

PAPER

Progressive Processing of Continuous Range Queries in Hierarchical Wireless Sensor Networks

Jeong-Hoon LEE[†], *Student Member*, Kyu-Young WHANG^{†a)}, Hyo-Sang LIM[†], Byung SUK LEE^{††},
and Jun-Seok HEO[†], *Nonmembers*

SUMMARY In this paper, we study the problem of processing continuous range queries in a hierarchical wireless sensor network. Recently, as the size of sensor networks increases due to the growth of ubiquitous computing environments and wireless networks, building wireless sensor networks in a *hierarchical* configuration is put forth as a practical approach. Contrasted with the traditional approach of building networks in a “flat” structure using sensor devices of the same capability, the hierarchical approach deploys devices of higher-capability in a higher tier, i.e., a tier closer to the server. While query processing in flat sensor networks has been widely studied, the study on query processing in hierarchical sensor networks has been inadequate. In wireless sensor networks, the main costs that should be considered are the energy for sending data and the storage for storing queries. There is a trade-off between these two costs. Based on this, we first propose a *progressive processing* method that effectively processes a large number of continuous range queries in hierarchical sensor networks. The proposed method uses the query merging technique proposed by Xiang et al. as the basis. In addition, the method considers the trade-off between the two costs. More specifically, it works toward reducing the storage cost at lower-tier nodes by merging more queries and toward reducing the energy cost at higher-tier nodes by merging fewer queries (thereby reducing “false alarms”). We then present how to build a hierarchical sensor network that is *optimal* with respect to the weighted sum of the two costs. This allows for a cost-based systematic control of the trade-off based on the relative importance between the storage and energy in a given network environment and application. Experimental results show that the proposed method achieves a near-optimal control between the storage and energy and reduces the cost by 1.002 – 3.210 times compared with the cost achieved using the flat (i.e., non-hierarchical) setup as in the work by Xiang et al.

key words: *hierarchical sensor network, progressive processing, continuous range queries*

1. Introduction

As the computing environment evolves toward ubiquitous computing, there has been increasing attention and research on sensor networks. In the sensor networks environment, sensor nodes are connected through the network to the server (or base station) which collects data sensed at the nodes [1]. Example applications in such an environment include environment monitoring (e.g., temperature, humidity), manufacturing process tracking, traffic monitoring, and intrusion detection in a surveillance system.

In particular, as *wireless* networks become more common, there has been a lot of research on wireless sensor net-

works in which sensor nodes are connected in an ad-hoc network configuration in order to reduce the cost of deployment. In general, the objective in a wireless sensor network is to deploy cheap sensor nodes with limited resources (e.g., battery power, storage space) effectively and to collect data from those sensor nodes by using their limited resources efficiently [10].

There is an increasing trend lately toward *large-scale* wireless sensor networks [14], [15], as the scope of applications extends to municipality management, global environmental monitoring, etc. These networks typically aim at supporting a large number of sensor nodes deployed in a large area for use by a large number of users. For example, in the Network for Observation of Volcanic and Atmospheric Change (NOVAC) project [13], wireless sensor networks deployed in 15 volcanoes spread across five continents are connected in a multi-tier configuration to support a global volcano monitoring project. As another example, the EarthNet Online [5] collects earth observation information such as the worldwide weather and bird migrations through wireless sensor networks and makes the information available for thousands of individuals or organizations. This kind of scale upgrade will bring about a proportionate increase of the number of concurrent queries and the amount of sensor data. Thus, we expect an increasing importance of processing a large number of queries and a high volume data *effectively* in wireless sensor networks. In addition, we expect that building such large scale wireless sensor networks *economically* is important as well.

With this background, in this paper, we consider *storage* requirement needed to store queries in sensor nodes and *energy* consumption (i.e., battery capacity) needed to send the collected data from those nodes to the server. There exists a trade-off between these two cost factors. Let us explain this trade-off with the centralized approach and the distributed approach [17], which are the two naive approaches to building wireless sensor networks. In the centralized approach, the sensor nodes do not store any query and simply send all data to the server, which then processes all the queries on the data received. In this case, there is no storage cost to store queries in individual sensor nodes but the energy cost is very high. In the distributed approach, on the other hand, individual sensor nodes store all the queries and send only the results of processing the queries to the server, which then simply collects the received query results (This scheme is known as *in-network query processing* [24]). In

Manuscript received March 2, 2009.

Manuscript revised January 9, 2010.

[†]The authors are with the Department of Computer Science, KAIST, Yuseong-Gu, Daejeon, Korea.

^{††}The author is with the Department of Computer Science, University of Vermont, Burlington, VT 05405, U.S.A.

a) E-mail: kywhang@cs.kaist.ac.kr

DOI: 10.1587/transinf.E93.D.1832

this case, the energy cost can be reduced but the storage cost is high.

Neither of these two approaches is suitable for building *large scale* sensor networks. In the centralized approach, since data are accumulated over the course of being relayed toward the server, sensor nodes near the server should send more data than the nodes farther from the server. Thus, the nodes near the server consume more energy than the ones farther from the server; as a result, they will be burnt out in a shorter time. On the other hand, the distributed approach becomes infeasible as the number of queries increases. A sensor node is not able to store and process a large number of queries (e.g., several thousands of queries) due to the limitation in its memory and computing power.

Recently, in order to overcome these large scale problems, building wireless sensor networks in a *hierarchical* configuration is considered a practical alternative. A hierarchical wireless sensor network is organized in a *multi-tier architecture* [2] configured with sensor nodes having different amounts of resources and computation power. Nodes closer to the server have more resources and computation power than those farther from the server, and this makes it possible to carry out the processing that cannot be done with low-capacity nodes only. In hierarchical wireless sensor networks, nodes with smaller resources and computing power are recursively connected to nodes with more resources and computing power [2], [16], [18]; thus, nodes near the server are capable of handling the larger amount of data accumulated from lower tiers.

This paper proposes a method for building large scale hierarchical sensor networks to process queries effectively with respect to the trade-off between the energy cost and the storage cost. The queries considered in this paper are *continuous range* queries. Here, the continuous query represents the query that is executed continuously and periodically (unlike the one-time query that is executed once) [3], [4], [10]. Range queries are an important query type in many sensor network applications, particularly in monitoring applications [10], and there has been active research done to improve range query processing performance [8]. The method proposed in this paper systematically controls the trade-off between the energy cost and the storage cost through controlled merging of queries with similar ranges. Some current methods can reduce the energy cost by merging queries to avoid duplicate transmission of query results [12], [21]–[23]. They, however, all focus on *flat* sensor networks and, therefore, cannot utilize the characteristics of the hierarchical sensor network in which nodes at different tiers have different capabilities. Besides, their work does not reflect anything about the trade-off because they do not consider the storage cost at all. In contrast, in this paper, we fully utilize the characteristics of the hierarchical sensor network by employing a *progressive* approach, which merges increasingly more queries as the tier goes from the server toward the lowest tier and, in this way, finds the optimal merging at each tier.

In this paper, we first propose the model and algorithms

of the progressive query processing method. This method has two phases: *query merging* and *query processing*. In the query merging phase, we merge queries progressively as the tier goes from the highest (i.e., the server) to the lowest. That is, we first merge the input queries to generate the queries for the second-tier nodes and then merge them to generate the ones for the third-tier nodes, and so on. We say that the queries thus stored at multiple tiers form the *inverted hierarchical query structure*[†] as a whole. The *inverted hierarchical query structure* is a new structure proposed in this paper. In the query processing phase, the queries are processed progressively, that is, by refining the query result to be more accurate as data are sent from a lower tier to a higher tier. For this, the inverted hierarchical query structure is used to retrieve the query result at each tier.

Next, we propose a method that builds an optimal hierarchical sensor network by systematically controlling the trade-off between the storage cost and the energy cost according to their weights. Since the relative importance between the two costs may vary depending on the application and environment, we formulate the cost of building the network as a weighted sum of the two costs and minimize the total cost. As the optimization target parameter, we use the *optimal merge rate* – the average rate of merging queries at each tier.

Finally, we show through experiments that the proposed method is useful for building a hierarchical sensor network in a cost-effective manner. Specifically, first we show that there is little difference between the optimal merge rate obtained from an analytic model and the rate obtained from experiments; second, we show the superiority of the proposed method over the existing query processing method for flat sensor networks in terms of the total cost.

The rest of this paper is organized as follows. Section 2 discusses related work. Section 3 describes the model and the algorithms of the proposed progressive processing method for hierarchical sensor networks. Section 4 proposes an analytical method for effectively building a hierarchical sensor network. Section 5 shows the superiority of the proposed method over the existing method through experiments. Section 6 concludes the paper.

2. Related Work

In this section, we review the existing research on the range query processing in sensor networks and the state of the art in the *hierarchical* wireless sensor networks.

2.1 Range Query Processing in Sensor Networks

Recently, there has been research to complement the centralized approach and the distributed approach. Specifically, the proposed methods are to share query processing in an overlapping region in case there are overlapping query conditions. By identifying the overlapping regions among the

[†]It is a *forest* structure to be more precise (see Fig. 2).

user queries and rewriting the queries accordingly, the proposed methods eliminate duplicate processing and duplicate data transmission. These methods can be classified into the *partitioning* method and the *merging* method.

Partitioning method

The server partitions the individual query regions into overlapping regions and non-overlapping regions. Then, it sends the partitioned regions and the original queries to sensor nodes, which store them. Query processing is done for each partitioned region, and the query results are merged in the server or sensor nodes. Trigoni et al. [20] and Yu et al. [25] use this method to process range queries on the *location* information of sensor nodes. This method has the advantage that the result of merging the results of processing each partition is the same as the result of processing the original queries and, therefore, no “false alarm” will happen. It, however, has the disadvantage that, if there are a large number of overlapping query conditions, then the number of partitions to be stored in certain sensor nodes increases and, thus, the necessary storage increases as well.

Merging method

The server merges the regions of overlapping queries into one merged query region. The server then sends the merged queries to the sensor nodes that store them. Query processing results are then “reorganized” into those of the original queries in the server or sensor node. This method has the advantage that it can process a large number of queries at the same time by reducing the number of queries stored in a sensor node. It, however, has the disadvantage that a “false alarm” may happen as a result of merging queries. Muller and Alonso [12] propose a method that compares the predicates of the range queries to extract those common to all queries and generates one query that has only the common predicates as the query condition. In this method, if there is no predicate common to all queries, then one query with no query condition is generated and, thus, has the problem of incurring a lot of false alarms in that case. In the literature [21], [22], Xiang et al. have proposed a method that *incrementally* merges overlapping query regions and processes the resulting merged queries instead of the original queries. Here, the incremental merging is done until the cost of sending the false alarms occurring when queries are merged is no larger than the cost of sending duplicate results of overlapping query regions when queries are not merged. Xiang et al.’s query processing method has the meaning of a hybrid approach (i.e., reducing the needed memory amount and the data transmission amount) taking advantage of both the centralized approach and the distributed approach. Then, in the literature [23], Xiang et al. have proposed methods that reduce energy consumption of each sensor node in the wireless network by eliminating duplicate data transmission and avoiding data retransmission caused by the link failures. However, all the work targets “flat” sensor networks in which all sensor nodes in the network have the same capability and store the same set of merged queries. Thus, these methods have the problem that they cannot utilize the

characteristics of *hierarchical* sensor networks. Our method in this paper basically uses the same query merging method as Xiang et al.’s, but enhances it to control the rate of merging queries depending on the capabilities of individual nodes and to build a hierarchical sensor network. Our method has the advantage that it allows for a systematic control of the trade-off between the memory amount needed and the amount of data sent.

2.2 Hierarchical Wireless Sensor Networks

As the scale of sensor networks increases, the hierarchical structure is used more in real applications than the flat structure in which all sensor nodes have the same capability [2].

Representative examples of such hierarchical wireless sensor networks are PASTA (Power Aware Sensing, Tracking and Analysis) [18] mentioned in COSMOS [18] and SOHAN [6]. PASTA is used in military applications for enemy movement surveillance and is configured with the server and about 400 intermediate tier nodes each clustering about 20 sensor nodes. SOHAN is used in traffic congestion monitoring applications to measure the traffic volume using roadside sensor nodes and is configured with the server and about 50 intermediate tier nodes each clustering about 200 sensor nodes.

We expect that hierarchical sensor networks will be increasingly more utilized in the future as the scale and the requirement of applications increase. However, there has not been any research done on processing multiple queries taking advantage of the characteristics that sensor nodes at different tiers have different capabilities. Srivastava et al. [19] investigated how and on which node to process each operation during query processing in a hierarchical sensor network. This research, however, mainly deals with single query processing and, thus, is difficult to apply to multiple query processing.

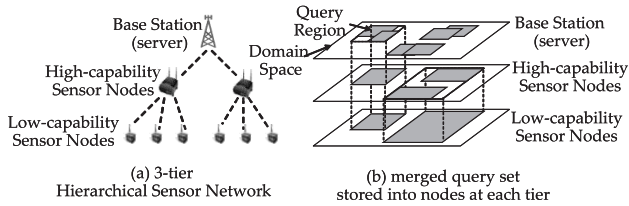
3. Progressive Processing in Hierarchical Sensor Networks

In this section, we present the progressive processing model and algorithms in hierarchical (i.e., multi-tier) wireless sensor networks.

3.1 Overview

In progressive processing, we systematically control the total processing cost by having the larger number of lower-capacity nodes (at lower tiers) partially process queries and the smaller number of higher-capacity nodes (at higher tiers) process the remainder.

Example 1 (Progressive processing in hierarchical wireless sensor networks): Figure 1 (a) shows an example of a hierarchical sensor network organized in three tiers. The nodes at the third (i.e., lowest) tier are the largest in number but the smallest in capability and are connected to the more capable



This figure shows a three-tier network as an example.

Fig. 1 Inverted hierarchical query structure in a hierarchical wireless sensor network.

nodes at the second tier. All nodes except the server generate data (i.e., partial query results) periodically and send them to the server relayed via the nodes at higher tiers. The server then provides the final query result to the user. Figure 1 (b) shows the set of queries stored in the nodes at each tier at the end of the query merging phase. In this figure, the rectangular regions represent range queries, and the boundary rectangle represents the domain space defined by the attributes specified in the queries. The server stores six original queries, the second-tier nodes store three queries resulting from the merge of the six original queries, and the third tier stores two queries resulting from further merging them. In the query processing phase, sensor nodes at the lowest tier process the two queries on the sensed data and send to the second tier only the data satisfying the conditions (i.e., ranges) of the two queries. Then, the sensor nodes at the second tier process the three queries on the data sent from nodes at the lower tier and the data they generate on their own, and send to the server only the data satisfying the query conditions. Since nodes at a higher tier have queries of finer granularity, they can reduce “false alarms” and thereby reduce energy consumption. The server processes the original queries on the data sent from all nodes at lower tiers and provides the final result to the user. □

From Fig. 1 (b), we can see that the stored queries altogether form an *inverted* structure of a multi-dimensional index tree. It is built from a multi-dimensional index storing the query ranges, by partitioning the index into multiple levels and then storing the root level of the index at the lowest-tier sensor nodes and the leaf level of the index in the server. In contrast to a multi-dimensional index tree structure in which all objects are stored in the leaf nodes and are merged to become more abstract at a higher level, in the proposed structure, the root (i.e., server) stores all objects (i.e., queries) and they are merged to become more abstract at a lower level.

The progressive processing has the *query merging* phase which generates queries to be stored at each tier of the hierarchical sensor network to form an inverted hierarchical query structure and the *query processing* phase which processes sensed data and sends the result to the server using the inverted hierarchical query structure. Query merging is performed off-line in batch processing, and query processing is performed on-line every time data are generated. In query merging, queries are sent toward the lowest tier while

merged “progressively”, and, in query processing, the sensor data are sent toward the server while being filtered “progressively”.

In the query merging phase, minimum bounding rectangles (MBRs) are obtained from the queries and expressed as merged queries. In this case, it is important to decide how many MBRs the queries should be merged into because the number of MBRs affects the trade-off between the energy consumption and the storage usage. That is, if more queries are merged, then the storage space used by the sensor nodes to store queries is reduced, but the energy consumption is increased due to more frequent false alarms. In this section, we present the model and algorithms under the assumption that the number of merged queries is known as each tier. Then, in Sect. 4, we present a method for determining the optimal number of merged queries analytically using a cost model.

In the query processing phase, all sensor nodes except the server process their own sensed data and the data received from the nodes at lower tiers, and send the results to the nodes at the next higher tier. Since more queries (of finer granularity) are stored at the higher-tier nodes, the accuracy of query result is higher in them, thus generating the query result progressively.

3.2 Network and Data Models

In this section, we first define the hierarchical sensor network. Then, we explain data and queries used in this paper.

The hierarchical sensor network

We make the following assumption about the configuration of a hierarchical sensor network. All sensor nodes are connected to form a tree rooted at the server, and the nodes at the same depth make one tier. Data are generated by not only the nodes at the lowest tier but also those at intermediate tiers, and the sensed data are sent to the server through the nodes at higher tiers. All sensor nodes at the same tier have the same capability, that is, the same amount of memory and battery power. Nodes closer to the server have higher capability, that is, a larger amount of memory and battery power. In addition, all nodes at the same tier store the same set of queries.

There have been various research on the hierarchical sensor network in the literature. However, the definitions of the hierarchical sensor network vary depending on specific environments. Nevertheless, it is a common understanding that a hierarchical sensor network consists of multiple tiers and deploys devices of different capabilities at different tiers [2], [6], [19]. We define the hierarchical sensor network as in Definition 1.

Definition 1 (The hierarchical sensor network): The hierarchical sensor network is defined as a tree $T = (V, E)$ of height h , where V is a set of vertices representing the sensor nodes and the server in the network (the root represents the server), and E is a set of edges representing the direct connection between a sensor node and its parent node. Let

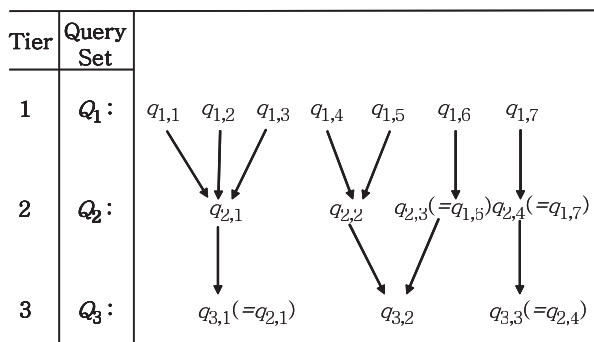


Fig. 2 An example of progressive query merging.

$node_i$ denote the node at i^{th} tier ($1 \leq i \leq h$). Let s_i and e_i denote the amount of storage and the amount of energy of $node_i$, respectively. Then, a hierarchical sensor network satisfies relationship: $s_j > s_k$ and $e_j > e_k$ ($1 \leq j < k \leq h$). \square

Query and data

In this paper, we focus on the range query as the query type in the hierarchical sensor network since it is an important query type in sensor networks applications [8], [10], [12], [21]. Consider a multi-dimensional domain space defined by the query attributes. Then, in the domain space, a query and a data element are represented as a hyper_rectangular region and a point, respectively [9].

3.3 Progressive Query Merging

3.3.1 The Model

Query merging in the first phase of progressive processing is done by finding the MBR enclosing the queries to be merged. *Progressive* query merging means that more queries are merged as the merging progresses to lower tiers. Thus, the size of a query region is larger at a lower tier while the number of queries is smaller. Let us refer to a query represented by an MBR that encloses certain queries at a higher tier node as a *merged query*, and denote the set of queries (or the *query set*) stored at the i^{th} tier node as Q_i . Then, we can represent the set of merged queries at each tier as one level in the inverted hierarchical query structure, as shown in Fig. 2. In this figure, an arrow represents the direction of query merging; queries at the tail of an arrow are merged to the query at the head of the arrow. For instance, the queries $q_{1,1}, q_{1,2}$ and $q_{1,3}$ at the 1^{st} tier are merged to the query $q_{2,1}$ at the 2^{nd} tier.

3.3.2 The Algorithm

For each i^{th} tier, the progressive query merging algorithm generates a merged query set Q_i . The objective of the algorithm is to minimize the query processing cost in consideration for the limited memory of sensor nodes. It is difficult to predict the cost of query processing for a given set of merged queries. The reason for this is that the cost depends not only

on the network-specific factors like routing but also on unknown factors such as the query and data distributions. In this paper, we use the simplified model proposed by Xiang et al. [21], in which the cost metric is the amount of data sent during the query processing, as the basis and extend it to fit into the hierarchical sensor network and take the memory usage into consideration. In Xiang et al.'s model, the size O of the overlapping region among queries and the size D of the dead region (i.e., the extra region added to make the MBR enclose the overlapping queries; it causes the false alarms) are calculated for each pair of two queries that are candidates to be merged, and the pair that maximizes the difference between the sizes of the two regions, $O - D$, are merged. The effect of this is to merge queries with large overlapped regions, which is a reasonable strategy for reducing the data transmission cost.

The proposed algorithm performs the query merging using a greedy approach based on the same strategy. Let $O(q_i, q_j)$ be the size of the overlapping region between two queries q_i and q_j , and $D(q_i, q_j)$ be the size of dead region between them. The algorithm chooses two queries q_i and q_j with the largest $O(q_i, q_j) - D(q_i, q_j)$ from the set of queries that are either merged queries or the original queries and merge them first. This strategy is the same as the strategy used by Xiang et al. [21] except that they consider only the pairs that satisfy $O(q_i, q_j) - D(q_i, q_j) \geq 0$. Specifically, in consideration of the storage cost for storing queries and the energy cost for sending query results, our approach determines the fixed number of queries that are to be stored into a sensor node at each tier. Then, we merge queries using a greedy method until we reach the number while Xiang et al.' approach determines the number of queries to be stored so as to only minimize the amount of data sent.

Figure 3 shows the progressive query merging algorithm. Inputs to this algorithm are the set of the original queries Q , the height h of the hierarchical sensor network to be built, and the set of the numbers of merged queries K to be stored in every node at each tier. The output is the sets of merged queries that are stored in every node at each tier. At each tier t , the algorithm repeats merging two queries at a time until the number of merged queries falls lower than k_t (lines 3-6). In order to find the pair of queries to be merged, it calculates the difference between the overlapping region and the dead region over every pair of queries and merges the pair with the maximum difference (lines 4-5).

3.4 Progressive Query Processing

3.4.1 The Model

In the query processing phase, for a given query, it is decided whether a data element falls inside the query region, that is, whether the attribute values representing the data element satisfy the range predicates representing the region. *Progressive* query processing is the process of propagating data elements bottom up in the inverted hierarchical query structure from the lowest-tier nodes to the highest tier node

Algorithm Progressive Query Merging

Input: (1) $Q = \{q_1, q_2, \dots, q_n\}$: the set of queries to be merged.
 (2) h : the height of a sensor network.
 (3) $K = \{k_1, k_2, \dots, k_h\}$: the set of numbers of merged queries stored into a sensor node in each of the 1st and the h^{th} tier.
 /* $k_1 \geq k_2 \geq \dots \geq k_h$, and $k_1 = n$, where n is the number of original queries. */
Output: $\{Q'_2, \dots, Q'_h\}$: The sets of merged queries stored into a sensor node at each of the 2nd to the h^{th} tier.

Algorithm:

```

begin
1. for tier  $t = 2$  to  $h$  begin
2.  $Q'_t = Q_{t-1}$  /*  $Q'_1 = Q$  */
3. repeat
4. find a pair of queries ( $q_i$  and  $q_j$ ) with the highest value of  $O(q_i, q_j) - D(q_i, q_j)$  for  $i=0, 1, \dots, |Q'_t|$  and  $j=0, 1, \dots, |Q'_t|$  ( $j < i$ ); remove them from  $Q'_t$ ; merge  $q_i$  with  $q_j$ .
5. insert the merged query (that is, the result of merging  $q_i$  with  $q_j$ ) into  $Q'_t$ .
6. until  $(|Q'_t| \leq k_t)$ 
7. end
8. return  $\{Q'_2, \dots, Q'_h\}$ 
end
    
```

Fig. 3 The progressive query merging algorithm.

Algorithm Progressive Query Processing

Input: (1) $Q_t = \{q_1, q_2, \dots, q_m\}$: a set of merged queries stored into a sensor node, named s_node , at the t^{th} tier. /* $t \geq 2$ */
 (2) $D_t = \{d_1, d_2, \dots, d_n\}$: data generated by s_node .
 (3) $R_{t+1} = \{r_1, r_2, \dots, r_o\}$: data received from child sensor nodes at the $(t+1)^{th}$ tier of s_node .
Output: $R_t = \{r'_1, \dots, r'_k\}$: query processing result to be transmitted from s_node to the parent sensor node at the $(t-1)^{th}$ tier ($R_t \subseteq (D_t \cup R_{t+1})$).

Algorithm:

```

begin
1.  $D = D_t \cup R_{t+1}$ 
2. for each data element  $e_i$  in  $D$  begin
3. for each query  $q_j$  in  $Q_t$  begin
4. if the value  $e_i$  belongs to the query region of  $q_j$  begin
5. insert  $e_i$  into  $R_t$ 
6. break
7. end
8. end
9. end
10. return  $R_t$ 
end
    
```

Fig. 5 The progressive query processing algorithm.

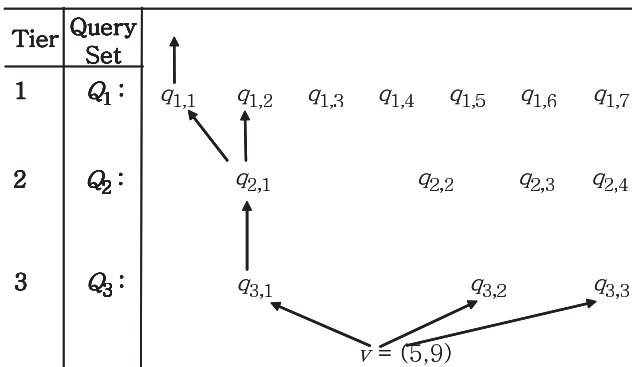


Fig. 4 An example of progressive query processing.

(server), while filtering the data elements depending on the result of evaluating the range predicates of the queries at each tier. (Precisely speaking, multiple data elements are sent in a batch for the sake of efficiency.)

Figure 4 shows an example of query processing. In this figure an arrow denotes an upward flow of a data item (v) as it satisfies the range predicate of the query at the arrow tail. In this example, the query $q_{1,1}$ at the server retrieves the data element v .

3.4.2 The Algorithm

Figure 5 shows the progressive query processing algorithm. The algorithm is run separately at each tier of the hierarchical sensor network. The algorithm is designed to run for each query on each data element, which may not be the most efficient in terms of the query processing time. However, the query processing time is independent of the energy cost and the storage cost which are the main cost items considered. Thus, it is not the focus of this paper.

In the progressive query processing, a sensor node at the t^{th} tier ($t \geq 2$) considers the data D_t generated by itself and the data R_{t+1} resulting from the query processing at the $(t+1)^{th}$ tier as the target data for query processing (line 1). The node compares the set of merged queries Q_t with the target data and inserts only the data elements that satisfy the query condition into R_t (lines 2-9). In order to prevent the node from sending duplicate results of overlapping query regions among merged queries, the algorithm stops the comparison once it finds a query whose region contains the target data element (line 6)[†] as is done by Xiang et al. [23]. Then, the node sends R_t to its parent node at the $(t-1)^{th}$ tier. This algorithm is run separately in every node at each tier to progressively filter the data to arrive at the highest

[†]When the algorithm is run at the server, Line 6 should be removed because the server must answer each query.

tier (i.e., server). Finally, the server (i.e., the 1st tier) performs post-processing to select the query results satisfying the condition of each query.

4. Determining the Optimal Number of Merged Queries

In this section, we propose an analytic method for determining the optimal number of merged queries to be stored at each tier when designing the hierarchical sensor network. We first propose the cost model in Sect. 4.1 and then the cost optimization method in Sect. 4.2.

4.1 The Cost Model

In this paper, we use the weighted sum of the storage cost for storing queries and the energy cost for sending the query result as the total cost. We use the total amount of memory used in all nodes as the storage cost and the total amount of data sent during the query processing as the energy cost. We use byte as the unit of both the storage cost and the energy cost.

Eq. (1) shows the cost model expressed as the function *weightd_sum*.

$$\begin{aligned} \text{Weighted_Sum} = & \alpha \cdot \text{the total amount of data sent} \\ & + \text{the total amount of memory used,} \end{aligned} \quad (1)$$

where $\alpha (> 0)$ is the scale factor provided by the user

In this equation, the value of α indicates the relative importance of the energy cost over the storage cost, and is set by the user based on one's preference. That is, in the environments where the energy cost is more important than the storage cost, the user gives a larger value of α , whereas in the environments where the storage cost is more important than the energy cost, the user gives a smaller value of α . In this paper, in order to control the trade-off between the two costs, we define the reference value of α , denoted as α_0 , which makes the importance of the two costs equal. This α_0 is the value for balancing between the two costs which use different scales, and is used as an example to determine the appropriate value of α for a given application. Eq. (2) shows the definition of α_0 :

$$\alpha_0 = \frac{\text{the maximum possible total amount of memory used}}{\text{the maximum possible total amount of data sent}} \quad (2)$$

In this equation, the denominator represents the total amount of data sent from sensor nodes when every node stores *only one* query merged from all the original queries, and the numerator represents the total amount of memory used for storing queries into sensor nodes when every node stores all the original queries. That is, α_0 is the result of dividing the worst case memory usage amount by the worst case data transmission amount.

In Eq. (1), the total memory usage amount is determined by the number of queries stored in the nodes at each tier, and the total data transmission amount is determined by the amount of data sent at each tier based on the queries. We first introduce the notion of the *merge rate* in order to formulate the number of queries stored in sensor nodes at each tier. We use it as the optimization parameter for the *Weighted_Sum*. The merge rate is defined as the ratio of the memory usage amounts of two nodes at adjacent tiers, as shown in Eq. (3).

$$\begin{aligned} \text{merge_rate} &= \frac{\text{the number of queries stored at a node at the } i^{\text{th}} \text{ tier}}{\text{the number of queries stored at a node at the } (i-1)^{\text{th}} \text{ tier}} \end{aligned} \quad (3)$$

for all $2 \leq i \leq h$, where h is the height of the hierarchical sensor network, and the server is at the first (highest) tier storing all the original queries.

According to the definition above, the merge rate has the value in the range of 0 to 1. If the value is closer to 0, it means that more queries are merged. On the other hand, if the value is closer to 1, it means that fewer queries are merged. That is, the number of queries stored in a node at each tier is determined by the merge rate. For example, if the merge rate is 0, our approach is equivalent to the centralized approach and if 1, it is equivalent to the distributed approach.

Next, we introduce the notion of *cover* to formulate the amount of data sent at each tier. The *cover* is defined as the ratio of the size of the domain space filled by all query regions over the size of the entire domain space. In order to obtain the exact amount of data transmission, we need additional information at each tier such as the selectivity of each merged query and the size of each dead region caused by query merging. This kind of information, however, is affected significantly by the application environment including the data and query distributions, making it difficult to obtain exact information at the time of designing the network. Thus, in this paper, we use an approximate model of the cover instead. Definition 2 shows the definition of the cover of a query set Q .

Definition 2 (The cover of a query set Q): For a given query set $Q = \{q_1, q_2, \dots, q_n\}$, its cover $\text{cover}(Q)$ is defined as:

$$\text{cover}(Q) = \frac{\|\Phi(q_1) \oplus \dots \oplus \Phi(q_n)\|}{\|D\|}$$

for $q_i, q_j \in Q (1 \leq i < j \leq n)$,

where D is the domain space,

$\Phi(q_i)$ is the region of the query q_i ,

$\Phi(q_i) \oplus \Phi(q_j)$ represents the union of the two regions

$\Phi(q_i)$ and $\Phi(q_j)$, and

$\|\cdot\|$ denotes the size of the given region. (4)

□

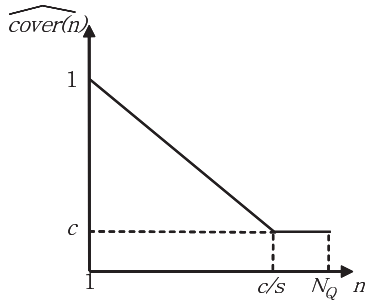


Fig. 6 The cover model.

Assuming that both queries and data are uniformly distributed in the same domain space, the amount of query results is approximately proportional to $cover(Q)$ because the cover represents the size of the union of the query regions (simply, the *union query region*) when the size of domain space is 1 in Definition 2.

Now, we approximate the cover of merged queries stored at the nodes at each tier as follows. Let N_Q denote the number of original queries in Q , $n (\leq N_Q)$ the number of merged queries, s the average selectivity of the set of the original queries, and c the cover of the set of the original queries, then $\widehat{cover}(n)$ in Fig. 6 is an approximation of $cover(Q)$.

$\widehat{cover}(n)$ has the following properties: (1) If $n = 1$, $\widehat{cover}(n)$ equals 1; (2) As n increases, $\widehat{cover}(n)$ decreases becoming c when $n = \frac{c}{s}$. That is, $\widehat{cover}(n) \leq \widehat{cover}(n - 1) \leq \dots \leq \widehat{cover}(1) = 1$.

These properties are from fact that the proposed merge method is based on MBR. Since the region of a merged query is represented by an MBR enclosing the regions of the queries to be merged, the size of the region of the merged query is always greater than or equal to that of the union of the regions of the queries merged. Thus, as the query merging proceeds, the number of merged queries n decreases, but the size of the region that is equivalent to the union of merged queries increases. In this paper we have assumed an environment in which we process a large number of queries with the uniform distribution, and thus, we assume that, when all queries are merged into one query, the cover of the merged query is 1. Even though this property does not guarantee the linearity of $\widehat{cover}(n)$, in order to make the model simple, we assume that the cover linearly increases as n decreases, and then, estimate the theoretical number of queries for which the cover is completely filled without overlap region as $\frac{c(\text{the cover of original queries})}{s(\text{the average selectivity of original queries})}$.

4.2 Optimization

In this subsection, we first formulate *Weighted_Sum* using the *merge_rate* and the cover model explained in Sect. 4.1, and then, analytically obtain the optimal merge rate – the merge rate that minimizes *Weighted_Sum*. For ease of exposition, we assume that each sensor node generates only one data element per unit time and both data and queries are

Table 1 The notation.

Symbol	Definition
N_Q	The number of original queries
α	The scale factor provided by the user
c	The cover of original queries
s	The average selectivity of original queries
d	The dimension of original queries
h	The height of a hierarchical sensor network
f	The fanout of a hierarchical sensor network
$Size_{de}$	The size of a data element
m	The merge rate

uniformly distributed in the domain space. We also assume that the range predicate of a query has start and end values. Table 1 shows the notation used in this section.

The *total_transmission* (i.e., the total amount of data sent per unit time) is the cost of sending the query results at all nodes at all tiers to the server. The cost is represented as the sum of the amount of the query results. In this section, we represent *total_transmission* as the sum of the amount of data (i.e., query result) sent from the nodes at each tier to the nodes at their parent tier. We denote the amount of the data sent at each tier as the *tier_transmission_i*. Thus, the *total_transmission* is formulated as the summation of *tier_transmission_i*'s. As we have explained in Sect. 3.1, in the query processing phase, a node processes its own sensed data and the data received from the nodes at the next lower tier. Thus, the amount of query results that the node sends to its parent node is equal to the sum of (1) the amount of query results on the sensed data (simply, *Amt_Qresult_{own_data}*) and (2) that of the query results on the received data (simply, *Amt_Qresult_{rcv_data}*). Here, by the cover property, (1) *Amt_Qresult_{own_data}* at the i^{th} -tier nodes is computed as $c_i \cdot Amt_Data_i$, where c_i is the cover of the merged queries at the i^{th} tier, and *Amt_Data_i* is the amount of sensed data at the nodes of the i^{th} tier, and (2) *Amt_Qresult_{rcv_data}* at the i^{th} -tier nodes is computed as $\frac{c_i}{c_{i+1}} \cdot tier_transmission_i$. We note that the size c_i of the union query region at the i^{th} tier is smaller than the size c_{i+1} of that at the $(i + 1)^{th}$ tier since a query at a lower tier is represented by an MBR that encloses certain queries at a higher tier, and the received data from the $(i + 1)^{th}$ tier fall inside the union query region whose size is c_{i+1} as explained in Sect. 3.4. Hence, at the i^{th} tier, the amount of query results on the received data from the $(i + 1)^{th}$ tier is reduced in proportion to $\frac{c_i}{c_{i+1}}$.

The *total_storage* (i.e., the total amount of memory used) is the cost of storing merged queries at all nodes at all tiers. The cost is represented as the sum of the amount of memory used to store the merged queries. In this section, we represent *total_storage* as the sum of the amount of memory (i.e., the memory used to store merged queries) used by all nodes at each tier. We denote the amount of memory used by each tier as *tier_storage_i*. Thus, the *total_storage* is formulated as the summation of *tier_storage_i*'s. As we have shown in Table 1, each node except for the nodes at the leaf tier has f (i.e., the fanout of a hierarchical sensor

network) child nodes. Thus, $tier_storage_i$ is computed as $f^{i-1} \cdot Amt_Mem_i$ where Amt_Mem_i is the amount of memory used to store merged queries at a node of the i^{th} tier. Here, we show the formulas for the total_transmission and total_storage as follows.

$$\begin{aligned}
 total_transmission &= \sum_{i=2}^h (tier_transmission_i) \\
 tier_transmission_i &= \begin{cases} c_i \cdot Amt_Data_i \\ + \frac{c_i}{c_{i+1}} \cdot tier_transmission_{i+1} & (2 \leq i < h) \\ c_i \cdot Amt_Data_i & (i = h) \end{cases} \\
 &= \sum_{j=i}^h (c_j \cdot Amt_Data_j) \quad (5)
 \end{aligned}$$

$$\begin{aligned}
 total_storage &= \sum_{i=2}^h (tier_storage_i) \\
 tier_storage_i &= f^{i-1} \cdot Amt_Mem_i \quad (2 \leq i \leq h) \quad (6)
 \end{aligned}$$

From Eq. (5) and Eq. (6), Weighted_Sum is formulated as follows (refer to Appendix for details).

$$\begin{aligned}
 Weighted_Sum &= \alpha \cdot total_transmission + total_storage \\
 &= Size_{de} \cdot \sum_{i=2}^h \left(f^{i-1} \cdot \left[\alpha \cdot \sum_{j=2}^i (-a \cdot m^{j-1} \cdot N_Q + b) \right. \right. \\
 &\quad \left. \left. + 2 \cdot N_Q \cdot m^{i-1} \right] \right) \\
 \text{where } a &= \frac{s \cdot (1 - c)}{c - s}, \text{ and } b = 1 + a \quad (7)
 \end{aligned}$$

In order to obtain the optimal merge rate, we take the derivative of the Weighted_Sum formula with respect to m and compute the roots from the derivative formula. Then, we substitute each root for m in the formula, and find the root that minimizes the computed Weighted_Sum. We use Maple [11], a mathematics software tool, for this computation.

5. Performance Evaluation

5.1 Experimental Data and Environments

We use two sets of experiments. In the first set, we show the accuracy of the proposed cost model as the parameters are varied. In the second set, we show the merit of our progressive approach over the iterative approach proposed by Xiang et al. [21] in terms of the total cost (i.e., the weighted sum) of query processing as the parameters are varied. A common set of seven parameters are used in both sets of experiments: the scale factor α for controlling the ‘‘importance’’ between the amount of data transmission and the amount of memory

usage, the cover of original queries c , the average selectivity of original queries s , the dimension of original queries d , the height of the sensor network h , the fanout of the sensor network f , and merge rate m . We use the optimal merge rate as the accuracy measure and the weighted sum (WS) as the performance measure.

We use the same data and query sets in both sets of experiments. We randomly generate synthetic queries and data with the uniform distribution. Here, ‘‘uniform’’ means that the locations of the queries (or the data elements) are set randomly in the query space (or the domain space). We generate queries with the same width in all domains (i.e., hypercubes) in two alternative ways: either by controlling the number of original queries or by controlling the cover of original queries. The latter is used only in the experiments for varying the cover of original queries, and the former is used in all the other experiments. The reason we do not control the number and the cover of the queries together is that there is a dependency between the two values. That is, given a set of random queries with a uniform distribution, if the number of queries increases (with the query selectivity fixed) then the cover also increases. This makes it impossible to generate a query set with a uniform distribution when both number and cover are controlled at the same time.

In the first set of experiments, we experimentally evaluate the accuracy of our model for estimating the optimal merge rate that minimizes the weighted sum of the storage cost and the energy cost (i.e., Eq. (1)). We first analytically compute the estimated optimal merge rate as explained in Sect. 4.2. Next, we experimentally find the actual optimal merge rate. Finally, we compare the two optimal merge rates.

In the second set of experiments, we compare the performance merit of our progressive approach with the iterative approach proposed by Xiang et al. [21]. We measure the weighted sums while varying parameters explained above. Here, in our approach, we use the estimated optimal merge rates measuring the weighted sums while varying parameters explained above. Table 2 summarizes all the experiments and the parameters used.

All experiments have been conducted using a Linux-Redhat system with a 4 GHz processor and 1 Gbytes of main memory. Since it is difficult to build an actual large-scale sensor network and change its configuration as we need, in order to conduct the experiments, we have implemented a simulator program using C as in existing sensor networks-related database research [8], [10], [21]. Table 3 summarizes the notation used in the next section to discuss the experimental results.

5.2 Experimental Results

5.2.1 Accuracy of the Cost Model

Experiment 0: existence of the trade-off and the optimal merge rate

Figure 7(a) shows the trade-off between the total storage

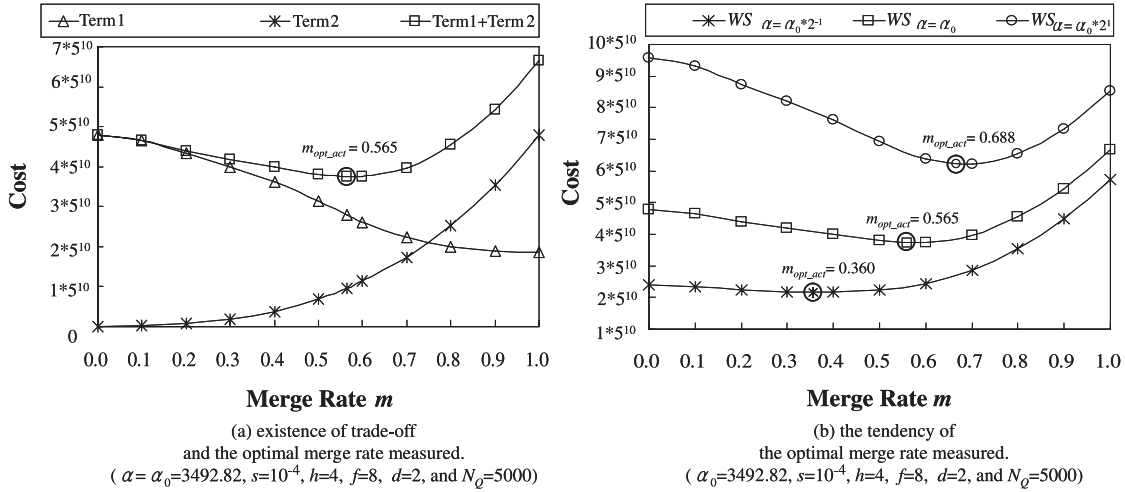


Fig. 7 The existence of trade off and the optimal merge rate measured.

Table 2 Experiments and parameters used for showing the accuracy of the cost model and the performance merit of our approach.

Experiments		Parameters	
Exp's. 1 and 7 accuracy (Exp. 1) and performance (Exp. 7) as α is varied	h		4
	f		8
	α		$\alpha_0 \cdot 2^{-2}, \alpha_0 \cdot 2^{-1}, \alpha_0, \alpha_0 \cdot 2^1, \alpha_0 \cdot 2^2$
	s		10^{-4}
	d		2
Exp's. 2 and 8 accuracy (Exp. 2) and performance (Exp. 8) as c is varied	h		4
	f		8
	α		α_0
	d		2
	c		0.01, 0.05, 0.10, 0.39, 0.50, 0.99
Exp's. 3 and 9 accuracy (Exp. 3) and performance (Exp. 9) as s is varied	h		4
	f		8
	α		α_0
	s		$10^{-3}, 10^{-4}, 10^{-5}$
	d		2
Exp's. 4 and 10 accuracy (Exp. 4) and performance (Exp. 10) as h is varied	h		3, 4, 5
	f		8
	α		α_0
	s		10^{-4}
	d		2
Exp's. 5 and 11 accuracy (Exp. 5) and performance (Exp. 11) as f is varied	h		4
	f		2, 4, 8, 16
	α		α_0
	s		10^{-4}
	d		2
Exp's. 6 and 12 accuracy (Exp. 6) and performance (Exp. 12) as d is varied	h		4
	f		8
	α		α_0
	s		10^{-4}
	d		1, 2, 3

cost and the total transmission cost (i.e., energy cost) as m is varied. In Fig. 7 (a), Term1 represents the first term of Eq. (1), Term2 represents the second term of Eq. (1), and Term1+Term2 represents Eq. (1). Here, we measure the cost for 5000 randomly generated queries (i.e., $N_Q=5000, s=10^{-4}, c=0.39, \text{ and } \alpha_0=3492.82$). Hereafter, we use $N_Q=5000$ unless otherwise specified. As explained in Sect. 3.1, Term1 (i.e., $\alpha \cdot \text{total_transmission}$) has a tendency

Table 3 Notation for explaining experiments.

Symbol	Definition
m_{opt_act}	The actual optimal merge rate measured
m_{opt_est}	The estimated optimal merge rate obtained using the analytical model
$accuracy_m^*$	The accuracy of m_{opt_est} over $m_{opt_act} = 1 - \frac{ m_{opt_act} - m_{opt_est} }{m_{opt_act}}$
$accuracy_{ws}$	The accuracy of the weighted sum measured using $m_{opt_est}(WS_{m=m_{opt_est}})$ over the weighted sum measured using $m_{opt_act}(WS_{m=m_{opt_act}}) = 1 - \frac{ WS_{m=m_{opt_act}} - WS_{m=m_{opt_est}} }{WS_{m=m_{opt_act}}}$
$gain_{ws}$	$\frac{\text{(the weighted sum measured using Xiang et al.'s iterative approach)}}{WS_{m=m_{opt_est}}}$

*When $accuracy_m < 0$, it implies that m_{opt_est} is larger than m_{opt_act} by $(|accuracy_m|+2)$ times according to the definition of $accuracy_m$.

to decrease as m increases. Term2 (i.e., total_storage) has a tendency to increase as m does. Thus, a value of m that minimizes Term1+Term2 (i.e., the weighted sum) exists as shown in Fig. 7 (a). Figure 7 (b) shows the trend of the optimal merge rate as m is varied. Here, $WS_{\alpha=\alpha_0 \cdot 2^{-1}}, WS_{\alpha=\alpha_0}$, and $WS_{\alpha=\alpha_0 \cdot 2^1}$ represent the weighted sum measured using $\alpha = \alpha_0 \cdot 2^{-1}, \alpha = \alpha_0, \text{ and } \alpha = \alpha_0 \cdot 2^1$, respectively. We observe that the optimal merge rate has a tendency to increase as α does.

Experiment 1: accuracy as α is varied

Figure 8 shows experimental results as α is varied. Here, the value of α_0 is 3492.82. We have different optimal merge rates for different scale factors as shown in this figure. In Fig. 8, $WS_{m=m_{opt_est}}$ and $WS_{m=m_{opt_act}}$ represent the weighted sum measured using m_{opt_est} and that measured using m_{opt_act} , respectively. From Fig. 8, we see that $accuracy_m$ ranges from 0.610 to 0.864, and $accuracy_{ws}$ ranges from 0.950 to 0.984. We also see that, as α increases, the total transmission cost becomes prevailing over the total storage cost, and thus, the optimal merge rate is determined towards reducing the total transmission cost, i.e., is made close to 1.0.

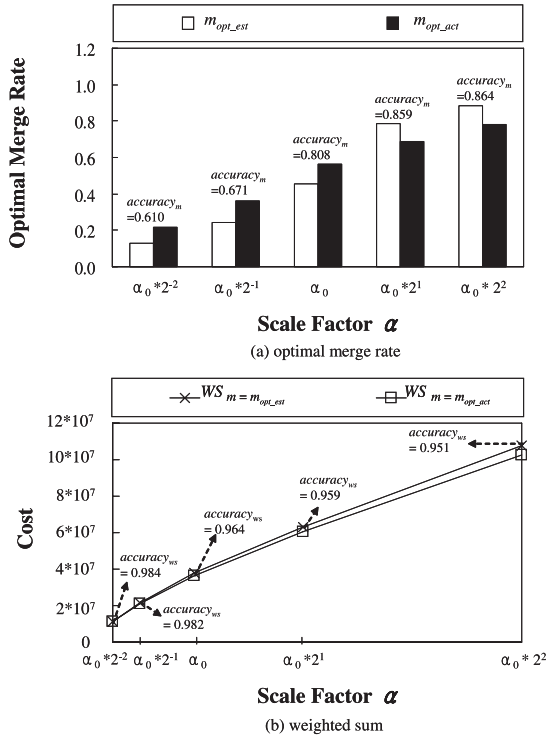


Fig. 8 The accuracy of our model as α is varied ($\alpha_0=3492.82$, $s=10^{-4}$, $h=4$, $f=8$, $d=2$, and $N_Q=5000$).

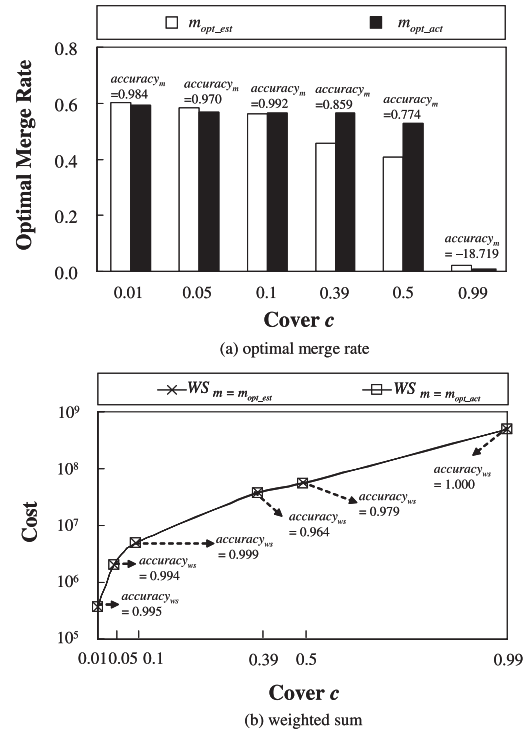


Fig. 9 The accuracy of our model as c is varied ($\alpha = \alpha_0$, $s = 10^{-4}$, $h=4$, $f=8$, and $d=2$).

Experiment 2: accuracy as c is varied

Figure 9 shows the experimental results as the cover is varied. The query sets for different covers are $N_Q=101$ and $\alpha_0=70.56$ when $c=0.01$, $N_Q=513$ and $\alpha_0=358.36$ when $c=0.05$, $N_Q=1046$ and $\alpha_0=731.00$ when $c=0.10$, $N_Q=5000$ and $\alpha_0=3492.82$ when $c=0.39$, $N_Q=6916$ and $\alpha_0=4831.27$ when $c=0.50$, and $N_Q=52685$ and $\alpha_0=36803.88$ when $c=0.99$. From Fig. 9, we see that $accuracy_m$ ranges from -18.719 to 0.992 . Other than the value -18.719 (i.e., m_{opt_est} is larger than m_{opt_act} by $(18.719+2)$ times as we have explained in the footnote in Page 11.) when the cover is 0.99 , $accuracy_m$ is larger than 0.774 for all the other values of the cover. That is, the optimal merge rates measured from the experiments are similar to those obtained from the analysis. Besides, we see that $accuracy_{WS}$ ranges from 0.964 to 1.000 . That is, the weighted sum measured from the experiments is very close to that obtained from the analysis. As the cover of the original queries approaches 1.0 , it becomes similar to that of the merged queries because the union of the original query regions is similar to the domain space by the definition of the cover. In this case, although the number of merged queries changes, the total transmission cost hardly does. Thus, the total data transmission cost has no significant influence on the total cost. Hence, the optimal merge rate is determined towards reducing the total storage cost, i.e., is made close to 0 .

Experiment 3: accuracy as s is varied

Figure 10 shows the experimental results as the selectivity is varied. Here, the value of α_0 is 3492.82 . In Fig. 10, we see

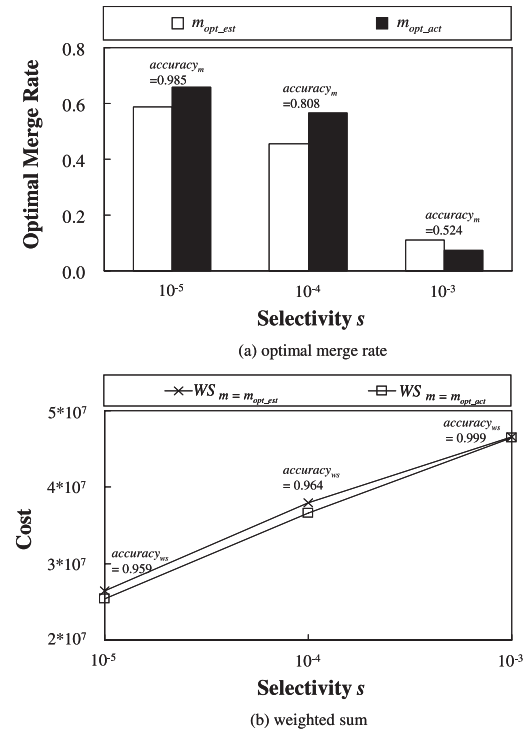


Fig. 10 The accuracy of our model as s is varied ($\alpha = \alpha_0=3492.82$, $h=4$, $f=8$, $d=2$, and $N_Q=5000$).

that $accuracy_m$ ranges from 0.524 to 0.895 , and $accuracy_{WS}$ ranges from 0.959 to 0.999 . Increase of the selectivity is closely related to increase of the cover. That is, if the selec-

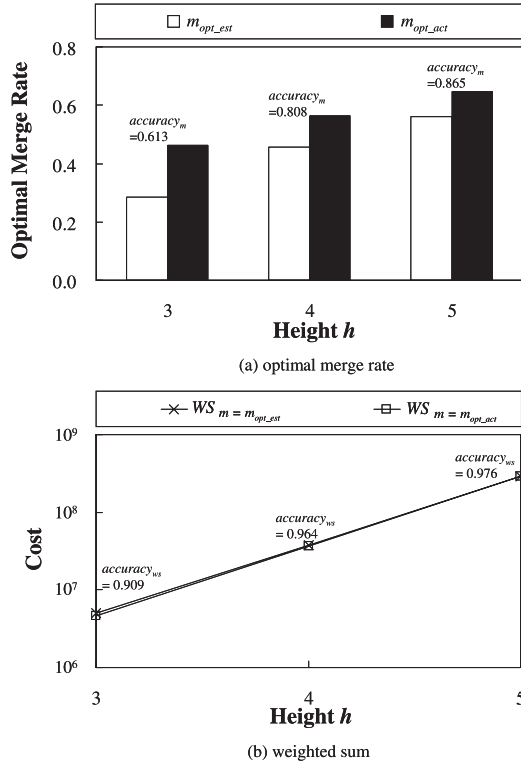


Fig. 11 The accuracy of our model as h is varied ($\alpha = \alpha_0, s = 10^{-4}, f=8, d=2$, and $N_Q=5000$).

tivity increases while the number of queries is fixed, then the cover of the original queries increases as well, and thus, like the case of varying the cover, the optimal merge rate moves close to 0.

Experiment 4: accuracy as h is varied

Figure 11 shows the experimental results as the height is varied. Here, we use $\alpha_0=5294.12$ when $h=3$, $\alpha_0=3492.82$ when $h=4$, and $\alpha_0=2592.94$ when $h=5$. We see that $accuracy_m$ ranges from 0.613 to 0.865, and $accuracy_{ws}$ ranges from 0.909 to 0.976. When the height of the sensor network increases, the data transmission cost increases faster than the memory usage cost. This stems from the fact that the data sent are accumulated at each tier. Thus, the optimal merge rate moves towards reducing the total data transmission cost, i.e., is made close to 1.0.

Experiment 5: accuracy as f is varied

Figure 12 shows the experimental results as the fanout is varied. Here, we use $\alpha_0=4117.65$ when $f=2$, $\alpha_0=3684.21$ when $f=4$, $\alpha_0=3492.82$ when $f=8$, and $\alpha_0=3408.24$ when $f=16$. We see that $accuracy_m$ ranges from 0.671 to 0.808, and $accuracy_{ws}$ ranges from 0.950 to 0.967. For the same reason as explained in Experiment 4, the optimal merge rate has a tendency to move close to 1.0 as f increases.

Experiment 6: accuracy as d is varied

Figure 13 shows the experimental results as the dimension is varied. Here, the value of α_0 is 3492.82. We see that $accuracy_m$ ranges from 0.707 to 0.945, and $accuracy_{ws}$ ranges from 0.964 to 0.996.

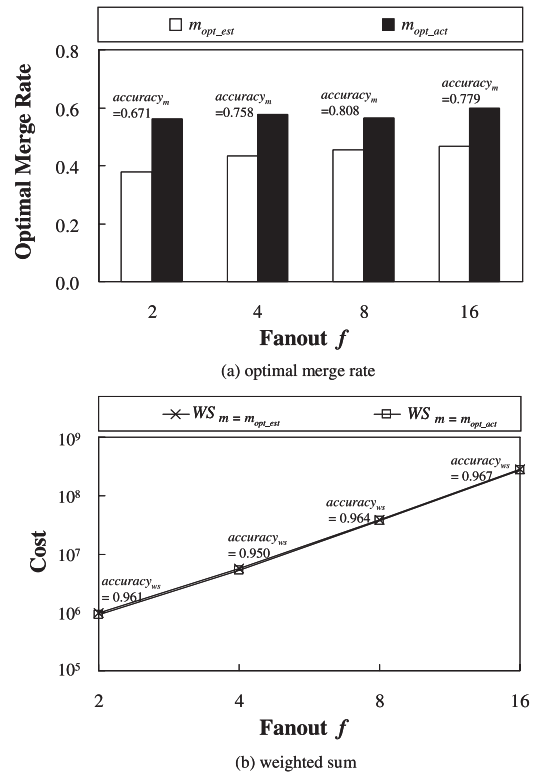


Fig. 12 The accuracy of our model as f is varied ($\alpha = \alpha_0, s = 10^{-4}, h=4, d=2$, and $N_Q=5000$).

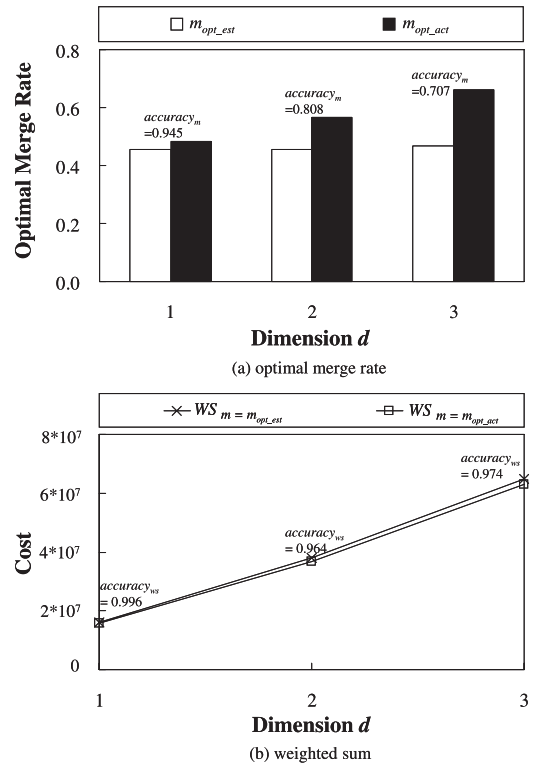


Fig. 13 The accuracy of our model as d is varied ($\alpha = \alpha_0=3492.82, s = 10^{-4}, h=4, f=8$, and $N_Q=5000$).

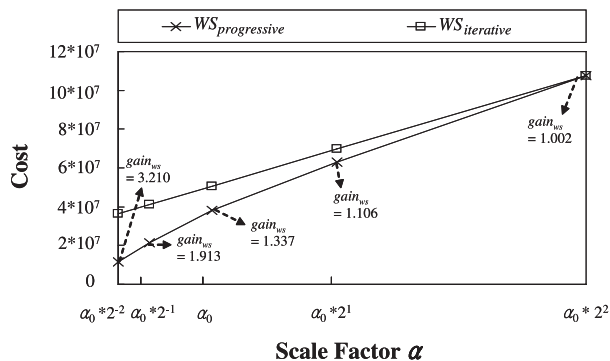


Fig. 14 The performance of progressive and iterative approaches as α is varied ($\alpha_0=3492.82$, $s=10^{-4}$, $h=4$, $f=8$, $d=2$, and $N_Q=5000$).

5.2.2 Performance Merit of our Approach

Experiment 7: performance as α is varied

Figure 14 shows the experimental result as α is varied. Here, the value of α_0 is equal to that used in Experiment 1. In Fig. 14, $WS_{progressive}$ represents the weighted sum measured in the progressive approach (i.e., our approach), and $WS_{iterative}$ represents the weighted sum measured in the iterative approach (i.e., Xiang et al.'s approach [21]). The optimal merge rates estimated for different scale factors are $m_{opt_est}=0.132$ when $\alpha=\alpha_0 \cdot 2^{-2}$, $m_{opt_est}=0.242$ when $\alpha=\alpha_0 \cdot 2^{-1}$, $m_{opt_est}=0.456$ when $\alpha=\alpha_0$, $m_{opt_est}=0.785$ when $\alpha=\alpha_0 \cdot 2^1$, and $m_{opt_est}=0.886$ when $\alpha=\alpha_0 \cdot 2^2$. The weighted sum measured in the progressive approach has a tendency to linearly increase as α increases since it is a linear function of α in Eq. (7). From this figure, we see that $gain_{ws}$ (i.e., the ratio of $WS_{iterative}$ to $WS_{progressive}$) ranges from 1.002 to 3.210. That is, the weighted sum in the progressive approach is smaller than that in the iterative approach because our progressive approach can near-optimally reduce the weighted sum by controlling the merge rate. From these results, we see that the progressive approach improves the performance over the iterative approach when memory usage is the prevailing cost (i.e., α is small), while giving a competitive performance when data transmission is the prevailing cost (i.e., α is large).

Experiment 8: performance as c is varied

Figure 15 shows the experimental result as the cover is varied. Here, the values of α_0 are equal to those used in Experiment 2. The optimal merge rates estimated for different covers are $m_{opt_est}=0.602$ when $c=0.01$, $m_{opt_est}=0.585$ when $c=0.05$, $m_{opt_est}=0.562$ when $c=0.10$, $m_{opt_est}=0.456$ when $c=0.39$, $m_{opt_est}=0.409$ when $c=0.50$, and $m_{opt_est}=0.021$ when $c=0.99$. The query sets for different covers are $N_Q=101$ when $c=0.01$, $N_Q=513$ when $c=0.05$, $N_Q=1046$ when $c=0.10$, $N_Q=5000$ when $c=0.39$, $N_Q=6916$ when $c=0.50$, and $N_Q=52685$ when $c=0.99$. We see that $gain_{ws}$ ranges from 1.019 to 2.498. This result shows that the progressive approach outperforms the iterative approach in the entire range of the cover. It also shows that, as the cover

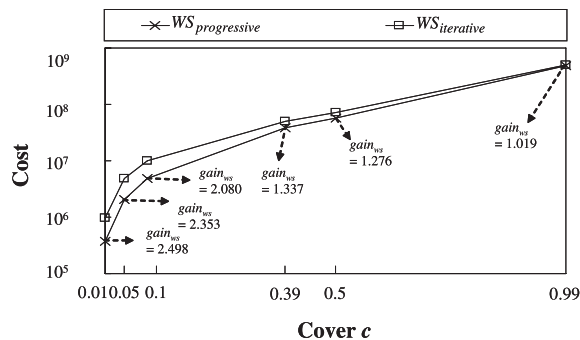


Fig. 15 The performance of progressive and iterative approaches as c is varied ($\alpha=\alpha_0$, $s=10^{-4}$, $h=4$, $f=8$, and $d=2$).

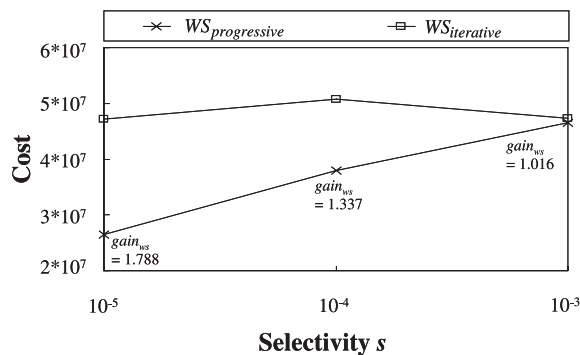


Fig. 16 The performance of progressive and iterative approaches as s is varied ($\alpha=\alpha_0=3492.82$, $h=4$, $f=8$, $d=2$, and $N_Q=5000$).

increases, the performance benefit of our approach over the iterative approach decreases. When the cover of the original queries approaches 1.0, all the original queries are merged into one query in both the progressive approach and the iterative approach; as a result, the total transmission amounts and the total storage amounts of the two approaches become similar and, therefore, the weighted sums of the two approaches become similar as well. Our proposed approach shows more performance benefit when the cover of the original queries is smaller, which is the case more likely to happen in a real environment.

Experiment 9: performance as s is varied

Figure 16 shows the experimental result as the average selectivity is varied. Here, the value of α_0 is equal to that used in Experiment 3. The optimal merge rates estimated for different selectivities are $m_{opt_est}=0.589$ when $s=10^{-5}$, $m_{opt_est}=0.456$ when $s=10^{-4}$, and $m_{opt_est}=0.111$ when $s=10^{-3}$. We see that $gain_{ws}$ ranges from 1.016 to 1.788. This result shows that the progressive approach outperforms the iterative approach in the entire range of selectivity. It also shows that, as the selectivity increases, the performance benefit of the progressive approach decreases. As already mentioned in Experiment 3, if the selectivity increases, then the cover increases as well causing the decrease of performance benefit. Thus, our proposed approach shows more performance benefit when the selectivity of the original queries is smaller.

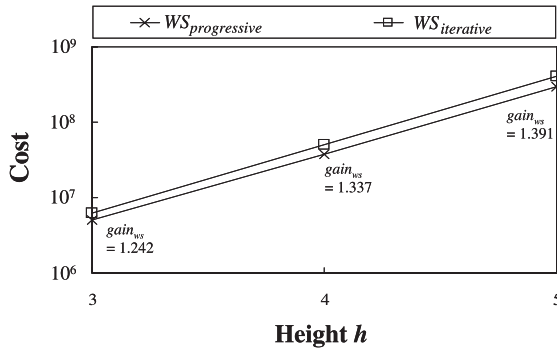


Fig. 17 The performance of progressive and iterative approaches as h is varied ($\alpha = \alpha_0$, $s = 10^{-4}$, $f=8$, $d=2$, and $N_Q=5000$).

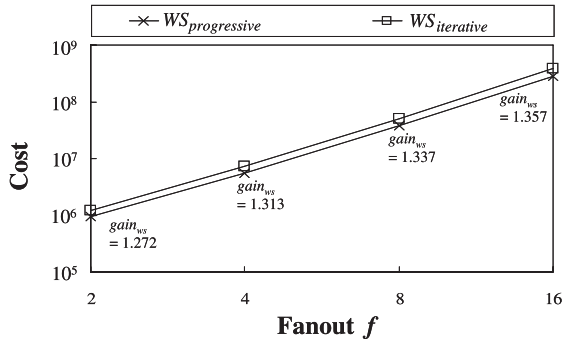


Fig. 18 The performance of progressive and iterative approaches as f is varied ($\alpha = \alpha_0$, $s = 10^{-4}$, $h=4$, $d=2$, and $N_Q=5000$).

Experiment 10: performance as h is varied

Figure 17 shows the experimental result as the height of the sensor network is varied. Here, the values of α_0 are equal to those used in Experiment 4. The optimal merge rates estimated for different heights are $m_{opt_est}=0.284$ when $h=3$, $m_{opt_est}=0.456$ when $h=4$, and $m_{opt_est}=0.560$ when $h=5$. We see that $gain_{ws}$ ranges from 1.242 to 1.391. This result shows that the progressive approach outperforms the iterative approach in the entire range of the height.

Experiment 11: performance as f is varied

Figure 18 shows the experimental result as the fanout of the sensor network is varied. Here, the values of α_0 are equal to those used in Experiment 5. The optimal merge rates estimated for different fanouts are $m_{opt_est}=0.377$ when $f=2$, $m_{opt_est}=0.434$ when $f=4$, $m_{opt_est}=0.456$ when $f=8$, and $m_{opt_est}=0.466$ when $f=16$. In Fig. 18, $gain_{ws}$ ranges from 1.272 to 1.357; thus, for all ranges of f , the progressive approach outperforms that of the iterative approach.

Experiment 12: performance as d is varied

Figure 19 shows the experimental result as the dimension of a query is varied. Here, the value of α_0 is equal to that used in Experiment 6. The optimal merge rates estimated for different dimensions are $m_{opt_est}=0.455$ when $d=1$, $m_{opt_est}=0.456$ when $d=2$, and $m_{opt_est}=0.469$ when $d=3$. In Fig. 19, $gain_{ws}$ ranges from 1.018 to 2.366; thus, for all ranges of d , the progressive approach outperforms that of the iterative approach.

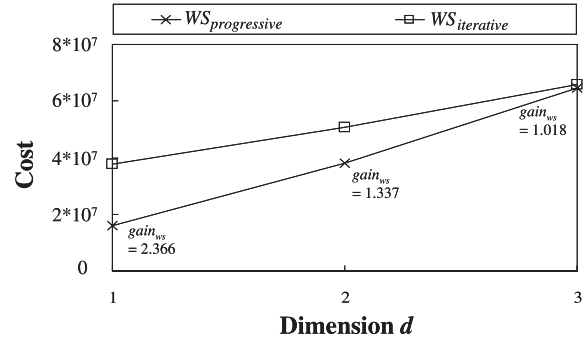


Fig. 19 The performance of progressive and iterative approaches as d is varied ($\alpha = \alpha_0=3492.82$, $s = 10^{-4}$, $h=4$, $f=8$, and $N_Q=5000$).

In summary, the experimental results show that our cost model gives accurate optimal merge rates, and the progressive approach outperforms the iterative approach by up to 3.210 times as the parameters such as α , the cover, average selectivity, dimension of original queries, and height and fanout of the sensor network vary.

6. Conclusions

In this paper, we have proposed *progressive processing* as a new approach to processing multiple continuous range queries in hierarchical sensor networks. The contribution of this paper are summarized as follows.

First, we have proposed a progressive processing model that considers the trade-off between energy and storage. This model takes advantage of the characteristics of the hierarchical sensor networks in which higher-capability sensor nodes are deployed at a tier closer to the server. It also has the advantage of reducing the cost of building the network by reducing the storage cost at lower tier nodes, which are larger in number. We have also presented query merging and query processing algorithms for this model.

Second, based on the proposed model, we have proposed a method for optimizing the total cost (formulated as the weighted sum of the energy and storage costs) according to the given weight, and have proposed a method for systematically building a hierarchical sensor network that minimizes the total cost.

Third, we have verified the merit of the proposed approach through extensive experiments. In the experiments for evaluating the accuracy of the proposed cost model, the results show that the accuracy of the optimal cost obtained from the analytical cost model over the optimal cost measured is 0.950 to 1.0. From these results we see that a hierarchical sensor network with near-optimal total cost can be built using the proposed model. In the experiments for evaluating the query processing performance, the results show that our approach outperforms the approach proposed by Xiang et al. [21] by up to 3.210 times. Moreover, if the height of the sensor network increases, our approach shows a better performance than Xiang et al.'s approach. Thus, we can see that our approach is suitable for a *large-scale* sensor network.

In conclusion, our approach provides a new framework for building a large-scale hierarchical sensor network that efficiently processes a large number of queries while considering the trade-off between the energy consumed and the storage required.

For further work, we plan to improve the query processing model and algorithms to consider different data and query distributions as well as different query types such as aggregate queries.

Acknowledgements

This work was partially supported by the National Research Foundation of Korea (NRF) through the National Research Lab Program (No.R0A-2007-000-20101-0) and Microsoft Research Asia.

References

- [1] S. Abiteboul et al., "The lowell database research self-assessment," *Commun. ACM (CACM)*, vol.48, no.5, pp.111–118, May 2005.
- [2] I.F. Akyildiz, T. Melodia, and K.R. Chowdhury, "A survey on wireless multimedia sensor networks," *Comput. Netw.*, vol.51, no.4, pp.921–960, March 2007.
- [3] A. Arasu et al., "STREAM: The stanford stream data manager," *IEEE Data Engineering Bulletin*, vol.26, no.2, pp.19–26, March 2003.
- [4] A. Arasu, S. Bahu, and J. Widom, "The CQL continuous query language: semantic foundations and query execution," *VLDB J.*, vol.15, no.2, pp.121–142, June 2006.
- [5] EarthNet News (available at <http://earth.esa.int/object/index.cfm?fobjectid=5106>).
- [6] S. Ganu, S. Zhao, L. Raju, B. Anepu, I. Seskar, and D. Raychaudhuri, Architecture and Prototyping of an 802.11-Based Self-Organizing Hierarchical Adhoc Wireless Network (SOHAN), White Paper, Rutgers University, March 2004.
- [7] C. Gui and P. Mohapatra, "Power conservation and quality of surveillance in target tracking sensor networks," *Proc. 10th Annual Int'l Conf. Mobile Computing and Networking (MOBICOM)*, pp.129–143, Philadelphia, PA, Sept. 2004.
- [8] X. Li, Y.-J. Kim, R. Govindan, and W. Hong, "Multi-dimensional range queries in sensor networks," *Proc. 1st Int'l Conf. Embedded Networked Sensor Systems (ACM Sensys)*, pp.63–75, Los Angeles, California, Nov. 2003.
- [9] H. Lim, J. Lee, M. Lee, K. Whang, and I. Song, "Continuous query processing in data streams using duality of data and queries," *Proc. 2006 ACM SIGMOD Int'l Conf. Management of Data*, pp.313–324, Chicago, Illinois, June 2006.
- [10] S.R. Madden, M.J. Franklin, J.M. Hellerstein, and W. Hong, "TinyDB: An acquisitional query processing system for sensor networks," *ACM Trans. Database Systems (TODS)*, vol.30, no.1, pp.122–173, March 2005.
- [11] Maple Software (available at <http://www.maplesoft.com>).
- [12] R. Muller and G. Alonso, "Efficient sharing of sensor networks," *Proc. 3rd IEEE Int'l Conf. Mobile Ad-hoc and Sensor Systems (MASS)*, pp.109–118, Vancouver, Canada, Oct. 2006.
- [13] NOVAC project (available at <http://www.novac-project.eu>).
- [14] C. Palo, "Towards next generation wireless sensor networks – rethinking middleware design," Nov. 2007 (available at research.microsoft.com/en-us/um/people/costa/getslides.aspx?doc=midsens07.pdf).
- [15] S.M. Park, "Technical trend of sensor network node platform & OS," *Electronics and Telecommunications Trends, ETRI*, vol.21, no.1, pp.14–24, Feb. 2006.
- [16] Z. Qingguang, C. Yanling, and L. Juan, "A lightweight key management protocol for hierarchical sensor networks," *Proc. 7th Int'l Conf. Parallel and Distributed Computing, Applications and Technologies (PDCAT)*, pp.379–382, Taipei, Taiwan, Dec. 2006.
- [17] S. Ratnasamy, B. Karp, S. Shenker, D. Estrin, R. Govindan, L. Yin, and F. Yu, "Data-centric storage in sensor networks with ght, a geographical hash table," *Mobile Networks and Applications*, vol.8, no.4, pp.427–442, Aug. 2003.
- [18] M. Singh and V.K. Prasanna, "A hierarchical model for distributed collaborative computation in wireless sensor networks," *Proc. 17th Int'l Symposium on Parallel and Distributed Processing (IPDPS)*, pp.166–258, Nice, France, April 2003.
- [19] U. Srivastava, K. Munagala, and J. Widom, "Operator placement for in-network stream query processing," *Proc. 24th ACM SIGACT-SIGMOD-SIGART Symposium on Principle of Database Systems (PODS)*, pp.250–258, Baltimore, Maryland, USA, June 2005.
- [20] N. Trigoni, Y. Yao, A.J. Demers, J. Gehrke, and R. Rajaraman, "Multi-query optimization for Sensor Networks," *Proc. 1st Int'l Conf. Distributed Computing in Sensor Systems (DCOSS)*, pp.307–321, Marina del Rey, CA, July 2005.
- [21] S. Xiang, H.B. Lim, and K.-L. Tan, "Impact of multi-query optimization in sensor networks," *Proc. Workshop on Data Management in Sensor Networks (DMSN'06)*, pp.7–12, Seoul, Korea, Sept. 2006.
- [22] S. Xiang, H.B. Lim, and K.-L. Tan, "Multiple query optimization for wireless sensor networks," *Proc. International Conference on Data Engineering (ICDE'07)*, pp.1339–1341, Istanbul, Turkey, April 2007.
- [23] S. Xiang, H.B. Lim, K.-L. Tan, and Y. Zhou, "Two-tier multiple query optimization for sensor networks," *Proc. 27th International Conference on Distributed Computing Systems (ICDCS'07)*, p.39, Toronto, CA, June 2007.
- [24] Y. Yao and J. Gehrke, "The cougar approach to in-network query processing in sensor networks," *SIGMOD Record*, vol.31, no.1, pp.9–18, March 2002.
- [25] W. Yu, T.N. Le, D. Xuan, and W. Zhao, "Query aggregation for providing efficient data services in sensor networks," *Proc. 1st IEEE Int'l Conf. Mobile Ad-hoc and Sensor Systems (MASS)*, pp.31–40, Florida, Oct. 2004.

Appendix: Derivation of the Formulas for Total Transmission and Total Storage

Derivation of total transmission

c_i of Eq. (5) is formulated as in Eq. (A.1). Here, $N_Q \cdot m^{i-1}$ represents the number of queries stored at the i^{th} tier.

$$c_i = \widehat{cover}(N_Q \cdot m^{i-1}) \quad (A.1)$$

Next, Amt_Data_j of Eq. (5) is formulated as in Eq. (A.2).

$$\begin{aligned} Amt_Data_j &= (\text{the number of sensor nodes at} \\ &\quad \text{the } j^{th} \text{ tier}) \\ &\quad \cdot (\text{the size of a data element}) \\ &= f^{j-1} \cdot Size_{de} \end{aligned} \quad (A.2)$$

By substituting c_i and Amt_Data_j in Eq. (5) for Eq. (A.1) and Eq. (A.2), we can rewrite the formula for total transmission as follows.

$$\begin{aligned} total_transmission \\ &= \sum_{i=2}^h \sum_{j=i}^h (c_i \cdot Amt_Data_j) \end{aligned}$$

$$\begin{aligned}
&= \sum_{i=2}^h (c_i \cdot Amt_Data_i + \dots + c_i \cdot Amt_Data_h) \\
&= Amt_Data_2 \cdot \sum_{j=2}^2 c_j + \dots + Amt_Data_h \cdot \sum_{j=2}^h c_j \\
&= \sum_{i=2}^h (Amt_Data_i \cdot \sum_{j=2}^i (c_j)) \\
&= \sum_{i=2}^h (Size_{de} \cdot f^{i-1} \cdot \sum_{j=2}^i (\widehat{cover}(N_Q \cdot m^{j-1}))) \\
&= \sum_{i=2}^h (Size_{de} \cdot f^{i-1} \cdot \sum_{j=2}^i (-a \cdot m^{j-1} \cdot N_Q + b)) \\
&\text{where } a = \frac{s \cdot (1 - c)}{c - s}, \text{ and } b = 1 + a \quad (\text{A.3})
\end{aligned}$$

Derivation of total_storage

Amt_Mem_i of Eq. (6) is formulated as in Eq. (A.4).

$$\begin{aligned}
Amt_Mem_i &= (\text{the number of merged queries stored at} \\
&\quad \text{a node at the } i^{\text{th}} \text{ tier}) \\
&\quad \cdot (\text{the amount of memory used to store} \\
&\quad \text{one query}) \\
&= (N_Q \cdot m^{i-1}) \cdot (2 \cdot Size_{de}) \quad (\text{A.4})
\end{aligned}$$

By substituting Amt_Mem_i in Eq. (6) for Eq. (A.4), the formula for total_storage can be rewritten as follows.

$$\begin{aligned}
total_storage &= \sum_{i=2}^h (f^{i-1} \cdot Amt_Mem_i) \\
&= \sum_{i=2}^h (f^{i-1} \cdot N_Q \cdot m^{i-1} \cdot 2 \cdot Size_{de}) \quad (\text{A.5})
\end{aligned}$$



Jeong-Hoon Lee received his B.S. and M.S. degrees in computer engineering from Kyungpook National University, in 1995 and 1997, respectively. From 1997 to 2002, he was a senior research engineer at Korea Information and Communication Co., Ltd. In 2002, he was a research staff at the Advanced Information Technology Research Center (AITrc). He is currently doing his Ph.D. at Korea Advanced Institute of Science and Technology (KAIST). His research interests include peer-to-peer and sensor networks.

sensor networks.



Kyu-Young Whang graduated (Summa Cum Laude) from Seoul National University in 1973 and received the M.S. degrees from Korea Advanced Institute of Science and Technology (KAIST) in 1975, and Stanford University in 1982. He earned the Ph.D. degree from Stanford University in 1984. From 1983 to 1991, he was a Research Staff Member at the IBM T.J. Watson Research Center, Yorktown Heights, NY. In 1990, he joined KAIST, where he currently is a KAIST Distinguished Professor at the Department of Computer Science.

He was the program chair (Asia and Pacific Rim) for COOPIS'98, the program chair (Asia, Pacific, and Australia) for VLDB 2000, and a program co-chair of ICDE 2006. He was the general chair of VLDB 2006, PAKDD 2003, and DASFAA 2004. Dr. Whang served as the Coordinating Editor-in-Chief of the VLDB Journal. He was a trustee of the VLDB Endowment from 1998 to 2004 (reelected in 2010) and the steering committee chair of the DASFAA Conference from 2007 to 2009. He is an ACM Fellow, an IEEE Fellow, and a member of IFIP WG 2.6.



Hyo-Sang Lim received his B.S. degree in Computer Science from Yusei University, Seoul, Korea, in 1998, and M.S. and Ph.D. degrees in Computer Science from KAIST, Daejeon, Korea, in 1999 and 2007, respectively. Currently, he is working at Purdue University, USA as a post doctoral researcher. His research interests include data streams, sensor networks, spatial databases and web databases.



Byung Suk Lee is Associate Professor of Computer Science at the University of Vermont. His main research interests are database systems, data management, and query processing. He held several positions in industry and academia: previously at Gold Star Electric, Bell Communications Research, Datacom Global Communications, and University of St. Thomas, and currently at the University of Vermont. He was also a visiting professor at Dartmouth College and a participating guest at Lawrence Livermore National Laboratory.

He served on international conferences as a program committee member, a publicity chair, a special session organizer, and a workshop organizer, and also on the review panels of US federal funding agencies. He holds a B.S. degree from Seoul National University, M.S. from KAIST, and Ph.D. from Stanford University.



Jun-Seok Heo received the B.S. (1995) and M.S. (1997) degrees in Computer Science and Statistics from University of Seoul. He earned the Ph.D. (2009) degree in Computer Science from Korea Advanced Institute of Science and Technology (KAIST). From 1997 to 2002, he was a senior researcher at Daewoo Telecom Co., Ltd. and Mercury Co., where he participated in developing a distributed real-time main-memory DBMS used for switching systems. In 2002, he was a research staff at the Advanced Information Technology Research Center (AITrc).

Currently, he is working in KAIST as a post doctoral researcher. His research interests include information retrieval, sensor networks, geographic information systems, and telecommunication systems.