Multi-root, Multi-Query Processing in Sensor Networks*

Zhiguo Zhang, Ajay Kshemkalyani, and Sol M. Shatz

Department of Computer Science, University of Illinois at Chicago, Chicago, Illinois 60607

Abstract. Sensor networks can be viewed as large distributed databases, and SQL-like high-level declarative languages can be used for data and information retrieval. Energy constraints make optimizing query processing particularly important. This paper addresses for the first time, multi-root, multi-query optimization for long duration aggregation queries. The paper formulates three algorithms - naive algorithm (NMQ), which does not exploit any query result sharing, and two proposed new algorithms: an optimal algorithm (OMQ) and a heuristic (zone-based) algorithm (ZMQ). The heuristic algorithm is based on sharing the partially aggregated results of pre-configured geographic regions and exploits the novel idea of applying a grouping technique by using the location attribute of sensor nodes as the grouping criterion. Extensive simulations indicate that the proposed algorithms provide significant energy savings under a wide range of sensor network deployments and query region options.

1 Introduction

One way to extract sensor data from a distributed sensor network is by using mobile agents that selectively visit the sensors and incrementally fuse appropriate measurement data [12]. Another technique, which is the subject of this paper, is to inject queries into the network, treating the sensors as a distributed database [4]. To reduce energy use associated with communication while gathering data, in-network aggregation [8] can be used, in addition to special network routing to minimize messages needed for query processing [1,6]. For example, we previously proposed a grouping technique based on query-informed routing to make in-network aggregation more energy efficient [13]. The sensors are programmed through declarative queries in a variant of SQL. The following is an example query for monitoring the radiation in a nuclear power plant:

SELECT room, AVG(radiation) FROM sensordb WHERE building = ERF GROUP BY room HAVING AVG(radiation) > 100 DURATION 30 days EVERY 1 minute

* This material is based upon work supported by the U.S. Army Research Office under grant number W911NF-05-1-0573.

Most previous research on query processing in sensor networks has focused on the processing of a single long-running aggregation query (see, for example [8,13]). As an extension to this line of research, Trigoni et al. [11] and Emekci et al. [3] considered the case of reducing message transmission by sharing sensor readings for multiple queries, where queries are represented by particular query regions. A query region is the geographical region a query is interested in retrieving information from. For example, for the query SELECT AVG(temperature) FROM sensorDB where $position.X \geq 25$ and $position.X \leq 75$ and $position.Y \geq 25$ and $position.Y \leq 75$ DURATION 1 day EVERY 1 minute, the region (25, 25)(75, 75) is the query region.

These existing multi-query processing techniques work for centralized environments only, and they require that the queries arrive at a common root node. In this case, since all query regions are known by one root node, intersection regions (the intersection areas among query regions) can be computed at the root node, making it possible to share partially aggregated results of intersection regions. Since these methods use centralized computation of intersection regions at the root node, they are not directly suitable for multiple queries injected from different root nodes. In addition, although existing methods attempt to share the sensor value readings of intersection regions, they do not account for the effect of the routing structure on the efficiency of aggregation, and do not address the problem of how to group sensor nodes according to query regions. As a result, many intermediate nodes need to unnecessarily wake up and transfer messages for sensor value readings of nodes that lie in the same query region but belong to different queries. As pointed out in [13], the routing tree structure of sensor networks can have significant impact on the aggregation efficiency of data retrieval in sensor networks. Therefore, using query information in the construction of a routing tree can provide improvement by reducing message transmission.

In this paper, we formulate and address for the first time the problem of multi-root, multi-query processing for long duration aggregation queries. This problem arises in many applications where loosely-coupled, or independent, stakeholders want to gather information from a common (shared) sensor network. As a specific example, consider a case of environmental monitoring, where scientists studying wildlife migration and climatologists studying pollution patterns are operating from different locations, but both need to monitor average rainfall volumes associated with different regions in a forest-based sensor field. Another example arises in a battlefield situation, where two remotely located battalions want to monitor enemy troop movements in different but partially overlapping battlefield sectors.

We consider the most general case, where multiple queries are injected asynchronously into the network at different root nodes. Since there is no global knowledge of the different queries, completely distributed solutions are required. We formulate, and compare, three algorithms: a naive algorithm (NMQ), an optimal algorithm (OMQ), and a heuristic algorithm (ZMQ). ZMQ is based on sharing partially aggregated results of pre-configured geographic regions (called zones [7]), and exploits the novel idea of applying a grouping technique for optimization of multi-root, multi-query processing, by using the location attribute of sensor nodes as the grouping criterion. This optimization aims to maximally share the reading and transmission of sensor node values belonging to multiple queries. Once sensors are deployed, the sensor field can be viewed as being divided into zones, and a logical data aggregation tree is established to hierarchically represent the zones. The idea of using such a recursive tree for query dissemination and data retrieval is not a new idea; authors in [7] use a similar so-called quad-tree structure for handling spatial aggregation queries in sensor networks. In [5], the authors use this kind of quad-tree structure for optimizing queries that have frequently changing aggregation groups. Since the goal of our heuristic approach is to share the sensor readings and data transmission among different queries if their query regions intersect, we group together sensor nodes in the same zone, so that sensor nodes in the intersection region of multiple queries only need to send their sensed value once, independent of the number of queries. In a distributed and asynchronous manner, a query taps into the data aggregation tree at the lowest possible tree node such that the zone represented by that node's sub-tree contains the geographic area of its query coverage. Our approach becomes more effective as the regions associated with multiple queries increasingly overlap.

We performed extensive simulations on the proposed OMQ, NMQ, and ZMQ algorithms. The NMQ algorithm treats queries independently and does not do any sharing of the aggregated data for sharing of message transmission. Our simulation studies indicate that the OMQ and ZMQ algorithms provide significant reduction in messages (and thus energy saving) under a wide range of network conditions and query region options.

2 Background

2.1 Assumptions and Challenges

We make the following assumptions regarding the sensor network system model and solution framework:

1. All queries of interest are querying the same type of sensor data, like the temperature of the environment.
2. Each node in the network knows its geographical position and the scale of the sensor field. Since the use of GPS in each sensor node incurs a high cost and high power consumption, it may not be practical to have GPS on all sensor nodes. However, GPS-free localization techniques [9,10] make our assumption still reasonable.
3. A query is characterized by a rectangular query region, like in the example in Section 1. An arbitrarily shaped query region can be split into rectangular regions to get approximate results since there is typically a high degree of redundancy in sensor networks.
4. Either all queries use the same sampling rate, or the effective sampling rate is set as the highest of the individual sampling rates. As noted in [3], this way

of handling different sampling rates is based on the observation that those sensor readings with higher sampling rate can give a better approximation of the environment.

5. Queries are first injected into the sensor network via root nodes, which are regular sensor nodes of the network. Users can inject different queries into the sensor network from different root nodes of their choice.

Our approach to multi-root, multi-query processing in sensor networks is motivated by the goal of sharing sensor readings and data transmission among different queries if the query regions of different queries intersect. In designing a distributed solution for optimization of multi-root, multi-query processing, we identify two challenges:

Challenge One (C1): How to determine the intersection regions of multiple queries, especially if those queries are injected at different sensor nodes.

Challenge Two (C2): How to make nodes in different query regions group together for aggregation efficiency.

To address challenge **C1** we use the notion of zones [7] to represent query regions. A zone is a subdivision of the geographical extent of a sensor field, and each sensor node can compute the zones according to the scale of the sensor field independently. We will give more detail about the definition of zones in Section 2.3. Since zones are predefined when the network is deployed, intersection zones are easy to decide even though the queries are input at different root nodes. To address challenge **C2**, we apply a grouping technique to group sensor nodes in the same zone to form an aggregation efficient tree topology for multiple queries. Grouping sensor nodes in the same zone together in a sub-tree not only increases the aggregation efficiency, but also makes possible the sharing of partially aggregated results of zones. The grouping technique is reviewed next.

2.2 A Grouping Technique for Query-Informed Routing

Query processing in sensor networks typically proceeds in three phases: (i) disseminating queries into the network, (ii) sensing data, and (iii) retrieving data from the network. For phases (i) and (iii), a tree topology is formed using some variants of the broadcast and convergecast techniques (e.g., [2] presents an optimized broadcast protocol using an adaptive-geometric approach). Here, each node performs two actions: 1) according to the messages it receives, each node decides its own level and selects a parent node with respect to the tree topology being created, and 2) the node broadcasts its own id and tree level. Once all nodes in the network have established their tree levels and parent nodes, the tree topology is defined.

In previous methods [8], sensor nodes select their parents using only tree levels. The grouping technique used in [13] is motivated by the fact that it is common for queries in a sensor network to be aggregation queries (such as COUNT, MAX, MIN, AVERAGE, etc.) using GROUP BY or WHERE. Such queries can form aggregation groups according to a specific attribute of the sensor nodes, and there are situations where such queries must remain active over long durations. The basic idea of the grouping technique is to try to force those sensor nodes with the same specific attribute, used in the GROUP BY or WHERE clause, to be logically close to each other when forming the tree topology.

Consider the query: SELECT SUM(value) FROM sensordb WHERE color =blue. Figure 1 illustrates the main idea. B represents blue node. The aggregation is completed at node S_3,3 if the tree topology, as shown in Fig 1a, is formed by using the grouping technique. However, if sensor nodes select parent nodes non-deterministically, we may end up with a tree topology as in Fig 1b.

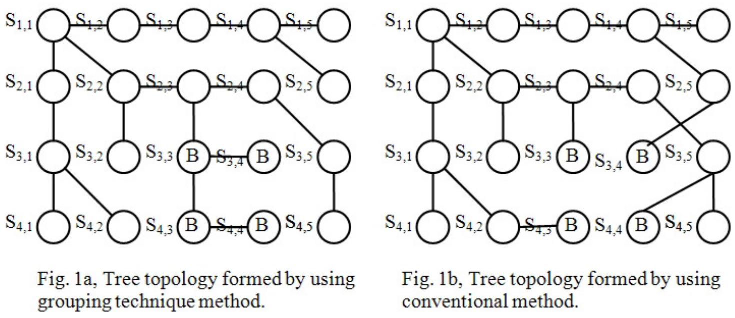
S1,1
S1,2
S1,3
S1,4
S1,5
S2,1
S2,2
S2,3
S2,4
S2,5
S3,1
S3,2
S3,3
S3,4
S3,5
S4,1
S4,2
S4,3
S4,4
S4,5
B
Fig. 1a, Tree topology formed by using grouping technique method.
Fig. 1b, Tree topology formed by using conventional method.

Fig. 1. An example for queries using WHERE clause

2.3 Sensor Field Division Using Zones

Zones for sensor networks have been used in the context of range queries [7]. Zones are a subdivision of the geographic extent of a sensor field. A zone is defined by the following constructive procedure. Consider a rectangle R on the x-y plane. Intuitively, R is the bounding rectangle that contains all sensors within the network. We call a sub-rectangle Z of R a zone, if Z is obtained by dividing R k times, $k \geq 0$, using a procedure that satisfies the following property: After the i-th division, $0 \leq i \leq k$, R is partitioned into $2i$ equal sized rectangles. If i is odd (even), the i-th division is along the values of the y-axis (x-axis). Thus, the bounding rectangle R is first sub-divided into two zones at level 1 by a vertical line that splits R into two equal pieces. Each of these sub-zones is split into two zones at level 2 by a horizontal line, and so on. The integer k is the level of zone Z, i.e., level(Z) = k.

A zone can be identified either by a zone code code(Z) or by an address addr(Z). The code code(Z) is a bit string of length level(Z), and is defined as follows. If Z lies in the left half of R, the first (from the left) bit of code(Z) is 0, else 1. If Z lies in the bottom half of R, the second bit of code(Z) is 0, else 1. The remaining bits of code(Z) are recursively defined on each of the four quadrants of R. This definition of the zone code matches the definition of zones given above, encoding divisions of the sensor field geography by bit strings.

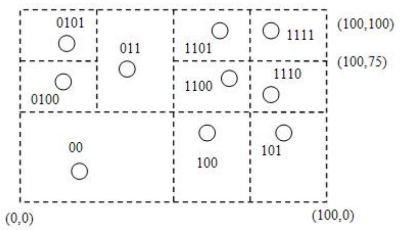

Fig. 2. Zone codes and boundaries

Figure 2 shows a deployed sensor network, and the zone code for each zone. The zone in the top right corner of R has a zone code of 1111, and its level is 4. The address of a zone Z, addr(Z), is defined to be the rectangle defined by Z. Each representation of a zone (its code and its address) can be computed from the other.

The zone with code 1111 represents the region [75, 100]×[75, 100] in sensor network space [0, 100]×[0, 100], where space [xmin, xmax]×[ymin, ymax] represents the rectangular region with the left bottom at (xmin, ymin) and right top at (xmax, ymax). Similarly, given a region [25, 75]×[50, 100], we can know that it contains zones 011 and 110. We use the same prefix of zones to represent a bigger zone that contains those zones. For example, zone with code 11 includes zones 1100, 1101, 1110, and 1111. Let *Prefix(codea, codeb)* be the longest common prefix of *codea* and *codeb*. For e.g., Prefix(1110, 11) equals 11.

As each node knows its position and the scale of the sensor field, these geographic zones are predefined once the network is deployed. Each sensor node knows its own zone code and the scale of the sensor field; hence it knows the geographical region that any zone code represents. Given a query represented by a query region, all sensor nodes identify the same set of zones that represent the region. This allows the distributed computing of intersection regions.

3 Multi-Query Modeling

3.1 The Naive Method for Multi-root, Multi-Query Processing (NMQ)

In Figure 3, Q1 and Q2 are injected from two different nodes R1 and R2. The different rectangles represent the different query regions. The NMQ algorithm sets up different tree structures for Q1 and Q2 separately. This naive algorithm does not share any sensor readings. The grey nodes, which are the nodes in the intersection region for query regions Q1 and Q2, need to send the same readings to different parent nodes twice, once for Q1 and once for Q2. Figure 3 shows the tree structures for Q1 and Q2 separately. A better algorithm would allow Q1 and Q2 to share the readings and messages of the grey nodes.

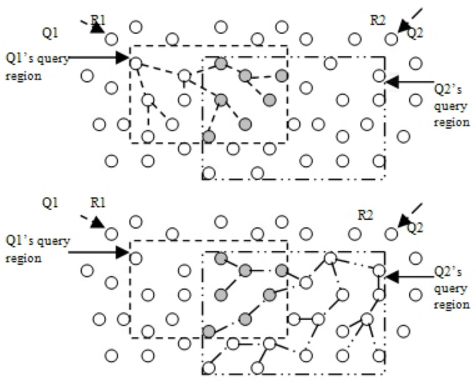

Fig. 3. NMQ for multi-root multiple query

3.2 Optimal Multi-root, Multi-Query Processing (OMQ)

To lower the cost of multi-query processing, we can share the readings of sensor nodes in the intersection region. Here, the two challenges **C1** and **C2** need to be solved.

To solve **C1**, reading-sharing methods need to know the identity of the nodes in the intersection regions. Consider Figure 3. Since Q1 is known only to R1 and Q2 is known only to R2, one option is to let the network first construct the tree structure for Q1 and then adjust the tree structure in the intersection region when Q2 is propagated in the network. This readjustment in the middle of processing Q1 would be problem-prone and energy consuming. The optimal algorithm (OMQ) pre-constructs the sub-trees for the intersection regions and the other regions of queries. Here, one and only one sub-tree is constructed for each region, and then the paths from R1 and R2 to those sub-trees are pre-setup, as in Figure 4. However, since the regions are known only after all queries are injected, this method is of limited use even for common-root multi-query processing. It is applicable to multi-root, multi-query processing only in the static case, where the regions of Q1 and Q2 are known in advance and do not change. Still, it can serve as a benchmark.

If we can know the regions, i.e., if **C1** is solved, we can use the grouping technique to solve **C2** -by grouping nodes in the same region into one sub-tree.

3.3 Zone-Based Multi-root, Multi-Query Processing (ZMQ)

To increase the sharing of data provided by sensor nodes in the presence of dynamically arriving queries, a practical way is to predefine a set of globally known regions in the network, and then represent query regions as pre-defined regions. This gives different nodes the same view of the field and lets them use the same known regions to represent the same query region.

Figure 5 shows an example of such pre-defined globally known regions, represented as a tree. Each node in the tree represents a globally known region. For

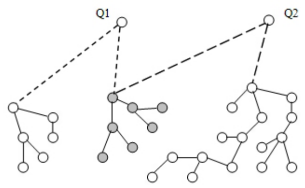

Fig. 4. OMQ method for multi-query processing

example, the network [0, X]×[0, Y] is one globally known region, as is the region represented by [0, X/2]×[0, Y/2]. In the tree structure, the globally known region represented by a parent node consists of the globally known regions represented by the children nodes. Since a node cannot know ahead of time what the intersection regions might be, it is not practical to pre-setup exactly one region for each intersection region. So, we use pre-defined globally known regions to represent all possible query regions, and an intersection region is represented by one or more such globally known regions. Each time a query is injected into the network, the root node computes the globally known regions that can be used to represent the query region, and sets up paths from itself to the root nodes of those globally known regions. Queries can share the partially aggregated results of globally known regions if they have globally known regions in common. Although the roots cannot know the intersection regions ahead of knowing the queries, all roots can use identical views of the globally known regions to represent query regions. The intersection regions for any set of queries would then be a set of globally known regions.

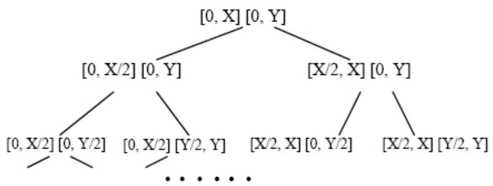

Fig. 5. Examples of predefined globally known regions

The framework of globally known regions solves challenge (**C1**). This framework is implemented using zones (Section 2.3). To solve challenge (**C2**), the ZMQ algorithm uses the grouping technique, reviewed in Section 2.3, to group nodes in each zone into one sub-tree. All those zone sub-trees form a globally pre-setup tree. Furthermore, the representing globally known regions are easy to compute given a query region. Given the size of the sensor field and a zone code, a node can easily compute the region represented by this zone. Therefore, intersection regions are very easy to determine even in a distributed environment.

4 Algorithm ZMQ (Zone-Based Multi-root, Multi-Query Processing)

4.1 Zone Setup

The system model assumes that each sensor node knows its location and the scale of the network region (see Section 2). Each node learns the locations of neighbors within radio range through direct-broadcast communication. Upon hearing any neighbor node, the node, say node A, calls an algorithm BUILD_ZONE to build its zone code and boundaries accordingly. The first split of the whole sensor field creates two sub-zones, 0 and 1.

On finding a new neighbor, a sensor node uses algorithm BUILD_ZONE to split its current zone either vertically or horizontally into two sub-zones, and then adjusts its zone code. Using the BUILD_ZONE algorithm, each node knows its own zone code. These zones form the zone-tree structure, See Figure 6. The parent-child relations in Figure 6 represent containment, where the parent zone is comprised of child zones. Each node in the zone tree is a zone; the path from the root node to the current node is the zone code of the zone represented by the current node.

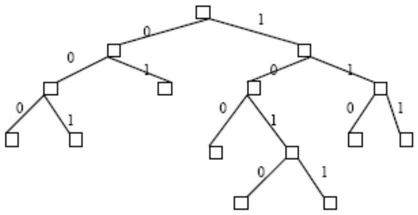

Fig. 6. The zone tree for all zones

The rectangular region represented by each zone is decided once the construction of zone codes is complete. For example, zone 1011 represents the region [3X/4, X]×[Y/4, Y/2]. After computing their zone codes, sensor nodes in the same zone automatically form groups based on the computed zone codes.

4.2 Sensor Node Grouping in Zones

In the algorithm GROUPING_ZONE, A is the sensor node executing the algorithm, Z_A is the zone represented by A, and code(Z_A) is the zone code of Z_A. B is any neighbor node of A. Zone code ES (empty string) represents the whole sensor field zone. Operator "$\gg n$" eliminates the last n characters of a string; e.g., $110011 \gg 1$ is 11001.

Each node executes this algorithm after it detects that all its neighbor nodes have decided their zones. The idea is that a node A first searches neighbor nodes in its immediate parent zone (the smallest zone contains Z_A), to find a node with minimum zone code. If such a node B exists, and its zone code is smaller than

```
GROUPING_ZONE
 1. $code = code\_old = \mathrm{code}(Z_A)$          // initialize zone code variables
 2. While ($code \neq ES$)                          // Iterate until code represents the whole network
 3.    $code = code\_old >> 1$  //code rep. the zone containing the zone rep. by code_old
 4.    $T = \varnothing$                             // initialize the temporary set T
 5.    For each $B$ where $B$ is neighbor of $A$
 6.       If ($Prefix(code, \mathrm{code}(Z_B)) = code$ and $Prefix(code\_old, code(Z_B)) \neq code\_old$)
 7.          $T = T \cup \{B\}$          // put any neighbor $B$ into set $T$, if $B$ is not inside
 8.                                     // zone of $code\_old$, but is in the zone of $code$
 9.    If ($T = \varnothing$)
10.       $code\_old = code$
11.       Continue                      //back to line 2 to try the upper level zone
12.    Let $C$ be the node in $T$ with smallest code
13.    If ($code(Z_C) < code(Z_A)$)
14.       Select $C$ as $A$'s grouping parent node
15.       Return                                     // parent node of $A$ is $C$
16.    $code\_old = code$
17. Return                                           // A is the root node
```

A's zone code, then A selects B as its grouping parent. Otherwise, A searches neighbor nodes in the parent zone of its immediate parent zone, and so on, until it finds a parent. Consider Figure 7. Node 1111 has three neighbors, 1100, 1101, and 1110. This node first searches the direct parent zone of zone 1111, which is zone 111, and finds node 1110, which has smaller zone code than itself. Therefore, node 1111 selects node 1110 as its grouping parent node. The root node of zone 111 is node 1110. Similarly, node 1110 also has three neighbors. This node first searches its direct parent zone 111, and finds node 1111, which has bigger zone code than itself. So node 1110 searches zone 111's parent zone which is zone 11, and it finds that node 1100's zone code is the smallest one. Node 1110 would select 1100 as its grouping parent node.

Definition 1. *Zone links for the grouping-tree network are the parent-child links formed using the GROUPING_ZONE algorithm. A node's zone link is the link from the node to its parent.*

All the links in Figure 7 are zone links. Zone links are pre-constructed once a network is deployed. Each node has a unique zone link. The zone-link tree formed by zone links has the property that nodes in the same zone are in one sub-tree. This achieves the method of grouping by zone. When queries are received, the routing network only needs to be adjusted to include paths from the root nodes of zones to the root nodes of queries. These paths are established by "forward links"; see Section 4.3.

Note that for unevenly distributed networks, or if nodes fail, the algorithm GROUPING_ZONE may form several tree structures in a network. Consider Figure 7. If node 100 is absent, then node 101 cannot communicate with node 00, and two tree structures get formed - one rooted at 00 and the other at 101.

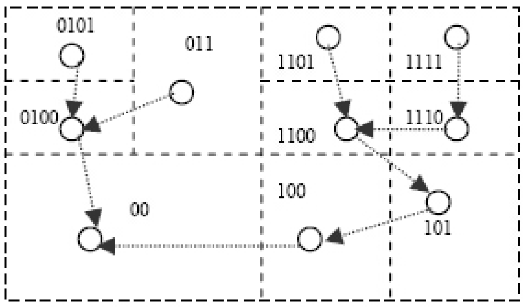

Fig. 7. A Grouping-tree example

However, the routing-tree construction algorithm of ZMQ would still form one routing tree for each query.

4.3 Query Handling

Definition 2. *Forward links are the parent-child links that connect root nodes of zones to root nodes of queries. While a node only has one zone link, it can have multiple forward links, one for each different query.*

Forward links are created in the grouping-tree in response to queries being injected to root nodes (see Section 4.3.2). These links transfer partially aggregated value of zones to the root nodes of queries during query processing. During query processing, forward links are all active, while zone links may be in an active or inactive state (i.e., some zone links may not be used to process some queries and thus they are not a part of the final routing topology).

Figure 8 shows an example of active zone links, inactive zone links and forward links. Two queries, Q1, with query region [0,75]×[50, 100], and Q2, with query region [25, 100]×[50, 100], are injected at different root nodes. After the queries have been propagated into the network, a routing topology is created. The dashed arcs illustrate the forward links, which form the paths from roots of zones to root nodes of queries. The dotted arcs illustrate inactive zone links, while the solid arcs illustrate active zone links, which form the sub-trees for queried zones.

4.3.1 Region Representation

Observation 1. *A region R (of a query Q) can be uniquely represented by a set of zones* $S = \{Z1, Z2, \ldots\}$*, where (i) zones in S do not overlap with each other (i.e., no node in* $Zi \in S$ *is in* $Zj \in S$*, for* $Zi \neq Zj$*), and (ii) no two zones in S can be siblings.*

Observation 1 is the basis for data sharing of intersection regions in our multi-root, multi-query processing. Once sensor nodes are deployed, they first use

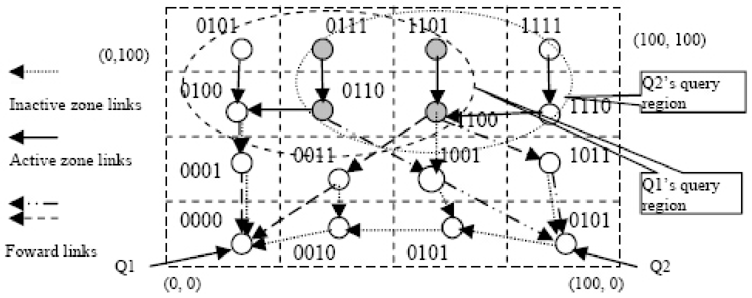

Fig. 8. Example routing structure for query processing

BUILD_ZONE to compute individual zone codes, and then use GROUPING_ZONE to group nodes in zones into sub-trees and thus form a complete zone-link tree. This was the pre-setup process. Now we address the process of handling queries, which requires an algorithm for setting up paths from root nodes of queries to root nodes of zones for queries, and algorithms for data retrieval using the paths.

4.3.2 Routing and Data Retrieval

In the tree built by the algorithm GROUPING_ZONE, each zone can compute its partial aggregation result in the zones root node in every sampling epoch. The processing for a query Q needs to set up paths from the root node of Q to each of the root nodes of the zones that belong to Q's zone representation. This is handled by an algorithm called BUILD_ROUTING_TOPOLOGY.

The algorithm BUILD_ROUTING_TOPOLOGY constructs routing tree topologies for data retrieval. The algorithm implements two features: 1) it uses a special forward-link notification message, FL_Notify, to build forward links from root nodes of zones to root node of queries; 2) it changes inactive zone links to active zone links based on whether a node is in a zone that is a representing zone of a query.

Each node in the network maintains a neighbor table recording status information of its neighbors, such as id, tree level, etc. Once a node receives a new query Q that has been injected into the network, this node sets its tree level (for Q) as 1 because it is the root. Then, this root node broadcasts a Query Broadcast (QB) message, containing its own id, tree level, the query information, and the zone representation of the query. The broadcast is across one hop, so only immediate neighbors of the sender receive the QB. On receiving such a QB message from some node Z, a node A updates its own neighbor table, specifically the data about neighbor Z (including Z's tree level for Q).

Each node A periodically executes the BUILD_ROUTING_TOPOLOGY algorithm. The algorithm is executed independently for each query Q. Consider any query Q for which some query broadcasts QB have been received. Node A first tests if it has already selected a routing-tree parent node for this query. If node A has not selected such a parent node, A checks its neighbor table to find a neighbor node M with minimum tree level for query Q. A then selects M as its routing-tree parent node for query Q, and sets its own tree level for Q as M's tree level for Q plus 1. A then broadcasts its id, tree level, and Q's query information in a QB message using a 1-hop broadcast. Then, if A is a root node of a representing zone of query Q, A sets M as its forward link parent for Q, and sends a FL_Notify message for Q to M. Else, if A is in a representing zone of the query but not a root node of that zone, A sets its own inactive zone link to be an active zone link.

Consider the case where A has previously selected a parent node M for query Q, and it has also received a FL_Notify message for Q from some child node. This can happen if one of A's child nodes is a root node of a zone representation for query Q, or one of A's child nodes received a FL_Notify message for Q from one of the childs children nodes. Node A constructs a forward link for Q by setting the routing-tree parent node as the forward-link parent node for Q and sending a FL_Notify message for Q to this forward link parent. Otherwise, node A has not received the FL_Notify message for Q from any child, implying that A might not be in a tree path from root nodes of zones to the root node of query Q and A might not need to be in the active state in the data retrieval phase. Hence, A exits the algorithm.

Since the zone link of each node is unique (pre-setup by the algorithm GROUPING_ZONE), if two queries have a common zone in their zone sets, they both actually change the status of the same zone links in the zone from inactive to active. Hence, that sub-tree of the routing tree for both queries is common. Only the root node of that common zone may construct different forward links for different queries - the partially aggregated value of the zone will be sent separately by the root node of the zone to the different root nodes of queries along different paths.

In the BUILD_ROUTING_TOPOLOGY algorithm, for any node A, its forward-link parent is initially null. Nodes having forward links or active zone links (as established by the above algorithm) would be in the active state, meaning that these nodes can transmit data messages during query processing. Other nodes may enter a sleep state to save energy. Nodes engaged in query processing do so by executing the algorithm DATA_RETRIEVAL. Using Figure 7 as an example, node 1111 would send the tuple (1111, value) to its parent node 1110, while node 1110 would create an aggregated value for zone 111 and then send the

```
DATA_RETRIEVAL(A)                          // A is the node executing the algorithm
1.  If (A is a leaf node)
2.          Send zone code and sensor value to parent node
3.          Return
4.  Aggregate data received from children based on zones          //merge smaller zones
5.  While (there is a forward link for a query)
6.          Send along the forward link the partially aggregated value for that query
7.  If (there is an active zone link)
8.          Send all partially aggregated values based on zones along the zone link
9.  Return
```

tuple (111, aggregated-value) to its parent node 1100. For node 101, it receives (11, value) from node 1100. It cannot aggregate its zone 101 with zone 11, so it would send a two-tuple message ((101, its own value), (11, value)) to its parent node 100; and so on.

5 Experimental Evaluation

We performed simulation experiments to compare and analyze the algorithms, NMQ, OMQ, and ZMQ.

– NMQ constructs a different routing tree for each query, thereby using a different tree for each query region.
– ZMQ is implemented based on the details in Section 4.
– Based on the position data for ZMQ, we can compute the intersection regions for OMQ. Then the grouping technique is used to group nodes in the same intersection regions together in the same sub-tree to compute the results for OMQ. As OMQ assumes that the multiple query regions are known before the query processing, it forms one sub-tree for each intersection region.

Deployment: We use a 256 × 256 cell matrix, where a sensor can be placed at the center of a cell. The length of each side of a cell is 1. Each node, except the nodes in the border cells, can communicate directly with its eight direct neighbors in the matrix. By default, each cell has a sensor placed in it. Input parameters:

N: The number of queries (Each query defines a query region).
D: The network density, defined as the number of sensors per cell of the sensor field matrix. To be consistent with the system model, the highest value of D is 1. This is also the default value.
QR: The query region representing a query.
OP: Overlap percentage, which we define as
$\left(\frac{\sum_I sizeof(I)*(numberof(I)-1)}{\sum_Q sizeof(Q)}\right)\left(\frac{N}{N-1}\right)$, where

– I is an intersection region, defined as the largest region in which all the nodes are queried by the same set of queries.
– sizeof(I) is the number of nodes in the intersection region I,
– numberof(I) is the number of queries that each node in I receives,
– N is the total number of query regions, and
– $\frac{N}{N-1}$ is a scale factor for normalization.

Metric: Average number of messages (ANM) per node per epoch, is defined as the total number of messages used in each retrieving epoch divided by the total number of nodes in the query regions. Nodes in intersection regions are counted separately for each query, i.e., nodes in intersection regions are counted multiple times. Formally,

$$ANM = \left(\frac{total\ number\ of\ messages}{\sum_Q sizeof(Q)}\right)$$

5.1 Impact of Overlap Percentage(OP)

In this experiment, we show the impact of the overlap percentage on the performance of the three algorithms. We hold constant the input parameters QR and N, but vary the position of different query regions to change the overlap percentage for all queries. As seen in Figure 9, we can observe that ZMQ outperforms NMQ as the percentage of overlapping increases, and ZMQ has nearly the same performance as OMQ. The reason that the number of messages used by ZMQ and OMQ decreases is because these algorithms share the readings of sensor nodes in the intersection regions. As the overlapping increases, more sharing is possible, and fewer messages are needed. Notice that ZMQ uses more messages than NMQ when there is no overlap between the two query regions. This is because ZMQ slightly decreases the aggregation extent and increases the number of messages needed if a query region contains more than one pre-setup zones. However, we can also see from Figure 9 that if there is an overlap between the query regions, the number of messages reduced by sharing readings and transmissions exceeds the number of messages increased by such a decrease in the aggregation extent.

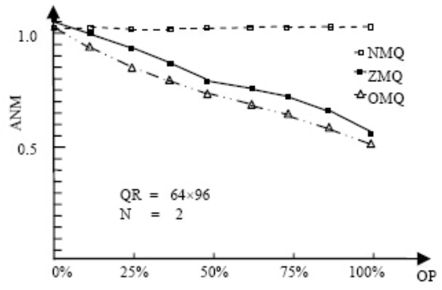

Fig. 9. Impact of OP on number of messages

The difference between ZMQ and OMQ is because each intersection region in ZMQ may consist of more than one zone, i.e., more than one sub-tree, while each intersection region in OMQ consists of just one sub-tree. The aggregation extent for intersection regions for ZMQ is slightly less than the aggregation extent for OMQ. This causes ZMQ to use more messages than OMQ.

5.2 Impact of Network Density (D)

In this experiment, we show the effect of the density of the sensor network on the performance of the three algorithms. We hold QR, N, and OP fixed, and vary the density of the sensor field. The average number of messages transmitted, as a function of density is shown in Figure 10. The density in Figure 10 is computed as the number of sensor nodes divided by the size of the sensor field. For example, for a 256 × 256 sensor field, the density 1/4 means that there are 128 × 128 sensor nodes evenly spread in the sensor field.

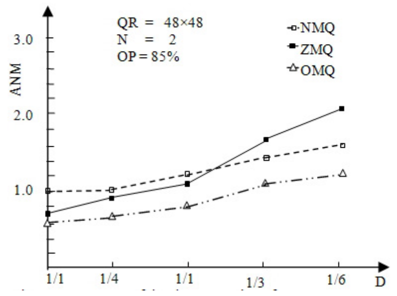

Fig. 10. Impact of density on number of messages

Observe that for all algorithms, as density decreases, the ANM increases. This is because when density decreases, the nodes become further apart, and more messages are needed to collect the readings. This can also be seen from the definition of ANM, viz., $\frac{total\ number\ of\ messages}{\sum_Q sizeof(Q)}$. As D decreases, the denominator decreases proportionately to D but the numerator does not change as rapidly. Observe that even if the query regions have significant overlap, ZMQ may perform worse than NMQ when the density is low enough. The reason is that as the density becomes lower, the number of nodes in the intersection area decreases, and the data and transmission sharing also decreases. At some threshold, the number of messages added because of the decrease in aggregation extent may exceed the number of messages decreased by data sharing.

5.3 Impact of Query Region Size (QR)

Figure 11 shows the relationship between size of query regions and the average number of messages transmitted in each epoch for the three algorithms. Observe that as QR increases, the ANM $(=\frac{total\ number\ of\ messages}{\sum_Q sizeof(Q)})$ for NMQ approaches. In addition, as QR increases, ANM of ZMQ approaches the ANM of OMQ. This implies that ZMQ performs better when the sizes of query regions are large.

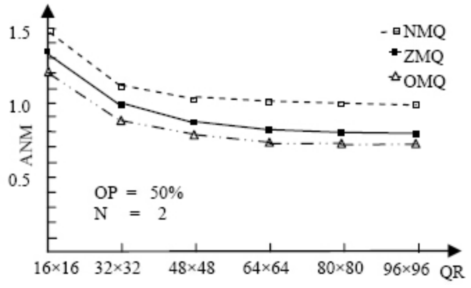

Fig. 11. Effect of QR on the algorithms

Figure 11 also shows that for all the algorithms, as the size of the query region increases, the average number of messages for each node first decreases and then becomes stable. This is due to two factors.

- Consider the ratio X:Y, where X is the number of messages used from the root nodes of each region to the root nodes of queries, and Y is the total number of sensor nodes in the query regions. This ratio represents an amortized overhead to reach the query root from the region roots. As QR increases, the numerator tends to decrease somewhat, and the denominator increases; thereby decreasing this overhead. As QR continues to increase, the value of this overhead becomes relatively small.
- With increasing QR, the predominant factor becomes the degree of node overlap and the sharing of sensor readings among regions. As QR increases, this also tends to have a saturation effect. In our example, this sets in around regions of size 48×48.

ZMQ is better than NMQ, irrespective of the query region size when the overlap percentage is not very low.

5.4 Impact of Number of Queries (N) Using Controlled Overlap Percentage

This experiment studies the effect of the number of queries on the performance of the three algorithms. We set OP and QR, and change the number of queries injected into the sensor field. Observe from Figure 12 that as the number of queries increases, the ANM for OMQ and ZMQ decreases, while it remains almost stable for NMQ. We also can find the answer from the definition of ANM as follows. Let:

- S be the number of nodes that can share reading and transmission,
- SN be the total number of sensor nodes in query regions (i.e., $\sum_Q sizeof(Q)$),
- E be the extra messages needed to send data from root nodes of sub-trees to root nodes of queries.

ANM ($=\frac{total\ number\ of\ messages}{\sum_Q sizeof(Q)}$) = (SN-S+E)/SN = 1-S/SN + E/SN. For NMQ, S is always 0, so ANM of NMQ is always more than 1 and decreases as SN increases.

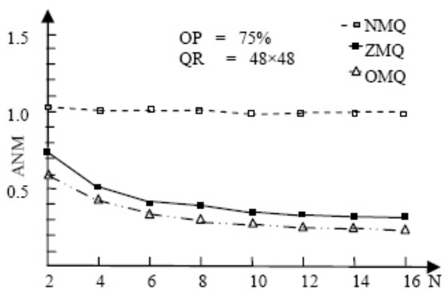

Fig. 12. Effect of N on algorithms, with fixed OP

As the number of queries increases, SN and S both increase, while E stays almost stable. Therefore, the overhead for OMQ and ZMQ approaches the value 1-S/SN as the number of queries increases.

6 Conclusion

This paper identified for the first time, multi-root, multi-query optimization for long duration aggregation queries. The paper then formulated two algorithms - an optimal algorithm (OMQ) and a heuristic algorithm (ZMQ) based on sharing the partially aggregated results of pre configured geographic regions. Simulations on OMQ and ZMQ, as well as the naive algorithm (NMQ) that does not do any sharing, indicate that the proposed algorithms provide significant energy savings under a wide range of network conditions and query region options. We found that OMQ always performs best, as expected. Furthermore, ZMQ performs generally better than NMQ when the sizes of the query regions are big, the density of the sensor field is high, and there are large overlaps among queries. ZMQ performs increasingly better than NMQ as the sizes of query regions become larger, the sensor field density increases, and overlap among query regions increases.

	Dynamic queries	Aggregation extent	Energy efficient	Time complexity	Space complexity	initialization	Latency
NMQ	Yes	Very good	Good	O(1)	O(1)	No	Small
ZMQ	Yes	Good	Very good	O(1)	O(1)	Yes, O(n)	Moderate
OMQ	No	Very good	Best	O(1)	O(1)	No	Small

Recommendation: The most applicable situation for ZMQ is when there are big query regions, big overlap among queries, and high network density. Otherwise, if the application requires dynamic query processing, NMQ is a good algorithm. If the queries are known a priori (before any processing), and no new queries come in during processing, OMQ is the best algorithm. ZMQ is a practical and energy efficient algorithm for multi-root, multi-query processing.

References

1. Dasgupta, K., Kalpakis, K., Namjoshi, P.: Improving the Lifetime of Sensor Networks via Intelligent Selection of Data Aggregation Trees. In: Proceedings of the Communication Networks and Distributed Systems Modeling and Simulation Conference (2003)
2. Durresi, A., Paruchuri, V., Iyengar, S.S., Kannan, R.: Optimized Broadcast Protocol for Sensor Networks. IEEE Transactions on Computers 54(8), 1013–1024 (2005)
3. Emekci, F., Yu, H., Agrawal, D., Abbadi, A.E.: Energy-Conscious Data Aggregation Over Large-Scale Sensor Networks, UCSB Technical report (2003)
4. Estrin, D., Srivastava, M.B., Sayeed, A.: Tutorial on Wireless Sensor Networks. In: ACM International Conference on Mobile Computing and Networking (MOBICOM) (2002)

5. Jia, L., Noubir, G., et al.: GIST: Group-Independent Spanning Tree for Data Aggregation in Dense Sensor Networks. In: International Conference on Distributed Computing on Sensor Systems (DCOSS) (2006)
6. Kannan, R., Sarangi, S., Iyengar, S.S.: Sensor-Centric Energy-Constrained Reliable Query Routing for Wireless Sensor Networks. Journal of Parallel and Distributed Computing 64(7), 839–852 (2004)
7. Li, X., Kim, Y.J., Govindan, R., Hong, W.: Multi-dimensional Range Queries in Sensor Networks. In: ACM Conference on Embedded Networked Sensor Systems (Sensys 2003), pp. 63–75 (2003)
8. Madden, S., Franklin, M.J., Hellerstein, J.M., Hong, W.: Tag: A Tiny Aggregation Service for Ad-hoc Sensor Networks. In: 5th Symposium on Operating Systems Design and Implementation, pp. 131–146 (2002)
9. Roumeliotis, S.I., Berkey, G.A.: Collective Localization: a Distributed Kalman Filter Approach to Localization of Groups of Mobile Robots. In: Proc. IEEE Intl Conf. on Robotics and Automation (ICRA), (2000)
10. Savvides, A., Han, C.-C., Srivastava, M.B.: Dynamic Fine-Grained Localization in Ad-Hoc Networks of Sensors. In: ACM SIGMOBILE (2001)
11. Trigoni, N., Yao, Y., Demers, A., Gehrke, J., Rajaraman, R.: Multi-Query Optimization for Sensor Networks. In: International Conference on Distributed Computing on Sensor Systems (DCOSS), pp. 307–321 (2005)
12. Wu, Q., Rao, N.S.V., Barhen, J., Iyengar, S.S., Vaishnavi, V.K., Qi, H., Chakrabarty, K.: On Computing Mobile Agent Routes for Data Fusion in Distributed Sensor Networks. IEEE Transactions on Knowledge and Data Engineering 16(6), 740–753 (2004)
13. Zhang, Z., Shatz, S.M.: A Technique for Power-Aware Query-Informed Routing in Support of Long-Duration Queries for Sensor Networks. In: International Conference on Sensing, Networking and Control (ICNSC 2006) (2006)