The Journal of Systems and Software 83 (2010) 2627–2641

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Effective processing of continuous group-by aggregate queries in sensor networks

Chun-Hee Lee [a], Chin-Wan Chung [a,*], Seok-Ju Chun [b]

[a] Department of Computer Science, Korea Advanced Institute of Science and Technology (KAIST), Daejeon 305-701, Republic of Korea
[b] Department of Computer Education, Seoul National University of Education, Seoul 137-742, Republic of Korea

ARTICLE INFO

Article history:
Received 16 April 2009
Received in revised form 18 June 2010
Accepted 18 August 2010
Available online 29 September 2010

Keywords:
Sensor network
Group-by aggregate query
Haar wavelet
Two-phase collection

ABSTRACT

Aggregate queries are one of the most important queries in sensor networks. Especially, group-by aggregate queries can be used in various sensor network applications such as tracking, monitoring, and event detection. However, most research has focused on aggregate queries without a group-by clause.

In this paper, we propose a framework, called the G-Framework, to effectively process continuous group-by aggregate queries in the environment where sensors are grouped by the geographical location. In the G-Framework, we can perform energy effective data aggregate processing and dissemination using two-dimensional Haar wavelets. Also, to process continuous group-by aggregate queries with a HAVING clause, we divide data collection into two phases. We send only non-filtered data in the first collection phase, and send data requested by the leader node in the second collection phase. Experimental results show that the G-Framework can process continuous group-by aggregate queries effectively in terms of energy consumption.

1. Introduction

Sensor networks consist of small sensors which have computing and communication facilities. With the advancement of sensor technology, sensors are becoming smaller and more powerful. Moreover, as the price of a sensor becomes low, we expect that a large number of sensors will be used in various sensor network applications.

For example, a volcanologist can use a sensor network to monitor a dangerous active volcanic area. Low-priced sensors can be scattered over the dangerous area from an airplane. Such sensors become a sensor network and monitor the volcano without humans' help. However, sensors have very limited resources (e.g., memory, computation, communication and energy). Among various resources, energy is one of the very important resources since the battery replacement is difficult or impossible in such environments. In sensor networks, since individual sensor readings are raw data, there are many applications using aggregate values. In many cases, the aggregate values of many regional areas are preferred to the aggregate value of the whole area since the aggregate value of the whole area does not provide the detailed information. That is, group-by aggregate queries are useful in sensor networks. Therefore, in this paper, we consider continuous group-by aggregate queries. Due to many shortcomings of the current technology, it is difficult to manage a large number of sensors. As one of the effective methods to deal with many sensors, we can use clustering in sensor networks (Heinzelman et al., 2002; Younis and Fahmy, 2004). Since sensor readings have spatial correlations, spatial clustering of sensors has many benefits. Therefore, we deal with group-by aggregate queries in the environment where sensors are grouped (clustered) by the geographical location. A group-by aggregate query may have a HAVING clause which is a predicate for the aggregation of the group. The queries we consider in this paper are shown in Fig. 1. However, we focus on the query in Fig. 2(a) since processing of queries in Fig. 1 can be extended from the processing of the query in Fig. 2(a). Also, the G-Framework can process local predicates in a straightforward method. Each node checks whether sensor readings satisfy local predicates and makes the bitmap. Then, the node sends only the satisfied data and the bitmap. Therefore, we will not mention local predicates in this paper for convenience of explanation.

Many papers proposed the processing of aggregate queries (Madden et al., 2002; Fan et al., 2002; Considine et al., 2004; Nath et al., 2004; Shrivastava et al., 2004; Deligiannakis et al., 2004; Sharaf et al., 2003, 2004). However, most of them do not consider group-by aggregate queries. Although some papers deal with processing group-by aggregate queries, they do not focus on processing group-by aggregate queries by the geographical location. In this paper, we focus on processing those queries. They can be used in many sensor networks applications such as tracking, monitoring, and event detection. To process them, we assume the following:

- Sensors are grouped according to the geographical location. See Fig. 2(b). A group consists of a leader node and member nodes. A leader node and member nodes are connected in one hop (the

* Corresponding author. Tel.: +82 42 350 3537; fax: +82 42 350 7737.
E-mail addresses: leechun@islab.kaist.ac.kr (C.-H. Lee), chungcw@kaist.edu (C.-W. Chung), chunsj@snue.ac.kr (S.-J. Chun).

0164-1212/$ – see front matter © 2010 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2010.08.049

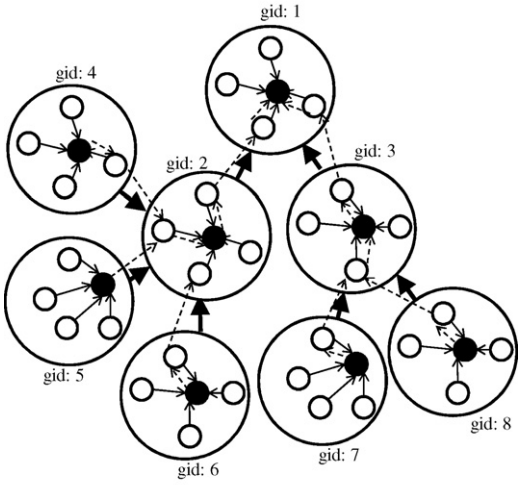
```
SELECT agg1(attr1), gid
FROM sensor
WHERE local predicates
GROUP BY gid
[HAVING agg2(attr2) op τ]
```

agg1, agg2: count, min, max, sum, average
attr1, attr2: any attribute in a sensor
op: >, <, ≥, ≤, =, ≠

Fig. 1. Query template.

```
SELECT sum(temperature), gid
FROM sensor
GROUP BY gid
HAVING sum(temperature) > τ
```

(a) Query



(b) Topology

Fig. 2. Query and topology.

solid lines in Fig. 2(b)). Although we assume the one hop connection, a larger group with multi-hop connections can be handled as discussed in Section 7.

- There is a tree routing topology among leader nodes (the dotted lines in Fig. 2(b)). The thick arrow between groups means the parent-child relationship. We can construct the tree routing topology for the leader nodes by flooding.
- Sensors are synchronized. To synchronize sensors in the G-Framework, we can use the synchronization approach in Ping (2003). In the approach, a master node is chosen as the time coordinator, and broadcasts the time synchronization message. A receiver node takes the message, measures the delay between the master node and itself and synchronizes the time. The approach in Ping (2003) is a lightweight approach (i.e., energy-efficient) and can be applied to multi-hop networks. After a certain time, synchronized sensors are unsynchronized due to various factors. Therefore, we should synchronize sensors periodically.

Fig. 2 shows the query and topology we consider. The query is to monitor the regions when their aggregate values are more than τ. A group is formed according to the geographical location and has a leader node and member nodes. The leader node collects data to compute the aggregation of the group. This query can be used in many sensor network applications. For example, consider building monitoring systems which automatically control the status of the building such as the room air temperature. If the room air temperature is more than a given threshold, we want to turn on the air conditioner. To monitor the status of each room, we install many sensors in the building. We can then group sensors according to the room. All sensors in the same room belong to the same group. The leader node of a room collects sensor readings from member nodes and sends the aggregate value to the base station. In building control systems, the base station receives the aggregate value of each room and controls the air conditioner using the aggregate value.

To effectively[1] process continuous group-by aggregate queries in sensor networks, we consider the following two factors.

- **Approximate processing**: A sensor gets sensor readings from the device. However, no matter how much the device is advanced, there are gaps between real values and sensor readings. Therefore, sensor readings have inevitable errors and small errors are allowed in such environments. Given an error threshold from a user, we will compute aggregate values of groups within the error threshold.
- **Delayed processing**: Continuous queries get results successively. In monitoring applications, a user does not need results immediately. Therefore, delayed results are allowed in such environments. We will compute aggregate values of groups with a delay.

Considering the above two factors, we propose a new framework, called the *G-Framework*, to process continuous group-by aggregate queries. In the G-Framework, we focus on reducing the communication cost since it is the primary factor of energy consumption. We use Haar wavelets to reduce the intra-group communication cost and inter-group communication cost in the G-Framework. Since Haar wavelets reduce data very effectively and are simple, they can be adapted well in sensor networks. To compute the aggregation of a group, sensor readings of member nodes are collected in the leader node. To reduce the intra-group communication cost (i.e., communication cost between the member node and the leader node), in the G-Framework, a member node collects sensor readings during a fixed period instead of sending a sensor reading immediately. Then, the member node compresses them using one-dimensional Haar wavelets and sends important wavelet coefficients to the leader node. The leader node receives wavelet coefficients from member nodes and computes the aggregation of the group.

The aggregate value of each group should be transmitted to the base station effectively. To do that, we use two-dimensional wavelets. The parent group receives the aggregation vectors from the child groups and compresses them using two-dimensional Haar wavelets. By using two-dimensional Haar wavelets, we can send the aggregation vectors to the base station effectively.

A group-by aggregate query may have a HAVING clause. In the G-Framework, we use two-phase collection to process a HAVING clause effectively. To perform two-phase collection, we set the filter condition $v_i \leq F_i$ (v_i is the sensor reading of node i and F_i is the filter value of node i) to each node i of a group according to the HAVING clause. In the first collection phase, a member node sends only sensor readings which are not valid for the filter condition. After receiving data in the first collection, the leader node determines sensor readings to send in the second collection phase.

1.1. Contributions

Our contributions are as follows:

[1] *Effectively* in this paper is used as the meaning of *effectively in terms of energy saving*.

- **Effective group-by aggregate query processing through two-dimensional Haar wavelets**: To process group-by aggregate queries in the environment where sensors are grouped by the geographical location, we use two-dimensional Haar wavelets as an approximate approach. By using two-dimensional Haar wavelets, we can aggregate data in a group and disseminate the aggregate value to the base station effectively. Also, we derive a mathematical formula which guarantees an error bound.
- **Effective HAVING clause processing through two-phase collection**: To process a HAVING clause effectively, we propose two-phase collection. In the two-phase collection, we install filters in all nodes of a group. Unnecessary transmissions are filtered in the first phase while necessary but filtered data is transmitted in the second phase. Since we divide data collection into two phases using filters, we can reduce the number of transmissions effectively.
- **Dummy data adjustment in Haar Wavelets**: One of the critical problems in Haar wavelets is that the performance of the compression can be bad if the number of data is not in a power of two. To adjust the number of data in a power of two, we can fill the remaining elements with any values (i.e., dummy data). Also, dummy data may be contained during the inter-group merging. The value of dummy data affects the compression quality. Therefore, we propose a method to determine the value of dummy data in order to improve the compression ratio under the same error bound.
- **Validation of the G-framework through experimental evaluations**: Through experimental evaluations, we validate that in the G-Framework, group-by aggregate queries are processed effectively in terms of energy consumption. Also, we show that the proposed dummy data adjustment algorithm improves the compression ratio of Haar wavelets.

1.2. Organization

The rest of the paper is organized as follows. In Section 2, we discuss the related work. In Section 3, we explain Haar wavelets as the preliminary. We describe the G-Framework to process continuous group-by aggregate queries in Sections 4, 5 and 6. Also, we discuss network issues of the G-Framework in Section 7. Finally, we show experimental results in Section 8 and conclude our work in Section 9.

2. Related work

Much work on managing aggregate queries in sensor networks has been reported (Madden et al., 2002; Trigoni et al., 2005; Fan et al., 2006, 2002; Trigoni et al., 2006; Considine et al., 2004; Nath et al., 2004; Shrivastava et al., 2004; Deligiannakis et al., 2004; Sharaf et al., 2003; Beaver et al., 2003; Sharaf et al., 2004; Zhang and Shatz, 2006). Madden et al. (2002) propose TAG (Tiny AGgregation) for in-network aggregate processing. They classify aggregates in sensor networks and define the structure of aggregates with a merging function f, an initializer i, and an evaluator e to compute aggregate queries in sensor networks. Approximate query processing in sensor networks is very useful due to the property of sensor networks. Therefore, many papers deal with approximate aggregate processing (Considine et al., 2004; Nath et al., 2004; Shrivastava et al., 2004; Deligiannakis et al., 2004; Sharaf et al., 2003, 2004). A method to process duplicate sensitive aggregate queries in a multi-path routing is proposed in Considine et al. (2004). Since data can be duplicate due to the multi-path routing, Considine et al. (2004) use duplicate-insensitive sketches and get approximate results. Nath et al. (2004) provide a general framework, *synopsis diffusion*, for the approximate aggregate processing in multi-path routing schemes. Shrivastava et al. (2004) deal with the processing of quantiles in sensor networks. To process the quantiles, they propose a new summary structure, q-digest (quantile digest). Also, they provide error bounds on quantile queries. Deligiannakis et al. (2004) propose an effective hierarchical in-network data aggregation technique. To reduce the amount of transmission, they utilize a bound filter $[L, H]$ in Olston et al. (2003). Each node i has its bound filter $[L_i, H_i]$. If the current sensor reading v in the node i is within the bound filter (i.e., $L_i \leq v \leq H_i$), the node does not send any messages to the parent node. Otherwise, the reading is sent to the parent node. In order to allocate the width ($W = H - L$) for the bound filter adaptively, they use a gain-based approach. By keeping simple statistics, W can be allocated adaptively.

As a class of aggregate queries, we can consider aggregate threshold queries which evaluate $f(\vec{v}) > r$, where $f : \mathbb{R}^d \rightarrow \mathbb{R}$ is an aggregate function and $\vec{v}$ is a global measurement vector. Sharfman et al. (2007) deal with this type of aggregate queries (i.e., aggregate threshold queries). To process the queries effectively, they use a geometric property which is related to the convex-hull. To reduce the communication cost, they check $f(\vec{v}) > r$ locally using the property.

Most approaches on aggregate processing do not consider group-by aggregate queries. Although TAG (Madden et al., 2002) considers group-by aggregate queries, the approach in TAG is straightforward. In TAG, if the aggregate value received from a child node is in the same group as some value in the parent node, the parent node simply combines the two values using the combining function.

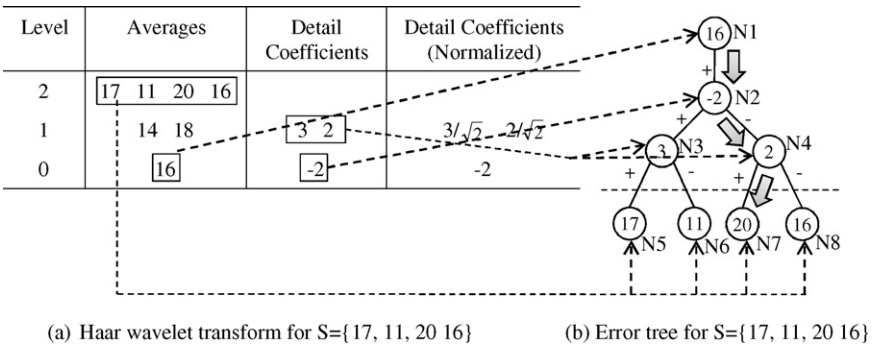
Sharaf et al. (Sharaf et al., 2004, 2003; Beaver et al., 2003) propose a group-aware construction of the routing tree and in-network aggregation by exploiting temporal correlations. They consider group-by aggregate queries. To construct the effective routing tree for group-by aggregate queries, they introduce a group-aware network configuration method which builds a routing tree in order for the nodes in the same group to be on the same path. Also, they use temporal suppression to reduce the communication cost between the child node and the parent node. Each node keeps the last partial aggregate results in the case of the non-leaf node or the last reported reading in the case of the leaf node. If new partial aggregate results or the current reported reading are within the given relative error tct ($(|V_{new} - V_{old}|)/V_{new} \leq tct$, where tct is the temporal coherency tolerance), they will be suppressed. However, Sharaf et al. do not consider a HAVING clause. Zhang and Shatz (2006) propose a method to create a routing topology for processing group-by aggregate queries in sensor networks, but they do not consider the processing of group-by aggregate queries.

3. Preliminary

In the G-Framework, we adopt Haar wavelets as a data compression tool. Since wavelets (Stollnitz et al., 1996) can reduce data very effectively, they have been used in many database areas (Matias et al., 1998; Gilbert et al., 2001; Vitter and Wang, 1999).

The one-dimensional Haar wavelet transform consists of one overall average and detail coefficients, which are called wavelet coefficients. The wavelet coefficients are computed by repeating two simple operations, averaging and differencing (Stollnitz et al., 1996). If the values in the pair are a and b, the averaging operation is $(a+b)/2$ and the differencing operation is $(a-b)/2$. The difference values become detail wavelet coefficients and the average values are used to get detail wavelet coefficients in the next level.

The detailed procedure for the Haar wavelet transformation is shown in Fig. 3(a). From the original data $S = \{17, 11, 20, 16\}$, the detail coefficients $\{3 = (17-11)/2, 2 = (20-16)/2\}$ and the average values $\{14 = (17+11)/2, 18 = (20+16)/2\}$ are computed.



(a) Haar wavelet transform for S={17, 11, 20 16} (b) Error tree for S={17, 11, 20 16}

Fig. 3. Haar wavelet transform and error tree for $S = \{17, 11, 20, 16\}$.

Again, from the average values {14, 18}, the detail coefficient $\{-2 = (14-18)/2\}$ and the average value $\{16 = (14+18)/2\}$ are computed. Therefore, wavelet coefficients are {16, −2, 3, 2}. Wavelet coefficients in a low level are more important than those in a high level. To normalize such wavelet coefficients, we divide wavelet coefficients in level l by $\sqrt{2^l}$. *Detail Coefficients (Normalized)* in Fig. 3(a) shows normalized wavelet coefficients in each level. {17, 11, 20, 16} is transformed into the normalized wavelet coefficients $\{16, -2, 3/\sqrt{2}, 2/\sqrt{2}\}$.

For data reduction, some coefficients should be selected. Wavelet thresholding is a way of selecting coefficients within the available space. It is well known that the largest normalized coefficients should be selected in order to minimize the L_2 error (Stollnitz et al., 1996). However, such thresholding does not provide the individual error. To solve this problem, a probabilistic wavelet thresholding scheme is proposed in Garofalakis and Gibbons (2002), which only provides the expected value. Also, Garofalakis and Kumar (2004) propose an optimal deterministic wavelet thresholding algorithm to minimize the maximum error. However, the algorithm proposed in Garofalakis and Kumar (2004) has large time complexity and space complexity. Therefore, practical wavelet thresholding algorithms for the maximum error are proposed in Karras and Mamoulis (2005). As a wavelet thresholding algorithm, we use GreedyAbs in Karras and Mamoulis (2005). However, any other wavelet thresholding algorithms for the maximum error can also be used in the G-Framework.

To understand the properties of Haar wavelet transform, we can build an error tree (Matias et al., 1998) with unnormalized wavelet coefficients. Fig. 3(b) shows the error tree for $S = \{17, 11, 20, 16\}$. The error tree consists of the root node and a binary tree. The overall average corresponds to the root node. In Fig. 3, since the overall average for $S = \{17, 11, 20, 16\}$ is 16, the root node N1 is denoted by 16. The detail coefficients in each level correspond to the binary tree and the original data corresponds to the leaves in the binary tree. In Fig. 3(b), detail coefficient $\{-2\}$ in Level 0 corresponds to N2. Detail coefficients {3, 2} in Level 1 correspond to N3 and N4, respectively. Finally, the original data {17, 11, 20, 16} corresponds to N5, N6, N7, and N8. We can observe that the value of a leaf node is constructed by traversing the path from the root node to the leaf node. While traversing the path, if the current node is the root node or we change the current node to its left child node, the value of the current node is added. Otherwise, the value of the current node is subtracted. For example, 20 is constructed by computing $+(16) - (-2) + (2) = 20$ as shown in the arrow of Fig. 3(b).

There are the standard construction and the non-standard construction for constructing the two-dimensional Haar wavelet transform. We mention only the standard construction since we use the standard construction in the G-Framework. Fig. 4 shows the process of the standard construction. In the standard construction, we first apply the one-dimensional Haar wavelet transform to each row of the original two-dimensional data. We then apply the one-dimensional Haar wavelet transform to each column of transformed two-dimensional data. In this way, we can get two-dimensional Haar wavelet coefficients. Since the one-dimensional Haar wavelet transform is applied to both rows and columns, the two-dimensional Haar wavelet transform reflects correlations for both rows and columns.

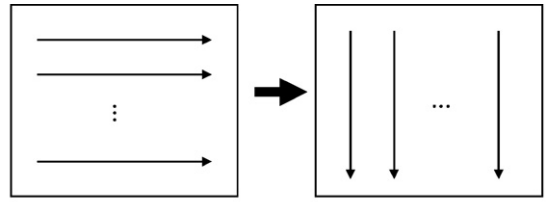


Fig. 4. Standard construction for two-dimensional Haar wavelet transform.

4. Framework to process group-by aggregate queries without a HAVING clause

We first describe how to process group-by aggregate queries without a HAVING clause in the G-Framework. The error threshold for the aggregation of a group is given by a user. If the user gives the maximum absolute error ϵ as an error threshold, we guarantee that $|r - \hat{r}| \leq \epsilon$, where r is the exact aggregate value and $\hat{r}$ is the approximate value computed from the G-Framework.

To process group-by aggregate queries without a HAVING clause, we perform two steps, intra-group aggregation and inter-group merging, in the G-Framework. In the intra-group aggregation, the leader node of each group computes a aggregation vector of the group, one for each epoch. Each node i in a group has the assigned error threshold ϵ_i and it is guaranteed that the errors for the computed aggregations are within $\epsilon_1 + \epsilon_2 + \cdots + \epsilon_k$, if k is the number of nodes in the group. In the inter-group merging, the aggregation vectors from groups are merged with the error threshold ϵ_{inter} and sent to the base station. To send the vectors to the base station, we construct the tree routing topology, which is independent of the topology in Fig. 2(b), from the base station by flooding. In Sections 4 and 5, we assume that error thresholds ϵ_1, $\epsilon_2, \ldots, \epsilon_k$, and ϵ_{inter} are assigned such that $\epsilon_1 + \epsilon_2 + \cdots + \epsilon_k + \epsilon_{inter} = \epsilon$. We will explain how to assign the error thresholds ($\epsilon_1, \epsilon_2, \ldots, \epsilon_k$, and ϵ_{inter}) effectively in Section 6. The algorithms in a member node and a leader node are summarized in Figs. 5 and 6, respectively.

4.1. Intra-group aggregation

In order to compute the aggregation of a group, all member nodes in the group send sensor readings to the leader node. However, if they send data to the leader node every round, much

```
memberNode()
begin
1: nVector:= collect n sensor readings;
2: nVector' := compress nVectorusing one-dimensional wavelets and
     apply wavelet thresholding;
3: send nVector' to leaderNode;
end
```

Fig. 5. memberNode() algorithm to process group-by aggregate queries without a HAVING clause.

```
leaderNode()
begin
 1: // Intra-Group Aggregation
 2: nVector:= collect n sensor readings;
 3: nVector' := compress nVectorusing one-dimensional wavelets;
 4: coefVector: = nVector';
 5: while the current time is within the allocated time according to
      the intra-group aggregation
 6:{
 7:   recvPacket := receive the packet from child nodes;
 8:   merge recvPacket.waveletCoefficient into coefVector;
 9:}
10: coefVector' := apply wavelet thresholdingto coefVector;
11: send coefVector' to parentLeaderNode;

12: // Inter-Group Merging
13: if(isGroupLeaderNode())
14:{
15:   while the current time is the allocated time according to
        the inter-group merging
16:   {
17:     recvPacket := receive the packet from child  groups;
18:     arrange recvPacket.coefVectorin the format of the matrix;
19:   }
20:   finalData := compress each column in the arranged matrix using
        one-dimensional wavelets and apply wavelet thresholding;
21:   send finalData to basestation;
22:}
end
```

Fig. 6. leaderNode() algorithm to process group-by aggregate queries without a HAVING clause.

energy will be consumed. Therefore, we use a compression scheme to reduce the communication cost. In this section, we consider only one group, which has $k-1$ member nodes ($Node_1$, $Node_2$, ..., $Node_{k-1}$) and the leader node $Node_k$. Also, $Node_i$ has the allocated error threshold ϵ_i. We will explain how to allocate the error threshold in Section 6.

In the G-Framework, the aggregation proceeds as follows:

Sensing and collecting sensor readings: Each node $Node_i$ including a leader node collects n sensor readings ($[v_{i1}\, v_{i2} \cdots v_{in}]$, v_{ij} is the jth sensor reading at $Node_i$) locally.

Compressing sensor readings: Each node $Node_i$ transforms n sensor readings into n wavelet coefficients using one-dimensional Haar wavelets. To reduce the communication cost, we drop unimportant wavelet coefficients, which is called wavelet thresholding. As a wavelet thresholding algorithm, we use GreedyAbs in Karras and Mamoulis (2005) which is efficient in both time and space complexity. Using GreedyAbs, we eliminate unimportant wavelet coefficients such that $|v_{ij} - v'_{ij}| \leq \epsilon_i$ for all j, where v'_{ij} is the approximate data constructed from wavelet coefficients after wavelet thresholding. At this time, the leader node $Node_k$ does not apply wavelet thresholding since the aggregation is computed at the leader node.

Sending wavelet coefficients to the leader node: To avoid a collision when member nodes send wavelet coefficients to the leader node, we use TDMA (Time Division Multiple Access). A member node communicates with the leader node at its allocated time slot.

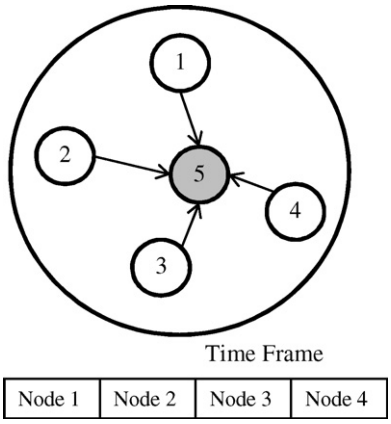


Fig. 7. Intra-group aggregation.

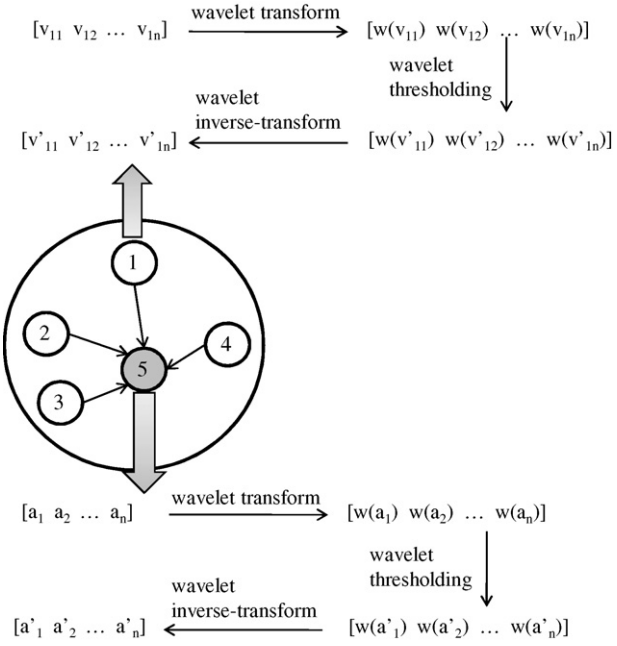


Fig. 8. Notation in intra-group aggregation.

As an example, Fig. 7 shows the group with 5 nodes. Each member node has the allocated time slot. In that time slot, it sends only important wavelet coefficients to the leader node and in other time slots, it is in the sleep mode.

Computing and compressing the aggregation of a group: Refer to the notation in Fig. 8. The leader node $Node_k$ receives only important wavelet coefficients $([w(v'_{i1})w(v'_{i1}) \cdots w(v'_{in})])$[2] from each member node $Node_i$. Most $w(v'_{ij})$ for $j \in \{1, 2, \ldots, n\}$ are zero. Only non-zero values are sent while zero values are not sent, but zero values are set at the leader node. To compute the aggregation, the leader node decompresses (inverse-transforms) sensor readings received from $Node_i$ and merges them. The jth aggregation a_j

[2] $w(x_{ij})$ is the jth Haar wavelet coefficient for $[x_{i1}\, x_{i2} \cdots x_{in}]$.

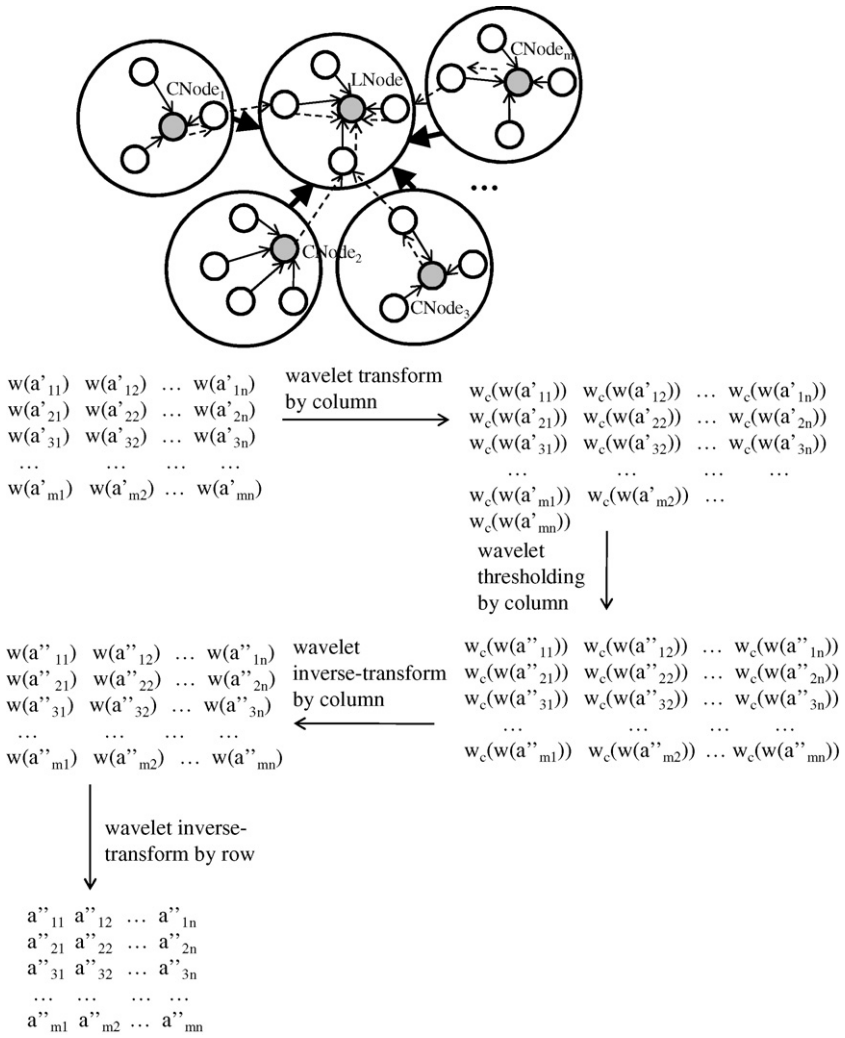


Fig. 9. Notation in inter-group merging.

is computed as follows:

$$a_j = v_{kj} + \sum_{i=1}^{k-1} v'_{ij}$$

Then, the leader node transforms $[a_1\, a_2 \cdots a_n]$ into wavelet coefficients $[w(a_1)\, w(a_2) \cdots w(a_n)]$ using one-dimensional Haar wavelets. However, since Haar wavelets have a linear property, we do not need to decompress wavelet coefficients in processing group-by aggregate queries without a HAVING clause. Instead, we merge wavelet coefficients directly.

$$w(a_j) = w(v_{kj}) + \sum_{i=1}^{k-1} w(v'_{ij})$$

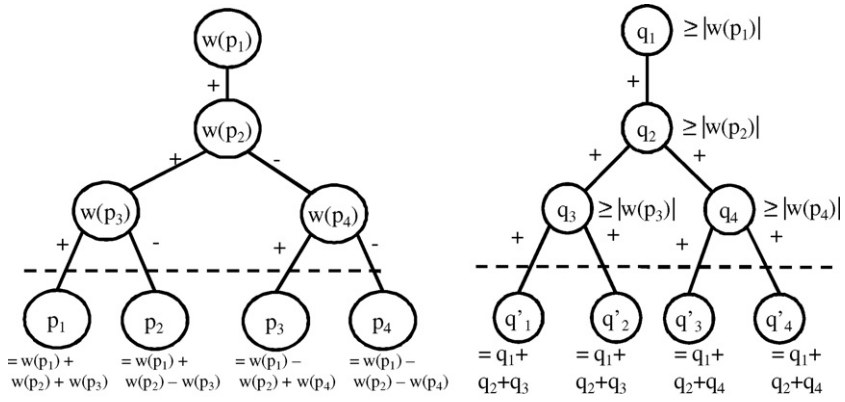
Finally, we apply wavelet thresholding to wavelet coefficients $[w(a_1)\, w(a_2) \cdots w(a_n)]$ using the allocated error ϵ_k in the leader node. Wavelet coefficients after wavelet thresholding are $[w(a'_1)\, w(a'_2) \cdots w(a'_n)]$. We can easily prove that $|r_i - a'_i| \le \sum_{j=1}^{k} \epsilon_j$ (r_i is the i th exact aggregate value) by the formula $|A+B| \le |A| + |B|$.

4.2. Inter-group merging

After the intra-group aggregation, the leader node in a group has the compressed aggregation vector $[w(a'_1)\, w(a'_2) \cdots w(a'_n)]$. The leader node can send the vector to the base station without further processing. However, sensor readings in sensor networks have strong spatial correlations. If we exploit spatial correlations, we can reduce the communication cost even more.

If two nodes are close, they generally have strong spatial correlations. Therefore, we merge aggregation vectors in the near groups. We can choose the near groups as the child groups under the same parent group. We will explain how to perform the inter-group merging focusing on one parent group since other parent groups are processed in the same way and the aggregation vectors after the inter-group merging are directly sent to the base station without further processing. We denote the leader node of the parent group by *LNode* and the leader node of the i th child group by $CNode_i$. We assume that there are m child groups under the parent group. Refer to the notation in Fig. 9. In Fig. 9, $w_c(x_{ij})$ is the i th Haar wavelet coefficient for $\begin{bmatrix} x_{1j} \\ x_{2j} \\ \cdots \\ x_{mj} \end{bmatrix}$.

The leader node of the parent group (*LNode*) receives the compressed aggregation vector $[w(a'_{i1})\, w(a'_{i2}) \cdots w(a'_{in})]$ from the leader node of the i th child group ($CNode_i$). In this section, we extend the notation a'_i to a'_{ij}. a'_{ij} is the jth aggregate value received from the i th child group. In the G-Framework, the inter-group merging is performed as follows:



(a) The error tree for P (b) The tree corresponding to (a)

Fig. 10. Error tree in Lemma 1.

Sending the aggregation vector of a group to the leader node of the parent group: After the intra-group aggregation, $CNode_i$ has the aggregation vector as one-dimensional Haar wavelet coefficients. $CNode_i$ sends one-dimensional Haar wavelet coefficients $[w(a'_{i1})w(a'_{i2})\cdots w(a'_{in})]$ to *LNode*. **Arranging the aggregation vectors from child groups**: *LNode* receives the aggregation vectors (one-dimensional Haar wavelet coefficients) from m child groups and arranges them in the form of a matrix as shown in the upper-left part of Fig. 9. The aggregation vector in LNode is not processed in LNode, but in its parent group. That is, the vector in LNode is handled the same as the vector in a child group. Therefore, the above matrix does not contain the aggregation vector in LNode. **Compressing the matrix using two-dimensional Haar wavelets**: Since each row was transformed, *LNode* transforms each column using one-dimensional Haar wavelets. It is called two-dimensional Haar wavelet transform. We can exploit temporal and spatial correlations using two-dimensional Haar wavelets. **Dropping the unimportant wavelet coefficients from the matrix**: Since sensor networks have limited computation and memory, we propose a simple solution for dropping two-dimensional Haar wavelet coefficients. If the error threshold ϵ'_j for the jth column is given, we apply wavelet thresholding in each column independently such that $|w(a'_{ij}) - w(a''_{ij})| \le \epsilon'_j$ for all i, where a''_{ij} is the approximate value constructed from the matrix after applying wavelet thresholding in each column.

Let the given error threshold for the inter-group merging be ϵ_{inter}. We derive Theorem 1 to guarantee $|a'_{ij} - a''_{ij}| \le \epsilon_{inter}$ for all i, j from $|w(a'_{ij}) - w(a''_{ij})| \le \epsilon'_j$ for all i, j. We define $F_S(i)$ where $S = \{s_1, s_2, \ldots, s_n\}$ as below. $F_S(1)$ is the maximum value among values for paths from the root to the leaf nodes illustrated in Fig. 10(b). The value for a path is computed by adding the values of the nodes on the path from the root node to the leaf node in the error tree for S. With $F_S(i)$, we derive Lemma 1.

$$F_S(i) = \begin{cases} s_1 + F_S(2) \text{ for } i = 1 \\ s_i + \max\{F_S(2i-1), F_S(2i)\} \text{ for } 1 < i \le n \\ 0 \text{ for } i > n \end{cases}$$

Lemma 1. *Let $P = \{p_1, p_2, \ldots, p_n\}$ be original data and $P' = \{w(p_1), w(p_2), \ldots, w(p_n)\}$ be the one-dimensional Haar wavelet coefficients for P. For $Q = \{q_1, q_2, \ldots, q_n\}$ such that $q_i \ge |w(p_i)|$ for all $i \in \{1, 2, \ldots, n\}$, $\max_i |p_i| \le F_{\{q_1, q_2, \ldots, q_n\}}(1)$.*

Proof. Fig. 10(a) shows the error tree for P. For convenience, we consider the case of $n = 4$ in Fig. 10. We build the tree of Fig. 10(b) corresponding to the error tree of Fig. 10(a) with $Q = \{q_1, q_2, \ldots, q_n\}$. Then, we construct the leaf nodes $\{q'_1, q'_2, \ldots, q'_n\}$ in Fig. 10(b) by the path from the root to the leaf node similarly to the error tree. However, in the tree of Fig. 10(b), when we construct the value of a leaf node from the path, we always add the value of the current node whether we change the current node to its left child node or right child node. Then, by the property of the trees in Fig. 10, $|p_i| \le q'_i$ for $i \in \{1, 2, \cdots, n\}$. Therefore, $max_i|p_i| \le max_i\{q'_i\}$. By the definition, $\max_i\{q'_i\}$ is $F_{\{q_1,\ldots,q_n\}}(1)$. Therefore, $\max_i|p_i| \le F_{\{q_1,\ldots,q_n\}}(1)$. □

Theorem 1.

$$\max_{i,j}|a'_{ij} - a''_{ij}| \le F_{\{\epsilon'_1, \epsilon'_2, \ldots, \epsilon'_n\}}(1)$$

Proof. To prove $\max_{i,j}|a'_{ij} - a''_{ij}| \le F_{\{\epsilon'_1, \epsilon'_2, \ldots, \epsilon'_n\}}(1)$, assume that i is fixed.

By the definition of ϵ'_j,

$$|w(a'_{i1}) - w(a''_{i,1})| \le \epsilon'_1$$
$$|w(a'_{i2}) - w(a''_{i2})| \le \epsilon'_2$$
$$\cdots$$
$$|w(a'_{in}) - w(a''_{in})| \le \epsilon'_n$$

Let $p'_j = w(a'_{ij}) - w(a''_{ij})$ and $q_j = \epsilon'_j$. Since the Haar wavelet transform has a linear property, $p'_j = w(a'_{ij}) - w(a''_{ij}) = w(a'_{ij} - a''_{ij})$.

If we let $P = \{p_1, \ldots, p_n\}$ be data reconstructed from $P' = \{p'_1, \ldots, p'_n\}$ by wavelet inverse-transform, $p_j = a'_{ij} - a''_{ij}$.

```
memberNode()
begin
 1: nVector:= collect n sensor readings;
 2: nonFilteredNVector:= remove filtered readings from nVector;
 3: bitmap := make the bitmap for the recollection
 4: nonFilteredNVector' := compress nonFilteredNVector using
      one-dimensional wavelets and apply wavelet thresholding;
 5: sendPacket := make the packet with nonFilteredNVector' and bitmap;
 6: send sendPacket to leaderNode;

 7: if(recollectionPacket is received from the leader node)
 8:{
 9:  recollectionVector:= extract sensor readings according to
        recollectionPacket.RBitmap;
10:  recollectionVector' := compress recollectionVectorusing
        one-dimensional wavelets and apply wavelet thresholding;
11:}
12: send recollectionVector' to leaderNode;
end
```

Fig. 11. memberNode() algorithm to process group-by aggregate queries with a HAVING clause.

```
leaderNode()
begin
 1: // Intra-Group Aggregation
 2: nVector:= collect n sensor readings;
 3: bitmap := make the bitmap for the recollection;
 4: dataVector: = nVector;
 5: RBitmap := bitmap;
 6: while the current time is within the allocated time according to the first collection phase
 7:{
 8:   recvPacket := receive the packet from child nodes;
 9:   childVector:= decompress recvPacket.waveletCoefficient using one-dimensional wavelets;
10:   merge childVectorinto dataVector;
11:   merge bitmap into RBitmap;
12:}

13: if (some element in RBitmap is not 0)
14:{
15:   broadcast the recollection message (RBitmap) to member nodes;
16:   while the current time is within the allocated time according to the second collection phase
17:  {
18:    recvPacket := receive the packet from child nodes;
19:    childVector:= decompress recvPacket.waveletCoefficient using one-dimensional wavelets;
20:    merge childVectorinto dataVector;
21:  }
22:}

23: validDataVector:= remove invalid aggregate values for the HAVING clause from dataVector;
24: leaderBitmap := make the bitmap for checking which aggregate values are valid for
       the HAVING clause;
25: validDataVector' = compress validDataVectorusing one-dimensional wavelets and
       apply wavelet thresholding;
26: sendPacket := make the packet with validDataVector' and leaderBitmap;
27: send sendPacket to parentLeaderNode;

28: // Inter-Group Merging
29: if(isGroupLeaderNode())
30:{
31:   while the current time is within the allocated time according to the inter-group merging
32:  {
33:     recvPacket := receive the packet from child groups;
34:     childVector:= decompress recvPacket.waveletCoefficient using one-dimensional wavelets;
35:     dataMatrix := arrange childVectorin the format of the matrix;
36:  }
37:   finalData := compress dataMatrix using two-dimensional wavelets and
         apply wavelet thresholding;
38:   send finalData to basestation;
39:}
end
```

Fig. 12. leaderNode() algorithm to process group-by aggregate queries with a HAVING clause.

By Lemma 1, $\max_j|p_j| \le F_{\{\epsilon'_1,\ldots,\epsilon'_n\}}(1)$. That is, $\max_{i,j}|a'_{ij} - a''_{ij}| \le F_{\{\epsilon'_1,\epsilon'_2,\ldots,\epsilon'_n\}}(1)$

For any $i \in \{1, 2, \ldots, m\}$, the above equation is valid. □

In this paper, we allocate the same error threshold x in each column. By Theorem 1, $\max_{i,j}|a'_{ij} - a''_{ij}| \le F_{\{x,x,\ldots,x\}}(1) = x + x \times \log_2 n = \epsilon_{inter}$, where ϵ_{inter} is the given the error threshold for the inter-group merging. To guarantee that $|a'_{ij} - a''_{ij}| \le \epsilon_{inter}$, we set x to $(\epsilon_{inter})/(1 + \log_2 n)$. Then, we are sure that | the exact aggregate value − the approximate aggregate value computed from the G-Framework | $\le \sum_{i=1}^{k} \epsilon_i + \epsilon_{inter}$.

5. Framework to process group-by aggregate queries with a HAVING clause

The basic framework to process group-by aggregate queries with a HAVING clause is the same as that in Section 4. However, if the aggregate value does not satisfy a HAVING clause, the leader node of a group does not need to send the aggregate value to the base station. Considering this point, we propose a two-phase collection to process a HAVING clause effectively. The algorithms in a member node and a leader node are summarized in Figs. 11 and 12, respectively.

5.1. Intra-group aggregation

For a two-phase collection, a member node has a filter condition. Suppose that a HAVING clause is $sum(attr) > \tau$. For an easy explanation, we consider only the jth aggregation (i.e., j is fixed). For evaluating the HAVING clause $sum(attr) > \tau$, we should check whether $\sum_{i=1}^{k} v_{ij} > \tau$, where v_{ij} is the jth value of $attr$ in $Node_i$. To check it locally, we set the filter condition $v_{ij} \le F_i$ to $Node_i$ (F_i is the filter value of $Node_i$). We allocate F_i such that $\sum_{i=1}^{k} F_i = \tau$. $Node_i$ then checks the filter condition $v_{ij} \le F_i$.

Suppose that the filter condition in all nodes is satisfied ($v_{ij} \le F_i$ for $i \in \{1, 2, \ldots, k\}$). Then, $\sum_{i=1}^{k} v_{ij} \le \sum_{i=1}^{k} F_i = \tau$. That is, the HAVING

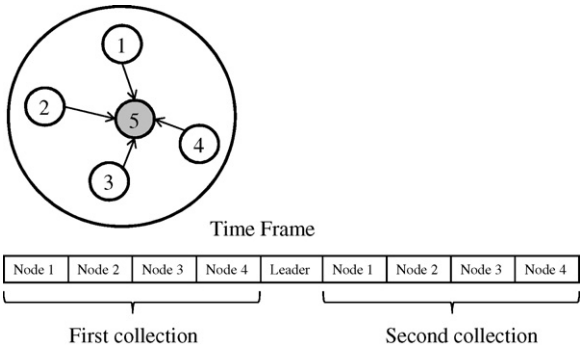


Fig. 13. Intra-group aggregation.

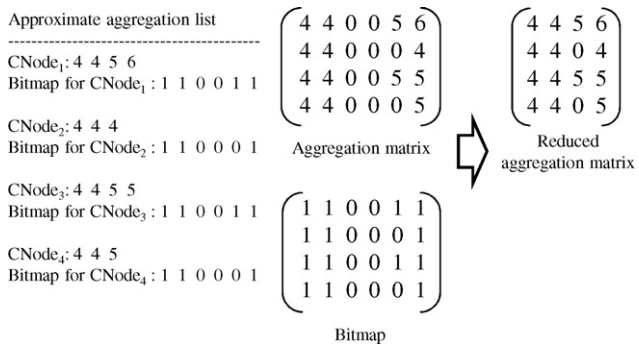


Fig. 14. Reduced aggregation matrix.

clause is not satisfied. In this case, a node does not need to send the sensor reading to the leader node. However, if the filter condition in any node is not satisfied, we should check the HAVING clause in the leader node. The two-phase collection based on this is performed as follows:

Sensing and collecting sensor readings: Like the processing of group-by aggregate queries without a HAVING clause, a node collects its sensor readings for compression.

The first collection phase: compressing only sensor readings which are not valid for the filter condition and sending them: In the first collection phase, each member node $Node_i$ sends only sensor readings which are not valid for the filter condition. It extracts non-filtered readings ($v_{ij} > F_i$) and transforms them into one-dimensional Haar wavelet coefficients. To know which wavelet coefficients are extracted, the member node makes a bitmap. After applying wavelet thresholding with the error threshold ϵ_i, $Node_i$ sends important wavelet coefficients and the bitmap to the leader node in the first time slot corresponding to $Node_i$ as shown in Fig. 13.

Broadcasting the recollection message to member nodes: A member node compresses only non-filtered sensor readings after extracting them. Therefore, the positions of non-filtered sensor readings in the n original sensor readings may be different according to the node. To aggregate such sensor readings, the leader node inverse-transforms wavelet coefficients from a member node into approximate sensor readings, while they were not inverse-transformed in Section 4. Then, the leader node arranges the approximate sensor readings using the bitmap and aggregates them. Finally, we make the bitmap for the recollection (*RBitmap*). If $v_{kj} \leq F_k$ for the leader node and all member nodes did not send the jth sensor reading, we set the jth element in *RBitmap* to 0. Otherwise, we set the jth element in *RBitmap* to 1. The leader node broadcasts *RBitmap* for the second collection phase. If all elements in *RBitmap* are 0, the leader node does not send any messages.

The second collection phase: compressing sensor readings in the recollection messages and sending them: If a member node receives the recollection message, it performs the second collection. In the second collection phase, the member node $Node_i$ extracts only sensor readings such that

- The element in *RBitmap* corresponding to the sensor reading is 1.
- The sensor reading was not sent to the leader node in the first collection phase.

Then, $Node_i$ transforms the extracted sensor readings into wavelet coefficients and applies wavelet thresholding with the error threshold ϵ_i. The important wavelet coefficients are sent to the leader node in the second time slot corresponding to $Node_i$.

Computing and compressing the aggregation of a group: The leader node inverse-transforms wavelet coefficients received in the second collection into approximate sensor readings. Then, the leader node aggregates them into the previous partial aggregate values. Then, the leader node determines whether the aggregate value is valid for the HAVING clause and makes the bitmap for checking which aggregate values are filtered. Finally, the leader node transforms only the aggregation vector which is valid for the HAVING clause into wavelet coefficients. After applying wavelet thresholding with the error threshold ϵ_k, the wavelet coefficients and the bitmap are sent to the leader node of the parent group.

5.2. Inter-group merging

Arranging aggregation vectors from child groups: The leader node of the parent group (*LNode*) receives wavelet coefficients from child groups and decompresses them. Then, *LNode* arranges the approximate aggregation vectors using the bitmaps and makes the aggregation matrix and the bitmap matrix as shown in Fig. 14. If the element in the bitmap is 0, we fill the element in the aggregation matrix with 0.

If all the elements in the column of the bitmap matrix are 0, we can remove the column. It is called the reduced aggregation matrix. For example, suppose that the parent group receives four aggregation vectors from 4 child groups as shown in Fig. 14. Since all elements in the 3rd column and 4th column of the bitmap matrix are 0, we reduce the 4×6 aggregation matrix to the 4×4 reduced aggregation matrix. **Compressing the reduced aggregation matrix using two-dimensional Haar wavelets**: *LNode* transforms the reduced aggregation matrix using two-dimensional Haar wavelets. It first transforms each row in the matrix using one-dimensional Haar wavelets and then transforms each column in the transformed matrix. However, in the reduced matrix, there may exist dummy data such that the corresponding elements in the bitmap matrix are 0s (e.g., $(2,3)$ and $(4,3)$ elements in the reduced aggregation matrix of Fig. 14). If we fill such an element with another value instead of 0, the compression quality is generally improved. Therefore, we extend a Haar wavelet transform algorithm. The details are explained in Section 5.2.1.

Dropping the unimportant wavelet coefficients from the matrix: To drop the unimportant Haar wavelet coefficients, we apply wavelet thresholding for each column like the previous section.

5.2.1. Dummy data adjustment

Since Haar wavelets are performed with the unit of a pair, the number of data should be in a power of two. If the number of data is not in a power of two, we fill the remaining elements with 0. For example, since the number of data in Fig. 16 is 6, two zeros are added in order to adjust the number of data in a power of two. However, if we fill the remaining elements with any other values instead of 0, we may improve the compression ratio. For example, in Fig. 16, if we fill the remaining elements with $7((6+8)/2=7)$,

```
input
  data: original data
  n: the number of data
  flag: bitmap for dummy data
output
  coef: wavelet coefficients

begin
 1: for (i:=n; i>=2; i:= i/2)
 2: {
 3:   // allocate temp and newFlag
 4:   float temp[i/2];
 5:   boolean newFlag[i/2];

 6:   // initialize newFlag
 7:   for (s:=0; s<i/2; s:=s+1) newFlag[s] = true;

 8:   for (s:=0; s<i; s:=s+2)
 9:   {
10:     if (flag[s] == true && flag[s+1] == false) data[s+1] := data[s];
11:     if (flag[s] == false && flag[s+1] == true) data[s] := data[s+1];
12:     if (flag[s] == false && flag[s+1] == false) newFlag[s/2] := false;

13:     // averaging and differencing
14:     temp[s/2] := (data[s] + data[s+1])/2;

15:     if (flag[s] == false && flag[s+1] == false) coef[i/2+s/2] = 0;
16:     else coef[i/2+s/2] := (data[s] - data[s+1])/2;
17:   }
18:   data := temp;
19:   flag = newFlag;
20:   if (i/2 == 1) coef[0] := temp[0]; // overall average
21: }
end
```

Fig. 15. Dummy data adjustment algorithm.

wavelet coefficients are (5.75 −1.25 −0.5 0 0 0 -1 0) while wavelet coefficients are (4 0.5 −0.5 3.5 0 0 −1 0) in the case of filling them with zeros. The number of zeros in (5.75 −1.25 −0.5 0 0 0 −1 0) is more than that in (4 0.5 −0.5 3.5 0 0 −1 0). Through this example, we know that we can improve Haar wavelets by filling the remaining elements with appropriate values if the number of data is not in a power of two. We can consider the remaining elements as dummy data since the remaining elements do not affect the original data during compression and decompression.

Similarly, consider the 2nd row in the reduced aggregation matrix of Fig. 14. Since the bitmap corresponding to the 3rd element in the 2nd row is 0, we can consider it as dummy data and any value can be filled at the position of dummy data without a problem.

We propose a method to determine the value of dummy data in order to improve the compression ratio. The idea is to fill the dummy data with a value similar to the adjacent elements. The dummy data adjustment algorithm is shown in Fig. 15. As the input for the algorithm, the bitmap flag which represents whether the element is dummy data or not is given. The underlined statements are the extended statements to the Haar wavelet transform algorithm. Since the averaging and differencing are performed for a pair, if there is only one dummy data in the pair, we fill the dummy data

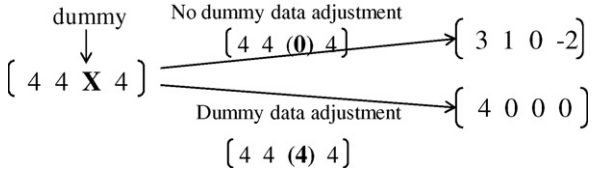


Fig. 16. Example 1 for the dummy data adjustment.

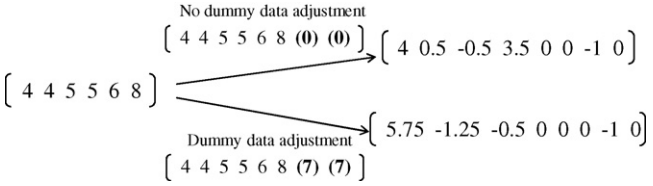


Fig. 17. Example 2 for the dummy data adjustment.

with the non-dummy element in the pair (Lines 10–11). However, if both elements in the pair are dummy, we fill the dummy data in the next level of the wavelet transform (Line 12).

For example, in Fig. 17, the 3rd element is dummy and the adjacent element (the 4th element) is 4. Therefore, we fill it with 4 and transform (4 4 4 4) instead of (4 4 0 4). We can know that wavelet coefficients after applying the dummy data adjustment algorithm have more zeros than before applying the algorithm. It means that the number of the dropped coefficients after applying the dummy data adjustment algorithm is more than before applying the algorithm, and the compression ratio is improved in the case of applying the algorithm.

6. Dynamic error and filter allocation

Since the environments in sensor networks are dynamic, we propose a method to allocate errors and filters dynamically. For dynamic allocation, we apply the gain-based approach in Deligiannakis et al. (2004) to the G-Framework. Deligiannakis et al. (2004) propose a hierarchical in-network aggregation technique using bound filters. Given the width of the bound W, they guarantee $L \le v \le H$, where v is the aggregate value computed from the hierarchical in-network aggregation and $W = H - L$. To guarantee the aggregate value with the bound $[L, H]$, each node $Node_i$ has the filter in the form $[L_i, H_i]$ with the width $W_i = H_i - L_i$. Note that $\Sigma W_i = W$. If the current sensor reading lies in $[L_i, H_i]$, the node suppresses it. Otherwise, the node sends the data to the parent node. The parent node has the filter in the form $[L_i, H_i]$ for the child node. If the child node does not send the data, the parent node predicts the sensor reading of the child node as $(H_i + L_i)/2$. If the parent node receives the sensor reading v_i of the child node, the parent node changes the filter of the child node into $[v_i - W_i/2, v_i + W_i/2]$. To allocate the width dynamically, Deligiannakis et al. (2004) reduce the width W_i in each node periodically by *shrinkFactor* ($W_i := W_i \times shrinkFactor$) and then, allocate the remaining width $(1 - shrinkFactor) \times W$ to nodes based on the potential gain. The potential gain $Gain_i$ of node i is defined with C_{shrink} and C_{expand}.

$$Gain_i = \delta G = C_{shrink} - C_{expand}$$

C_{shrink} is the number of messages over the bound if the width of the bound is shrunk ($W_i := shrinkFactor \times W_i$) and C_{expand} is the number of messages over the bound if the width of the bound is expanded ($W_i := W_i + dW$, where dW is an increment factor). Then, the remaining width is allocated in proportion to $Gain_i$ of node i.

6.1. Dynamic error allocation

Given an error threshold ϵ, we guarantee that | exact aggregate value − approximate aggregate value | $\le \epsilon$. In the G-Framework, since we drop wavelet coefficients according to the error threshold, we should allocate the error threshold effectively. This is similar to the allocation of the width in Deligiannakis et al. (2004). We should allocate $\epsilon_1, \epsilon_2, \ldots, \epsilon_k$, and ϵ_{inter} with the given error threshold ϵ such that $\epsilon = \epsilon_1 + \epsilon_2 + \cdots + \epsilon_k + \epsilon_{inter}$, where ϵ_{inter} is $F_{\{\epsilon'_1, \epsilon'_2, \ldots, \epsilon'_n\}}(1)$ in Theorem 1. Initially, each error threshold is allocated uniformly such that $\epsilon_1 = \epsilon_2 = \cdots = \epsilon_k = \epsilon_{inter} = \epsilon/(k+1)$. For periodical dynamic

error updates, we shrink errors by *shrinkFactor* and allocate the remaining error threshold using the gain.

If $g_1, g_2, \ldots, g_k$, and $g_0 = g_{inter}$ are the gains for $\epsilon_1, \epsilon_2, \ldots, \epsilon_k$, and $\epsilon_0 = \epsilon_{inter}$, respectively, we update ϵ_i based on g_i.

$$\epsilon_i := \epsilon_i \times shrinkFactor + \epsilon \times (1 - shrinkFactor) \times \frac{g_i}{\sum_{j=0}^{k} g_j}$$

We let C_{shrink} be the sum of the number of retained coefficients in the node i if the error threshold is $shrinkFactor \times \epsilon_i$ and C_{expand} be the sum of the number of the retained coefficients if the error threshold is $\epsilon_i + dW$. The gain g_i of node i is computed as follows:

$$g_i = \frac{weight_i \times (C_{shrink} - C_{expand})}{(\epsilon_i + dW) - (\epsilon_i \times shrinkFactor)}$$

The difference between the error threshold $\epsilon_i + dW$ for C_{expand} and the error threshold $\epsilon_i \times shrinkFactor$ for C_{shrink} varies depending on the node. Therefore, to normalize $(C_{shrink} - C_{expand})$, we divide $(C_{shrink} - C_{expand})$ by $((\epsilon_i + dW) - (\epsilon_i \times shrinkFactor))$. Also, we multiply the gain with the weight due to the different communication overhead. We define *weight* as follows:

- $weight_0$ = the number of hops between the leader node of the parent group and the base station.
- For $i \in \{1, 2, \ldots, k-1\}$, $weight_i = 1$
- $weight_k$ = the number of hops between the leader node of the child group and the leader node of the parent group

Also, we let $dW = ((1 - shrinkFactor) \times \epsilon)/(k+1)$ since the number of nodes is $k+1$ and the error threshold to be allocated is $(1 - shrinkFactor) \times \epsilon$. After updating the error threshold, C_{shrink} and C_{expand} are initialized.

6.2. Dynamic filter allocation

Given τ in a HAVING clause, we should allocate τ such that $\sum_{i=1}^{k} F_i = \tau$. For an effective filter value allocation, we compute the gain for the filter similar as in Section 6.1. Initially, each filter value F_i is allocated uniformly such that $F_1 = F_2 = \cdots = F_k = \tau/k$. For a dynamic filter value allocation, we periodically shrink filter values by *shrinkFactorFilter*. Then, filter values are allocated according to the gain.

If $g'_1, g'_2, \ldots, and g'_k$ are the gains for $F_1, F_2, \ldots, and F_k$, respectively, we update F_i based on g'_i.

$$F_i := F_i \times shrinkFactorFilter + \tau \times (1 - shrinkFactorFilter) \times \frac{g'_i}{\sum_{j=1}^{k} g'_j}$$

The gain of node i is computed as follows:

$$g'_i = \frac{(C'_{shrink} - C'_{expand})}{(F_i + dW') - (F_i \times shrinkFactorFilter)}$$

We define C'_{shrink} as the sum of the number of data which is not valid for the filter condition $v_i \leq F_i \times shrinkFactorFilter$ and C'_{expand} as the sum of the number of data which is not valid for the filter condition $v_i \leq F_i + dW'$. We let $dW' = ((1 - shrinkFactorFilter) \times \tau)/k$ since the number of nodes is k and the error to be allocated is $(1 - shrinkFactorFactor) \times \tau$. After updating the filter value, C'_{shrink} and C'_{expand} are initialized.

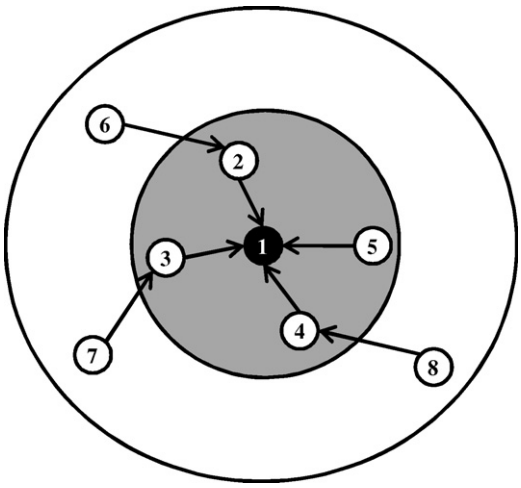


Fig. 18. Example for a group which is connected in one or two hops.

7. Discussion

Although we propose an effective framework to process group-by aggregate queries in this paper, we should consider some network issues. First, in sensor network applications, the leader node and member nodes in a group can be connected in multi-hops. In the G-Framework, although we assumed that the leader node and member nodes are connected in one hop, we can process group-by aggregate queries in the case that a group is connected in multi-hops. If the leader node and member nodes are in two or more hops, we can extend the internal node to perform processing. The internal node aggregates its own data and the data received from child nodes. Thus, any member node connected to the leader node in one hop or multi-hops sends one compressed aggregation vector instead of multiple vectors. Because of that, the G-Framework will not show a bad performance in spite of multi-hop transmissions. Fig. 18 shows a group which is connected in one or two hops. Nodes 6, 7 and 8 compress their own data using one-dimensional wavelets. Then, Nodes 6, 7 and 8 send their compressed data to Nodes 2, 3, and 4, respectively. Nodes 2, 3 and 4 aggregate their own data and the data received from Nodes 6, 7 and 8, respectively and send the aggregate data to Node 1.

Second, the leader node consumes more energy than member nodes. Therefore, we need to rotate the leader node in a group. We can rotate the leader node as follows: Periodically, the leader node collects the amount of the remaining energy for each member node. Then, the current leader node assigns the member node with the largest energy or itself to the next leader node. When a new leader node is assigned, the topology structure in a group can be constructed by flooding which is used in TAG (Madden et al., 2002). In the future, we plan to extend our paper in two points. First, we will propose effective intra-group aggregation in the case of multi-hop groups. Although we can process the intra-group aggregation in the case of multi-hop groups as mentioned above, it will not be effective as the maximum hop count increases. Therefore, we plan to extend the intra-group aggregation in the case of multi-hop groups. Second, we will adapt and implement our approach in a real environment. Then, we will discuss the lessons learned from the implementation.

8. Experiments

In order to validate the effectiveness of the G-Framework, we conduct experimental evaluations.

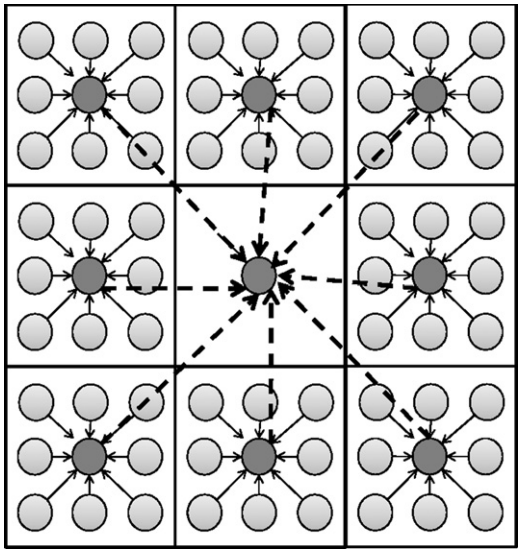


Fig. 19. Topology.

8.1. Experimental environment

To evaluate the performance of the G-Framework, we implement the G-Framework on our own simulation environment. As the comparison system, we implement the method in Deligiannakis et al. (2004) denoted by the *Hierarchical* on our own simulation environment and adapt it for group-by aggregation queries. However, in the Hierarchical, we do not extend to check a HAVING clause in the leader node of a group but process the clause at the base station since we do not know whether the dropped aggregate value is due to the suppression or due to the HAVING clause. As a topology for experimental evaluations, we use the 9×9 grid topology in Fig. 19. In the G-Framework, query processing is performed with the unit of a group. Therefore, the experimentation is not affected on the total network size. Through the small network 9×9 grid, we can validate the effectiveness of the G-Framework. One group consists of 9 nodes as a 3×3 grid. The center node in a 3×3 grid (i.e., the dark gray node in Fig. 19) is the leader node of the group. Also, since the inter-group merging is achieved only in a parent group, we consider a flat topology in which there is only a parent-child group relationship. The center group in the 9×9 grid topology is a parent group and other groups are child groups. Also, since data in the parent group is processed in its parent group, we do not consider sensor readings in the nodes of the parent group.

Since the major factor in consuming energy is the communication, we use the amount of transmission as a performance measure. Also, the hop count is considered as a distance measure in the simulation environment. We set the number of hops from a member node to its leader node to 1, the number of hops from $CNode_i$ to *LNode* to 3 and the number of hops from *LNode* to the base station to 20. Since the Hierarchical does not need the processing in the parent group, the aggregation value in the leader node of a group is directly sent to the base station. Therefore, we set the number of hops from $CNode_i$ to the base station to 23. As the header information, we simply use the source address and destination address. The size of source address is 4 bytes and the size of destination address is 4 bytes.

Basically, communications are based on TDMA. Each node has various allocated slots and tasks corresponding to the slots. We can adapt more effective protocols that have been studied in network areas. However, it is out of scope of this paper. We focus on data management issues rather than the communication among nodes. The simulation environments are summarized in Fig. 20.

We generate synthetic data based on real data. As real data, we use the 10,000 wind speed readings (Earth climate and weather, in press) which are collected at one place every minute. We assume that the wind blows from the east to the west, the wind speed gets weaker as it goes to the west, and the real data is measured in the right side of the 9×9 grid topology. Therefore, we generate sensor readings in nodes with the following formula.

$$v'_{xy}(t) = v(t) + \frac{a}{r} + b\delta,$$

$v'_{xy}(t)$ is the wind speed at t time at (x, y), $v(t)$ is the real wind speed at t time, a and b are constants, δ is a random value in [0, 1), and r is the distance on the x-coordinate between the location where the real data is measured and the (x, y).

For convenience, we assume that a packet has a simple header information which consists of a source address and a destination address. In the G-Framework, we send wavelet coefficients in a node as one packet. Sending data as one packet consumes less energy than multiple packets due to the header information. Therefore, we can have the additional benefit in the communication cost. To show the effectiveness of the G-Framework without the additional benefit, we measure the total transmission cost in both the case of including the header information and the case of not including the header information. In this section, we tag "_H" after the name if we include the header information in measuring the total transmission cost and "_NH" if we do not include it.

8.2. Experimental results

We conduct experimental evaluations with two types of queries, a group-by aggregate query without a HAVING clause and a group-by aggregate query with a HAVING clause, as shown in Fig. 21. We show experimental results for the query of Fig. 21(a) in Section 8.2.1 and those for the query of Fig. 21(b) in Section 8.2.2.

Parameter	Setting value
Toplogy	9 x 9 grid
Distance from a member node to its leader node	1 hop
Distance from $CNode_i$ to LNode	3 hops
Distance from LNode to the base station	20 hops
Header Information	<Source address, Destination address> The size of source address is 4bytes. The size of destination address is 4bytes.
Protocol	TDMA(Time Division Multiple Access)

Fig. 20. Parameters.

```
SELECT sum(temperature), gid
FROM sensor
GROUP BY gid
```

(a) group-by aggregate query without a HAVING clause

```
SELECT sum(temperature), gid
FROM sensor
GROUP BY gid
HAVING sum(temperature) > τ
```

(b) group-by aggregate query with a HAVING clause

Fig. 21. Queries for experimental evaluations.

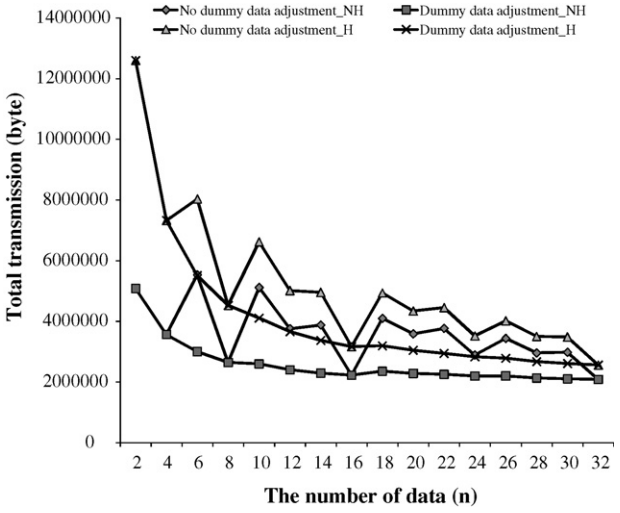


Fig. 22. Experiment for dummy data adjustment algorithm for group-by aggregate queries without a HAVING clause.

8.2.1. Group-by aggregate queries without a HAVING clause

We first show the effectiveness of the dummy data adjustment algorithm in Fig. 22.[3] The error bound is set to 10 in the experiment of Fig. 22. Both the G-Framework with the dummy data adjustment and the G-Framework without the dummy data adjustment show the tendency that the amount of transmission decreases as n (the number of data collected at one node to be processed together) increases. If n is large, we have more possibility to reduce data. Therefore, as n increases, we can reduce data more effectively.

However, the performance of the G-Framework without the dummy data adjustment is not smooth and shows a good performance only when n is a power of two. This is because the Haar wavelet transform is performed in the unit of a power of two. The performance of the G-Framework with the dummy data adjustment is always better than or equal to that of the G-Framework without the dummy data adjustment. Also, the amount of transmission in the G-Framework with the dummy data adjustment goes down softly since the dummy data is filled with a proper value. The result for the case of including the header information and that for the case of not including the header information are similar.

Fig. 23 shows the amount of transmission according to the error threshold when n is 16. The x-axis of Figs. 23 and 25, as a scale, represents the absolute error in the upper part as well as the relative error (absolute error/maximum value $\times$ 100) in the lower part. The G-Framework_H and the G-Framework_NH have better performance than the Hierarchical_H and the Hierarchical_NH, respectively. In the G-Framework and the Hierarchical, the performance becomes good as the error threshold increases. The gap between the G-Framework_H and the Hierarchical_H is big compared to the gap between the G-Framework_NH and the

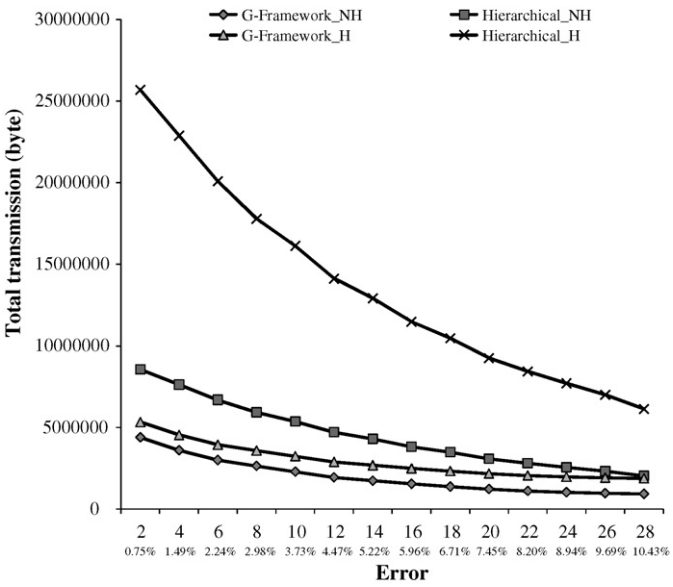


Fig. 23. Total transmission according to the error threshold for group-by aggregate queries without a HAVING clause.

Hierarchical_NH. This is because, in the G-Framework, we have the additional benefit for sending many coefficients as one packet.

8.2.2. Group-by aggregate queries with a HAVING clause

Fig. 24 shows the effectiveness of the dummy data adjustment algorithm. The error bound is set to 10 in the experiment of Fig. 24. In group-by aggregate queries with a HAVING clause, we can improve the compression ratio using the dummy data adjustment algorithm. Like the previous section, the performance of the G-Framework with the dummy data adjustment is better than that without the dummy data adjustment and changes softly. In group-by aggregate queries with a HAVING clause, data in a member node is sent to the leader node in only the case that the filter condition is satisfied. Therefore, although n increases, the amount of transmission does not always decrease. We use the G-Framework with the dummy data adjustment in the following experiments.

Fig. 25 shows the experimental results according to the error threshold when n is 16 and τ is 80. In the G-Framework and the Hierarchical, the amount of transmission decreases as the error threshold increases as we expect. The G-Framework shows better

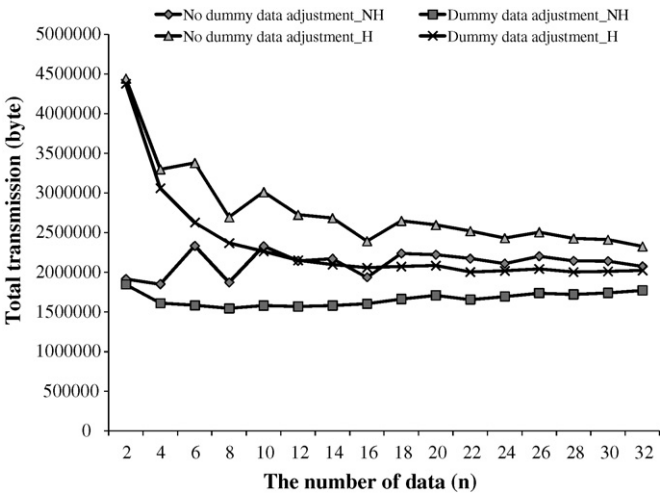


Fig. 24. Experiment for dummy data adjustment algorithm for group-by aggregate queries with a HAVING clause.

[3] We do not perform error and filter updates in experiments of Figs. 22 and 24 in order to show the effectiveness of the dummy data adjustment algorithm.

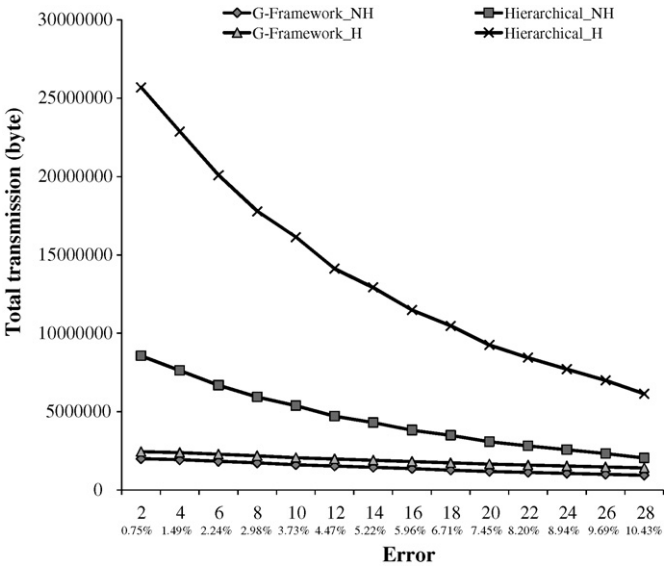


Fig. 25. Total transmission according to the error threshold for group-by aggregate queries with a HAVING clause.

performance than the Hierarchical. While the gap between the G-Framework_H and the G-Framework_NH is small, the gap between the Hierarchical_H and the Hierarchical_NH is big like Fig. 23. Also, since we use a two-phase collection to process group-by aggregate queries with a HAVING clause, the performance gap between the G-Framework and the Hierarchical in Fig. 25 is bigger than that in Fig. 23.

Finally, we conduct experiments on according to the filter value in Fig. 26 when n is 16 and ϵ (error threshold) is 10. The x-axis of Fig. 26 represents the filter value τ in the upper part as well as the ratio N_1/N_2 (N_1 is the number of aggregate values which satisfies the HAVING clause and N_2 is the total number of aggregate values) in the lower part. The G-Framework shows better performance than the Hierarchical. In the Hierarchical, since a HAVING clause can not be processed as in-network processing, the filter value does not have an effect on the Hierarchical. Therefore, the performance of the Hierarchical approach is constant, and the G-Framework generally becomes good as the filter value increases.

Consequently, through the experimental evaluations, we show that the G-Framework effectively processes group-by aggregate queries without a HAVING clause and with a HAVING clause. Also, using the dummy data adjustment, we can improve the compression ratio of Haar wavelets.

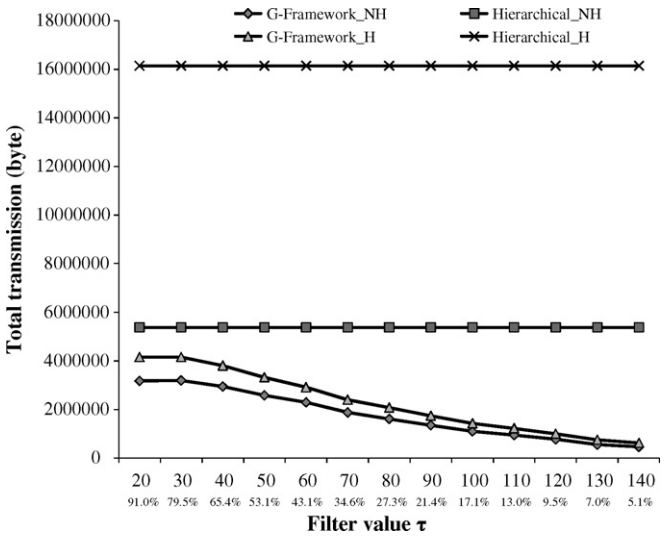


Fig. 26. Total transmission according to the filter value.

9. Conclusion

Continuous group-by aggregate queries can be used in many sensor networks applications. To process group-by aggregate queries in sensor networks, we propose the G-Framework. In the G-Framework, using two-dimensional Haar wavelets, we reduce the communication cost in the intra-group aggregation as well as in the inter-group merging. Also, in the G-Framework, group-by aggregate queries with a HAVING clause can be processed effectively in terms of energy consumption. Since sensor readings in sensor networks are dynamic, the dynamic property in sensor networks is considered in the G-Framework. Finally, we validate the effectiveness of the G-Framework by experimental evaluations.

Acknowledgment

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MEST) (No. 2010-0000863).

References

Beaver, J., Sharaf, M.A., Labrinidis, A., Chrysanthis, P.K., 2003. Location-aware routing for data aggregation in sensor networks. In: Geo Sensor Networks Workshop.
Considine, J., Li, F., Kollios, G., Byers, J.W., 2004. Approximate aggregation techniques for sensor databases. In: IEEE International Conference on Data Engineering (ICDE), pp. 449–460.
Deligiannakis, A., Kotidis, Y., Roussopoulos, N., 2004. Hierarchical in-network data aggregation with quality guarantees. In: International Conference on Extending Database Technology (EDBT), pp. 658–675.
Earth Climate and Weather. http://www-k12.atmos.washington.edu/k12/grayskies.
Fan, K.-W., Liu, S., Sinha, P., 2002. Supporting aggregate queries over ad-hoc wireless sensor networks. In: IEEE Workshop on Mobile Computing Systems and Applications (WMCSA), pp. 49–58.
Fan, K.-W., Liu, S., Sinha, P., 2006. Scalable data aggregation for dynamic events in sensor networks. In: ACM Conference on Embedded Networked Sensor Systems (SenSys), pp. 181–194.
Garofalakis, M.N., Gibbons, P.B., 2002. Wavelet synopses with error guarantees. In: ACM SIGMOD International Conference on Management of Data, pp. 476–487.
Garofalakis, M.N., Kumar, A., 2004. Deterministic wavelet thresholding for maximum-error metrics. In: ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS), pp. 166–176.
Gilbert, A.C., Kotidis, Y., Muthukrishnan, S., Strauss, M., 2001. Surfing wavelets on streams: one-pass summaries for approximate aggregate queries. In: International Conference on Very Large Data Bases (VLDB), pp. 79–88.
Heinzelman, W.B., Chandrakasan, A.P., Balakrishnan, H., 2002. An application-specific protocol architecture for wireless microsensor networks. IEEE Transactions on Wireless Communications 1 (4), 660–670.
Karras, P., Mamoulis, N., 2005. One-pass wavelet synopses for maximum-error metrics. In: International Conference on Very Large Data Bases (VLDB), pp. 421–432.
Madden, S., Franklin, M.J., Hellerstein, J.M., Hong, W., 2002. Tag: a tiny aggregation service for ad-hoc sensor networks. In: USENIX Symposium on Operating Systems Design and Implementation (OSDI), pp. 131–146.
Matias, Y., Vitter, J.S., Wang, M., 1998. Wavelet-based histograms for selectivity estimation. In: ACM SIGMOD International Conference on Management of Data, pp. 448–459.
Nath, S., Gibbons, P.B., Seshan, S., Anderson, Z.R., 2004. Synopsis diffusion for robust aggregation in sensor networks. In: ACM Conference on Embedded Networked Sensor Systems (SenSys), pp. 250–262.
Olston, C., Jiang, J., Widom, J., 2003. Adaptive filters for continuous queries over distributed data streams. In: ACM SIGMOD International Conference on Management of Data, pp. 563–574.
Ping, S., 2003. Delay measurement time synchronization for wireless sensor networks. In: Intel Research, IRB-TR-03-013.
Sharaf, M.A., Beaver, J., Labrinidis, A., Chrysanthis, P.K., 2004. Balancing energy efficiency and quality of aggregate data in sensor networks. VLDB Journal 13 (4), 384–403.
Sharaf, M.A., Beaver, J., Labrinidis, R., Chrysanthis, P.K., 2003. Tina: a scheme for temporal coherency-aware in-network aggregation. In: ACM International Workshop on Data Engineering for Wireless and Mobile Access (MobiDE), pp. 69–76.
Sharfman, I., Schuster, A., Keren, D., 2007. Aggregate threshold queries in sensor networks. In: IEEE International Parallel and Distributed Processing Symposium (IPDPS), pp. 1–10.

Shrivastava, N., Buragohain, C., Agrawal, D., Suri, S., 2004. Medians and beyond: new aggregation techniques for sensor networks. In: ACM Conference on Embedded Networked Sensor Systems (SenSys), pp. 239–249.

Stollnitz, E.J., DeRose, T.D., Salesin, D.H., 1996. Wavelets for Computer Graphics. Morgan Kaufmann.

Trigoni, N., Guitton, A., Skordylis, A., 2006. Routing and processing multiple aggregate queries in sensor networks. In: ACM Conference on Embedded Networked Sensor Systems (SenSys), pp. 391–392.

Trigoni, N., Yao, Y., Demers, A.J., Gehrke, J., Rajaraman, R., 2005. Multi-query optimization for sensor networks. In: IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS), pp. 307–321.

Vitter, J.S., Wang, M., 1999. Approximate computation of multidimensional aggregates of sparse data using wavelets. In: ACM SIGMOD International Conference on Management of Data, pp. 193–204.

Younis, O., Fahmy, S., 2004. Distributed clustering in ad-hoc sensor networks: a hybrid, energy-efficient approach. In: IEEE International Conference on Computer Communications (INFOCOM), pp. 629–640.

Zhang, Z., Shatz, S.M., 2006. A technique for power-aware query-informed routing in support of long-duration queries for sensor networks. In: IEEE International Conference on Networking, Sensing and Control (ICNSC), pp. 48–53.