleted edges (2)

Benefit = Sum of effective weights of new edges (3)

where effective weight $w'_T(A_i, A_j)$ of an edge $(A_i, A_j)$ is,

$w'_T(A_i, A_j) = w_T(A_i,A_j) + \Sigma\{$virtual weights of $A_j\}$ (4)

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 (5)

```
PSEUDOCODE : AdjustCorrelationTree
Line 1.   If ( $w_T(A_p,A_x) > w'_{PS}(A_q,A_x)$ ) then
Line 2.   {  // $A_x$ has more correlation with $A_p$ than with $A_q$
Line 3.      Make $A_x$ child of $A_p$ and set $w_{PS}(A_p,A_x)= w_T(A_p,A_x)$
Line 4.      If ( $P(A_p) \geq P(A_q)$ ) then
Line 5.      {   Make $A_q$ child of $A_p$ provided benefit>cost }
Line 6.      Else
Line 7.      {   Call TempTree = Bubble(PS,$A_p$,$A_q$)
Line 8.          PS = TempTree provided benefit>cost }
Line 9.      Add virtual weight ($w_{PS}(A_q,A_x)$, $A_q$,$A_x$) to $A_q$ and $A_x$ }
Line 10.  Else // $A_x$ has more correlation with $A_q$ than with $A_p$
Line 11.  {  // $A_x$ remains child of $A_q$ in the new partial solution
Line 12.      If ( $P(A_q) \geq P(A_p)$ ) then
Line 13.      {   Make $A_p$ child of $A_q$ provided benefit>cost }
Line 14.      Else
Line 15.      {   Call TempTree = Bubble(PS,$A_p$,$A_q$)
Line 16.          PS = TempTree provided benefit>cost }
Line 17.      Add virtual weight ($w_{PS}(A_p,A_x)$, $A_p$,$A_x$) to $A_p$ and $A_x$ }
```

### C. Algorithm AllocateGrid

Input: *CT* - Correlation tree created by *CreateCorrelationTree*
List of attributes $A_i$ with access probabilities $P(A_i)$
Output: Allocation of attribute set *A* to grid *G*

```
PSEUDOCODE : AllocateGrid
Line 1.  Sort attributes in descending order of access probabilities
Line 2.  Choose $A_x$ = attribute with highest probability
Line 3.  Assign $A_x$ to central-most grid cell
Line 4.  For ( each remaining attribute $A_x$
               from sorted list of attributes ) do
Line 5.  {   If ( $A_x$ has no parent in CT ) then
Line 6.       {  opt_list = Call FindNearestCells(Center)
Line 7.          zone$_x$ = Call FindMinChildren(opt_list)
Line 8.          Assign $A_x$ to grid cell zone$_x$ }
Line 9.      Else // $A_x$ has parent in CT
Line 10.     {  $A_p$ = parent of $A_x$ in CT
Line 11.        If ( $w_{CT}(A_p,A_x) = P(A_x)$ )
Line 12.        {  zone$_p$ = Assigned grid cell of $A_p$
Line 13.           opt_list = Call FindNearestCells(zone$_p$)
Line 14.           opt_list=Call FindFarthestCenter(opt_list)
Line 15.           zone$_x$ = Call FindMinChildren(opt_list)
Line 16.           Assign $A_x$ to grid cell zone$_x$ }
Line 17.        Else
Line 18.        {  // $A_x$ has correlations with other attributes also
Line 19.           attr_list = Call FindCorrelationAttributes($A_x$)
Line 20.           attr_corr_list = Call FindCorrelations($A_x$)
Line 21.           attr_cell_list = Call FindStorageCells(attr_list )
Line 22.           opt_list = Call FindOptionCells(attr_cell_list)
Line 23.           zone$_x$=Call FindBestCell(opt_list, attr_cell_list,
                                               attr_corr_list)
Line 24.           Assign $A_x$ to grid cell zone$_x$ }
Line 25.}   }
```

This is the second stage of the DCAAR scheme. This algorithm allocates attributes to grid cells while trying to preserve the correlations between attributes as represented by the correlation tree. The algorithm begins with the attribute $A_x$ having the maximum access probability and allocates $A_x$ to the central-most grid cell. It then iteratively assigns attributes in descending order of their access probabilities as shown in the pseudocode (Lines 4-25). For each attribute $A_x$, the algorithm finds out from the correlation tree *CT* the attributes that $A_x$ has correlations with and then determines its optimal position in the grid. If the $A_x$ has no parent in *CT*, it is placed as close to the center as possible (Lines 5-8). However there may be multiple available grid cells at the same distance from the center. In that case, the optimal grid cell is the one for which adjacent already-allocated attributes have minimum number of unassigned children (Line 7). 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. Next the algorithm handles the case where the correlation that $A_x$ has with its parent $A_p$ is the same as its access probability $P(A_x)$ (Line 11-16). This implies that any query that accesses $A_x$ also accesses $A_p$ and thus $A_x$ should be stored close to $A_p$. Finally the algorithm considers the case where $A_x$ has correlations with multiple attributes (Lines 17-24). The algorithm *FindBestCell* returns the optimal cell for assigning $A_x$ by scanning the list *opt_list* for the cell which minimizes the value $\Sigma\{\text{distance}(opt\_list[i],attr\_cell\_list[j])*attr\_corr\_list[j])|\forall j\}$.

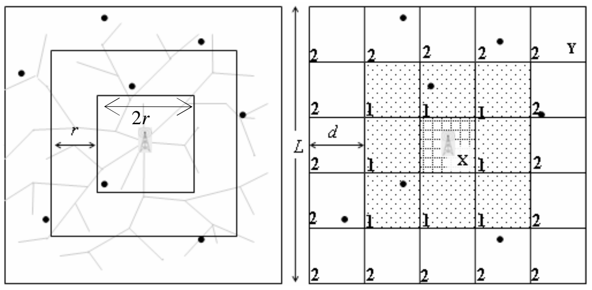
The complexity of the algorithms *CreateCorrelationTree* and *AllocateGrid* is *O(mlogm)*, since *CreateCorrelationTree* is similar to creating a heap of *m* attributes while *AllocateGrid* is similar to traversing the heap.

## V. Analysis

The energy consumption in DCAAR scheme is compared with that of TAG [5], where an aggregation tree spanning all the sensor nodes is formed in a distributed manner. For both the schemes, we assume that only those readings greater than the hard threshold $\Phi$ are reported to the sink. Further for the DCAAR scheme we assume that the update messages are sent only when the soft threshold $\varphi$ is crossed. We assume that the sink is at the centre of the network as shown in Fig. 3. We also assume 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 II.

TABLE II. 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 |
| $\Phi$ | Hard Threshold |
| $\varphi$ | Soft Threshold |
| $\xi$ | 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 > $\varphi$ |
| $D$(sk,stg) | Distance between sink and storage region |
| $\mu$ | Frequency of attribute value crossing $\Phi$ |
| $\mu'$ | Frequency of attribute value crossing $\varphi$ |
| $f_Q$ | Query Frequency |



(a) (b)

Figure 3. Aggregation Tree and DCAAR

## A. *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 spreads outwards from the center of the region. For best case performance, we assume that the AT spans the region uniformly. We divide the entire area into *n* concentric square regions, each of width *r* as shown in Fig. 3(a). The nodes present in the $i^{th}$ concentric region are assumed to be at a depth *i* in the AT. Thus *n* is the depth of the tree where $2nr = L$.

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

Thus, the probability that a node lies in region *i* is given by $p(i) = \frac{A(i)}{L^2}$. Similarly, out of *N* nodes, the number of such nodes in the $i^{th}$ region is given by $Np(i)$. Thus the energy cost incurred at each level is $E(i) = I(i) \times \xi \times P_{tx}$ and the total energy is given by: $E_{Data(AT)} = \sum_{i=1}^{n} E(i)$. 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* from level *i+1*. We thus obtain a recursive relation as:

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

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

Thus the total incurred communication cost:

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

Through algebraic calculations,

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

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

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

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

$$E_{Query(AT)} = N_{total} P_{tx} \tag{7}$$

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

$$E_{Total}(AT)_P = f_Q\left(E_{Query(AT)} + E_{Data(AT)}\right) \tag{8}$$

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

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

## B. *Energy cost of DCAAR Scheme*

Let us assume that the storage region X in Fig. 3(b) 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 $i^{th}$ 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,

$p(1) = \frac{8d^2}{L^2}$, $p(2) = \frac{16d^2}{L^2}$ … up to $\lfloor L/2d \rfloor$ tiers.

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

$N'$ 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,

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

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,

$$E(i) = \frac{8iN'd^2}{L^2}\left(\frac{di}{r}\right)\xi P_{tx} \tag{9}$$

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,

$$E_{Data(DS)} = \sum_{i=1}^{L/2d} E(i) = \frac{8N'd^3}{rL^2}\xi P_{tx}\sum_{i=1}^{L/2d} i^2$$

$$= \sum_{i=1}^{L/2d} E(i) = \frac{2N'd^3}{3rL^2}\xi P_{tx}\left[(L/d)(L/2d+1)(L/d+1)\right] \tag{10}$$

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

The consequent reply from the storage region to the sink incurs a cost of $E_{Reply(DS)} = \frac{D(\mathrm{sk},\mathrm{stg})}{r}N\xi P_{tx}$

The total energy expended in our scheme is given by,

$$E_{Total}(DS) = f_Q\left(E_{Query(DS)} + E_{Reply\,(DS)}\right) + \left(E_{Data(DS)} + E_{Update}\right)\mu' \tag{11}$$

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

$$f_Q\left(E_{Query(DS)} + E_{Reply(DS)}\right) < \begin{cases} f_Q\left(E_{Query(AT)} + E_{Data(AT)}\right) & \text{(Query-driven)} \\ + (E_{Data(DS)} + E_{Update})\mu' \quad E_{Data(AT)}\mu & \text{(Event-driven)} \end{cases} \tag{12}$$

To validate our models of the aggregation tree and our proposed scheme, we simulated a testbed of 1030 nodes distributed randomly in an area of $630 \times 630$ square units. A node at (x,y) was 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 allowed 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 was $\pm 1.8$ and 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 were over $\Phi = 22$ averaged at 380. The other constant simulation parameters are as described in Section VI. The calculated and observed values are summarized in Table III and are in good agreement thus validating our model. As predicted by Equation (12), DCAAR has lower energy cost under these conditions and is the preferred choice.

## VI. Simulation Results

Using Simjava, a discrete event simulator, we have conducted simulations to compare our DCAAR scheme with the aggregation tree (AT) algorithm of TAG [5].

### A. *Simulation Environment*

In our study, we dispersed 1030 nodes randomly in a square area of $630 \times 630$ units, each with a transmission range of 40 units to form a connected network. Each packet of 30 bytes is transmitted over a 20 kbps channel, incurring a cost of 0.81mW and 0.3mW 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. We measured the energy consumed by all the nodes for different topologies. We have measured this by varying both the rate at which queries are injected into the network by the sink and the rate at which the sensed attribute shows a variation in its magnitude. The control nodes in individual subregions, as well as and the nodes in the aggregation tree try and pack all available data in the fewest possible packets thus maintaining an economy of transmission. For both schemes, the sink is assumed to be at the center of the network. To evaluate the best case of our scheme, we assume that the attribute queried is stored in the central grid cell (near 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.

TABLE III. Energy Cost in mJ

| DCAAR | | Aggregation Tree | |
|---|---|---|---|
| *Simulation* | *Calculated* | *Simulation* | *Calculated* |
| 19.7311 | 19.5864 | 26.351 | 21.073 |

### B. *Effect of Query Rate on Performance*

We first keep the rate of fluctuations constant and vary the query rate. From Fig. 4(a), we observe that 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. We reason 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 our proposed scheme (and more so for our 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 energy cost. There is however, 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 energy cost. Fig. 4(b) 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 8 in the best case) after which DCAAR performs much better than AT.

### C. *Effect of Fluctuation Rate on Performance*

In this experiment (Figs. 4(c) and 4(d)), we vary the rate of fluctuations while keeping a steady query rate of 20 queries/simulation time. We observe 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 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 (Fig. 4(c)), the DCAAR scheme performs better than AT till the fluctuation rate reaches a considerable value of 26 fluctuations /simulation time.

## VII. 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). Shenker et al. [8] proposed the Data-Centric Storage scheme and have shown that DCS outperforms other approaches such as External Storage and Local Storage under certain circumstances. Ratnasamy et al. [7] have also proposed a Geographic Hash Table (GHT) to hash events into geographic coordinates. In DIFS [1], Greenstein et al. have designed a spatially distributed index to facilitate range searches over attributes, while Li et al. [6] have built a distributed index (DIM) for multidimensional range queries of attributes.

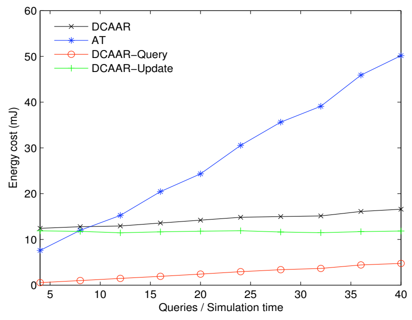


Figure 4(a). Effect of Query rate (Best Case)

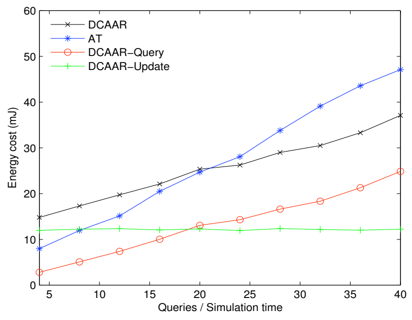


Figure 4(b). Effect of Query Rate (Attribute far from sink)

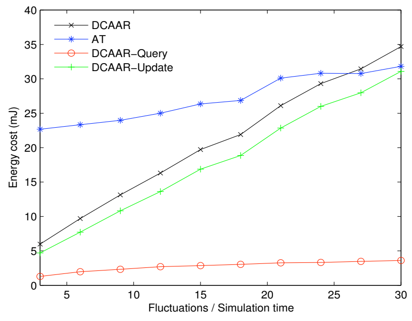


Figure 4(c). Effect of Fluctuation rate (Best Case)

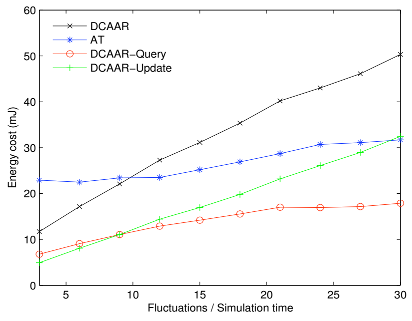


Figure 4(d). Effect of Fluctuation Rate (Attribute far from sink)

In this paper, we have proposed a data-centric storage scheme that differs from the existing data-centric approaches [1][6][7] in that it distributes attributes instead of specific user-defined events. We reason as follows. If an attribute is included in many events, then 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.

## VIII. 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. We have proposed a communication architecture that minimizes the cost of maintaining and retrieving data from such a system and have determined analytically the conditions under which the architecture should be preferred. We have conducted simulations that compare DCAAR performance with other aggregation schemes. 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] B. Greenstein, D. Estrin, R. Govindan, S. Ratnasamy, and S. Shenker, “DIFS: a distributed index for features in sensor networks,” *First IEEE. International Workshop on Sensor Network Protocols and Applications*, pp. 163-173, May 2003.

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

[3] L. Doherty, K. S. J. Pister and L. E. Ghaoui, “Convex Position Estimation in Wireless Sensor Networks,” *IEEE Infocom*, pp. 1655–1663, Alaska, April 2001.

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

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

[6] X. Li, Y. J. Kim, R. Govindan and W. Hong, “Multi-dimensional range queries in sensor networks,” *1st international conference on Embedded networked sensor systems*, Nov. 2003.

[7] S. Ratnasamy, B. Karp, L. Yin, F. Yu, D. Estrin, R. Govindan, and S. Shenker, “GHT: A Geographic Hash Table for Data-Centric Storage,” *First ACM International Workshop on Wireless Sensor Networks and Applications (WSNA 2002),* Atlanta, GA, September, 2002.

[8] S. Shenker, S. Ratnasamy, B. Karp, R. Govindan and D. Estrin, “Data-centric storage in sensornets,” *ACM SIGCOMM Computer Communication Review,* Volume 33 Issue 1, January 2003.

[9] Ram Kumar, Vlasios Tsiatsis, Mani B. Srivastava, “Computation Hierarchy for In-network Processing,” *2nd ACM International Conference on Wireless Sensor Networks and Applications*, San Diego, CA, September 2003.