Hierarchical In-Network Data Aggregation with Quality Guarantees

Antonios Deligiannakis[1]*, Yannis Kotidis[2], and Nick Roussopoulos[1]

[1] University of Maryland, College Park MD 20742, USA,
{adeli,nick}@cs.umd.edu
[2] AT&T Labs-Research, Florham Park NJ 07932, USA,
kotidis@research.att.com

Abstract. Earlier work has demonstrated the effectiveness of in-network data aggregation in order to minimize the amount of messages exchanged during continuous queries in large sensor networks. The key idea is to build an aggregation tree, in which parent nodes aggregate the values received from their children. Nevertheless, for large sensor networks with severe energy constraints the reduction obtained through the aggregation tree might not be sufficient. In this paper we extend prior work on in-network data aggregation to support approximate evaluation of queries to further reduce the number of exchanged messages among the nodes and extend the longevity of the network. A key ingredient to our framework is the notion of the residual mode of operation that is used to eliminate messages from sibling nodes when their cumulative change is small. We introduce a new algorithm, based on potential gains, which adaptively redistributes the error thresholds to those nodes that benefit the most and tries to minimize the total number of transmitted messages in the network. Our experiments demonstrate that our techniques significantly outperform previous approaches and reduce the network traffic by exploiting the super-imposed tree hierarchy.

1 Introduction

Technological advances in recent years have made feasible the deployment of hundreds or thousands of sensor nodes in an ad-hoc fashion, that are able to coordinate and perform a variety of monitoring applications ranging from measurements of meteorological data (like temperature, pressure, humidity), noise levels, chemicals etc. to complex military vehicle surveillance and tracking applications. Independently of the desired functionality of the sensors, all the above applications share several similar characteristics. First of all, processing is often driven by designated nodes that monitor the behavior of either the entire, or parts of the network. This monitoring is typically performed by issuing queries, which are propagated through the network, over the data collected by the sensor nodes. The output of the queries is then collected by the monitoring node(s) for further processing. While typical database queries are executed once, queries in monitoring applications are long-running and executed over a specified period, or until explicitly being terminated. These types of queries are known as *continuous queries* [3,14].

* Work partially done while author was visiting AT&T Labs-Research. Author also supported by the DoD-Army Research Office under Awards DAAD19-01-2-0011 and DAAD19-01-1-0494.

E. Bertino et al. (Eds.): EDBT 2004, LNCS 2992, pp. 658–675, 2004.

Another common characteristic of sensor node applications revolves around the severe energy and bandwidth constraints that are met in such networks. In many applications sensor nodes are powered by batteries, and replacing them is not only very expensive but often impossible (for example, sensors in a disaster area). In such cases, energy-aware protocols involving the operation of the nodes need to be designed to ensure the longevity of the network. This is the focus of the work in [9,10], where energy-based query optimization is performed. The bandwidth constraints arise from the wireless nature of the communication among the nodes, the short-ranges of their radio transmitters and the high density of network nodes in some areas.

Recent work [8,9,15] has focused on reducing the amount of transmitted data by performing in-network data aggregation. The main idea is to build an aggregation tree, which the results will follow. Non-leaf nodes of that tree aggregate the values of their children before transmitting the aggregate result to their parents. At each epoch, ideally, a parent node coalesces all partial aggregates from its child nodes and transmits upwards a single partial aggregate for the whole subtree.

All the above techniques try to limit the number of transmitted data while always providing accurate answers to posed queries. However, there are many instances where the application is willing to tolerate a specified error, in order to reduce the bandwidth consumption and increase the lifetime of the network. In [11], Olston et al. study the problem of error-tolerant applications, where the users register continuous queries along with strict precision constraints at a central *stream processor*. The stream processor then dynamically distributes the error budget to the remote data sources by installing filters on them that necessitate the transmission of a data value from each source only when the source's observed value deviates from its previously transmitted value by more that a threshold specified by the filter.

The algorithms in [11] cannot be directly applied to monitoring applications over sensor networks. While the nodes in sensor networks form an aggregation tree where messages are aggregated and, therefore, the number of transmitted messages depends on the tree topology, [11] assumes a flat setup of the remote data sources, where the cost of transmitting a message from each source is independent to what happens in the other data sources. Moreover, as we will show in this paper, the algorithms in [11] may exhibit several undesirable characteristics for sensor networks, such as:

- The existence of a few volatile data sources will make the algorithms of [11] distribute most of the available budget to these nodes, without any significant benefit, and at the expense of all the other sensor nodes.
- The error distribution assumes a worst-case behavior of the sensor nodes. If any node exceeds its specified threshold, then its data needs to be propagated to the monitoring node. However, there might be many cases when changes from different data sources effectively cancel-out each other. When this happens frequently, our algorithms should exploit this fact and, therefore, prevent unnecessary messages from being propagated all the way to the root node of the aggregation tree.

In this paper we develop new techniques for data dissemination in sensor networks when the monitoring application is willing to tolerate a specified error threshold. Our techniques operate by considering the potential benefit of increasing the error threshold at a sensor node, which is equivalent to the amount of messages that we expect to save

by allocating more resources to the node. The result of using this gain-based approach is a robust algorithm that is able to identify volatile data sources and eliminate them from consideration. Moreover, we introduce the *residual mode of operation*, during which a parent node may eliminate messages from its children nodes in the aggregation tree, when the cumulative change from these sensor nodes is small. Finally, unlike the algorithms in [11], our algorithms operate with only local knowledge, where each node simply considers statistics from its children nodes in the aggregation tree. This allows for more flexibility in designing adaptive algorithms, and is a more realistic assumption for sensors nodes with very limited capabilities [9].
Our contributions are summarized as follows:

1. We present a detailed analysis of the current protocols for data dissemination in sensor networks in the case of error-tolerant applications along with their shortcomings.
2. We introduce the notion of the *residual* mode of operation. When the cumulative change in the observed quantities of multiple sensor nodes is small, this operation mode helps filter out messages close to the sensors, and prevents these messages from being propagated all the way to the root of the aggregation tree. We also extend previous algorithms to make use of the residual mode and explore their performance.
3. We introduce the notion of the *potential gain* of a node or an entire subtree and employ it as an indicator of the benefit of increasing the error thresholds in some nodes of the subtree. We then present an adaptive algorithm that dynamically determines how to rearrange the error thresholds in the aggregation tree using simple, local statistics on the potential gains of the nodes. Similarly to [11], the sensor nodes in our techniques periodically shrink their error thresholds to create an error "budget" that can be re-distributed amongst them. This re-distribution of the error is necessary to account for changes in the behavior of each sensor node. Unlike [11], where nodes are treated independently, our algorithm takes into account the tree hierarchy and the resulting interactions among the nodes.
4. We present an extensive experimental analysis of our algorithms in comparison to previous techniques. Our experiments demonstrate that, for the same maximum error threshold of the application, our techniques have a profound effect on reducing the number of messages exchanged in the network and outperform previous approaches, up to a factor of 7 in some cases.

The rest of the paper is organized as follows. Section 2 presents related work. In Sect. 3 we describe the algorithms presented in [11] and provide a high level description of our framework. In Sect. 4 we discuss the shortcomings of previous techniques when applied to sensor networks. Section 5 presents our extensions and algorithms for dynamically adjusting the error thresholds of the sensor nodes. Section 6 contains our experiments, while Sect. 7 contains concluding remarks.

2 Related Work

The development of powerful and inexpensive sensors in recent years has spurred a flurry of research in the area of sensor networks, with particular emphasis in the topics of network self-configuration [2], data discovery [6,7] and in-network query processing [8,

9,15]. For monitoring queries that aggregate the observed values from a group of sensor nodes, [8] suggested the construction of a greedy aggregation tree that seeks to maximize the number of aggregated messages and minimize the amount of the transmitted data. To accomplice this, nodes may delay sending replies to a posed query in anticipation of replies from other queried nodes. A similar approach is followed in the TAG [9], TinyDB [10] and Cougar [15] systems. In [5], a framework for compensating for packet loss and node failures during query evaluation is proposed. The work in [9] also addressed issues such as query dissemination, sensor synchronization to reduce the amount of time a sensor is active and therefore increase its expected lifetime, and also techniques for optimizations based on characteristics of the used aggregate function. Similar issues are addressed in [10], but the emphasis is on reducing the power consumption by determining appropriate sampling rates for each data source. The above work complements our in many cases, but our optimizations methods are driven by the error bounds of the application at hand.

The work in [11] addressed applications that tolerate a specified error threshold and presented a novel, dynamic algorithm for minimizing the number of transmitted messages. While our work shares a similar motivation with the work in [11], our methods apply over a hierarchical topology, such as the ones that are typically met in continuous queries over sensor networks. Similarly, earlier work in distributed constraint checking [1,13] cannot be directly applied in our setting, because of the different communication model and the limited resources at the sensors. The work of [12] provides quality guarantees during in-network aggregation, like our framework, but this is achieved through a uniform allocation strategy and does not make use of the residual mode of operation that we introduce in this paper. The evaluation of probabilistic queries over imprecise data was studied in [4]. Extending this work to hierarchical topologies, such as the ones studied in our paper, is an open research topic.

3 Basics

We first describe Olston's framework and algorithms [11] for error-tolerant monitoring applications in flat, non-hierarchical, topologies, and then provide a high-level description of our framework. The notation that we will use in the description of our algorithms is presented in Table 1. A short description of the characteristics of sensor nodes, and sensor networks in general can be found in the full version of this paper.

3.1 Olston's Framework for Error Tolerant Applications

Consider a node $Root$, which initiates a continuous query over the values observed by a set of data sources. This continuous query aggregates values observed by the data sources, and produces a single aggregate result. For each defined query, a maximum error threshold, or equivalently a precision constraint E_Global that the application is willing to tolerate is specified. The algorithm will install filters at each queried data source, that will help limit the number of transmitted messages from the data source. The selection process for the filters enforces that at any moment t after the installation of the query to the data sources, the aggregate value reported at node $Root$ will lie within

the specified error threshold from the true aggregate value (ignoring network delays, or lost messages).

Initially, a filter F_i is installed in every data source S_i, such that the initial error guarantees are not violated. Each filter F_i is an interval of real values $[L_i, H_i]$ of width $W_i = H_i - L_i$, such that any source S_i whose current observed value $Current_i$ lies outside its filter F_i will need to transmit its current value to the $Root$ node and then re-center its filter around this transmitted value, by setting $L_i = Current_i - W_i/2$ and $H_i = Current_i + W_i/2$. However, if $Current_i$ lies within the interval specified by the filter F_i, then this value does not need to be transmitted.

In order for the algorithm to be able to adapt to changes in the characteristics of the data sources, the widths W_i of the filters are periodically adjusted. Every Upd time units, Upd being the *adjustment period*, each filter shrinks its width by a *shrink percentage* ($shrinkFactor$). At this point, the $Root$ node obtains an *error budget* equal to $(1 - shrinkFactor) \times E\_Global$, which it can then distribute to the data sources. The decision of which data sources will increase their window W_i is based on the calculation of a *Burden Score* metric B_i for each data source, which is defined as: $B_i = \frac{C_i}{P_i \times W_i}$. In this formula, C_i is the cost of sending a value from the data source S_i to the $Root$ and P_i is the *estimated streamed update period*, defined as the estimated amount of time between consecutive transmissions for S_i over the last period Upd. For a single query over the data sources, it is shown in [11] that the goal would be to try and have all the burden scores be equal. Thus, the $Root$ node selects the data sources with the largest deviation from the target burden score (these are the ones with the largest burden scores in the case of a single query) and sends them messages to increase the width of their windows by a given amount. The process is repeated every Upd time units.

A key drawback of the approach, when applied over sensor networks, is the requirement from each node that makes a transmission because its measured value was outside its error filter to also transmit its burden score. In hierarchical topologies, where messages from multiple nodes are aggregated on their path to the $Root$ node, this may result in a significant amount of side-information that needs to be communicated along with the aggregate value (one burden score for each node that made a transmission), which defeats the purpose of the algorithm, namely the reduction in the amount of information being transmitted in the network.

3.2 A Framework for Hierarchical Data Aggregation

The model that we consider in this paper is a mixture of the Olston [11] and TAG [9] models. Similarly to [11], we consider error-tolerant applications where precision constraints are specified and filters are installed at the data sources. Unlike [11], we consider a hierarchical view of the data sources, based on the paths that the aggregate values follow towards the *Root* node.

We assume that the aggregation tree (ex: Fig. 1) for computing and propagating the aggregate has already been established. Techniques for discovering and modifying the aggregation tree are illustrated in [9]. There are two types of nodes in the tree. *Active* nodes, marked grey in the figure, are nodes that collect measurements. *Passive* nodes (for example, node 2 in the figure) are intermediate nodes in the tree that do not record any data for the query but rather aggregate partial results from their descendant nodes. By

Table 1. Symbols Used in our Algorithms

Symbol	Description
N_i	Sensor node i
W_i	The width of the filter of sensor N_i
$E_i = W_i/2$	Maximum permitted error in node N_i
E_Sub_i	Maximum permitted error in entire subtree of node N_i
E_Global	Maximum permitted error of the application
Upd	Update period of adjusting error filters
$shrinkFactor$	Shrinking Factor of filter widths
T	Number of nodes in the aggregation tree
$Root$	The node initiating the continuous query
$Gain$	The estimated gain of allocating additional error to the node
$CumGain$	The estimated gain of allocating additional error to the the node's entire subtree
$CumGain\_Sub[i]$	The estimated gain of allocating additional error to the the node's i-th subtree

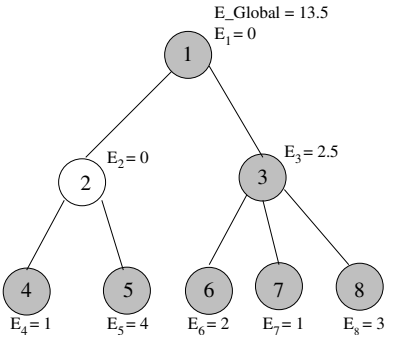

Fig. 1. Sample Aggregation Tree

default all leaf nodes are active, while intermediate nodes may be either active or passive. Our algorithms will install a filter to any node N_i in the aggregation tree, independently on whether the node is an active or passive one. This is a distinct difference from the framework of [11], where filters are assigned only to active nodes.

Similarly to the work in [11], our work covers queries containing any of the five standard aggregate functions: SUM, AVG, COUNT, MIN and MAX. The COUNT function can always be computed exactly, while the AVG function can be computed by the SUM and COUNT aggregates. The use of MIN and MAX is symmetric. The work in [11] demonstrated how these queries can be treated as a collection of AVG queries. The same observations apply in our case as well. In this paper, the focus of our discussion will thus be on queries involving the SUM aggregate function.

Figure 1 shows the maximum error of each filter for a query calculating the SUM aggregate over the active nodes of the tree. Notice that the sum of the errors specified is equal to the maximum error that the application is willing to accept (E_Global). Moreover, there is no point in placing an error filter in the $Root$ node, since this is where the result of the query is being collected.

We now describe the propagation of values in the aggregation tree, using a radio synchronization process similar to the one in [9]. During an epoch duration and within the time intervals specified in [9] the sensor nodes in our framework operate as follows:

- An active leaf node i obtains a new measurement and forwards it to its parent if the new measurement lies outside the interval $[L_i, H_i]$ specified by its filter.
- A passive (non-leaf) node awaits for messages from its children. If one or more messages are received, they are combined and forwarded to its own parent only if the new partial aggregate value of the node's subtree does not lie within the interval specified by the node's filter. Otherwise, the node remains idle.
- An active non-leaf node obtains a new measurement and waits for messages from its children nodes as specified in [9]. The node then recomputes the partial aggregate on its subtree (which is the aggregation of its own measurement with the values received by its child-nodes) and forwards it to its parent only if the new partial aggregate lies outside the interval specified by the node's filter.

Along this process, the value sent from a node to its parent is either (i) the node's measurement if the node is a leaf or (ii) the partial aggregate of all measurements in the node's subtree (including itself) if the node is an intermediate node. In both cases, a node remains idle during an epoch if the newly calculated partial aggregate value lies within the interval $[L_i, H_i]$ specified by the node's filter. This is a distinct difference from [11], where the error filters are applied to the values of the data sources, and not on the partial aggregates calculated by each node.

Details on the operation of the sensor nodes will be provided in the following sections. In our discussion hereafter, whenever we refer to Olston's algorithm we will assume the combination of its model with the model of TAG, which aggregates messages within the aggregation tree. Compared to [11], we introduce two new ideas:

1. A new algorithm for adjusting the widths of filters in the nodes: Our algorithm bases its decisions on estimates of the expected gain of allocating additional error to different subtrees. In this way, our algorithm is more robust to the existence of volatile nodes, nodes where the value of the measured quantity changes significantly in each epoch. Moreover, the estimation of gains is performed based only on local statistics for each node (*that take into account the tree topology*), in contrast to Olston's framework where sources are independent and a significant amount of information needs to be propagated to the $Root$ node. These statistics are a single value for each node, which can be piggy-backed at the transmitted messages. The details of this process are presented in Sect. 5.
2. A hierarchical-based mode of operation: The filters in non-leaf nodes are used in a mode that may potentially filter messages transmitted from their children nodes, and not just from the node itself. We denote this type of operation as *residual-based* operation, and also denote the error assigned to each node in this case as a *residual* error. We show that under the *residual-based* mode nodes may transmit significantly fewer messages than in a *non-residual* operation because of the coalescing of updates that cancel out and are not propagated all the way to the $Root$ node.

4 Problems of Existing Techniques

We now discuss in detail the shortcomings of the algorithms of [11] when applied in the area of sensor networks, and motivate the solutions that we present in the following section.

Hierarchical Structure of Nodes. As we have mentioned above, the sensor nodes that either measure or forward data relevant to a posed continuous query form an aggregation tree, which messages follow on their path to the node that initiated the query. Due to the hierarchical structure of this tree, transmitted values by some node in the tree may be aggregated at some higher level (closer to the $Root$ node) with values transmitted by other sensor nodes. This has two main consequences: (1) While each data transmission by a node N_i may in the worst case require transmitting as many messages as the distance of N_i from the $Root$ node, the actual cost in most cases will be much smaller; and (2) The actual cost of the above transmission is hard to estimate, since this requires knowledge of which sensors transmitted values, and their exact topology. However, this is an unrealistic

scenario in sensor networks, since the additional information required would have to be transmitted along with the aggregate values. The cost of transmitting this additional information would outweight all the benefits of our (or Olston's) framework.

The calculation of the *Burden Score* in [11] (see Sect. 3.1) requires knowledge of the cost of each transmission, since this is one of the three variables used in the formula. Therefore, the techniques introduced in [11] can be applied in our case only by using a heuristic function for the cost of the message. However, it is doubtful that any heuristic would be an accurate one. Consider, for example, the following two cases: (1) All the nodes in the tree need to transmit their observed value. Then, if the tree contains T nodes, under the TAG [9] model exactly $T - 1$ messages will be transmitted (the $Root$ node does not need to transmit a message), making the average cost of each transmission to be equal to 1; and (2) If just one node needs to transmit its observed value, then this value may be propagated all the way to the $Root$. This always happens under the framework of [11], but not in our framework. In this case, the cost of the message will be equal to the distance (in number of hops or tree edges) of the node from the $Root$, since this is the number of messages that will ultimately be transmitted.

In our algorithms we will thus seek to use a metric that will not be dependent on the cost of each transmission, since this is not only an estimate that is impractical to calculate, but also because it varies substantially over time, due to the adaptive nature of our algorithms, and the (possibly) changing characteristics of the data observed by the sensor nodes.

Robustness to Volatile Nodes. One of the principle ideas behind the adaptive algorithms presented in [11] is that an increase in the width of a filter installed in a node will result in a decrease at the number of transmitted messages by that node. While this is an intuitive idea, there are many cases, even when the underlying distribution of the observed quantity does not change, where an increase in the width of the filter does not have any impact in the number of transmitted messages. To illustrate this, consider a node whose values follow a random step pattern, meaning that the observed value at each epoch differs by the observed value in the previous epoch by either $+\Delta$ or $-\Delta$. In this case, any filter with a window whose width is less than $2 \times \Delta$ will not be able to reduce the number of transmitted messages. A similar behavior may be observed in cases where the measured quantity exhibits a large variance. In such cases, even a filter with considerable width may not be able to reduce but a few, if any, transmissions.

The main reason why this occurs in Olston's algorithm is because the Burden Score metric being used does not give any indication about the expected benefit that we will achieve by increasing the width of the installed filter at a node. In this way, a significant amount of the maximum error budget that the application is willing to tolerate may be spent on a few nodes whose measurements exhibit the aforementioned volatile behavior, without any real benefit.

In the algorithms that are presented in the next section we propose a method for distributing the available error using a gain-based policy, which will distribute the error budget to subtrees and nodes based on the expected gain of each decision. Our algorithms identify very volatile nodes which incur very small potential gains and isolate them. Even though the result of this approach is a constant transmission by these nodes, the cost of these transmissions is amortized due to the aggregation of messages that

we described above. Our experiments validate that such an approach may result in a significant decrease in the number of transmitted messages, due to the more efficient utilization of the error budget in other non-volatile sensor nodes.

Negative Correlations in Neighboring Areas. According to the algorithms in [11], each time the value of a measured quantity at a node N_i lies outside the interval specified by the filter installed at N_i, then the new value is transmitted and propagated to the $Root$ node. However, there might be cases when changes from nodes belonging to different subtrees of the aggregation tree either cancel out each other, or result in a very small change in the value of the calculated aggregate. This may happen either because of a random behavior of the data, or because of some properties of the measured quantity. Consider for example the aggregation tree of Fig. 1, and assume that each node observes the number of items moving within the area that it monitors. If some objects move from the area of node 4 to the area of node 5, then the changes that will be propagated to node 2 will cancel out each other. Even in cases when the overall change of the aggregate value is non-zero, but relatively small, and such a behavior occurs frequently, we would like our algorithms to be able to detect this and filter these messages without having them being propagated to the $Root$ node.

The algorithm that we will present in the next section introduces the notion of a *residual* mode of operation to deal with such cases. The width of the filters installed in non-leaf nodes is increased in cases when it is detected that such an action may result in filtering out a significant number of messages from descendant nodes.

5 Our Algorithms

In this section we first describe in detail our framework and demonstrate the operation of the nodes. We then present the details of our algorithms for dynamically modifying the widths of the filters installed in the sensor nodes.

5.1 Operation of Nodes

We now describe the operation of each sensor node using the notation of Table 1. Detailed pseudocode for this operation can be found in the full version of this paper. A filter is initially installed in each node of the aggregation tree, except from the Root node. The initial width of each filter is important only for the initial stages of the network's operation, as our dynamic algorithm will later adjust the sizes of the filters appropriately. For example, another alternative would have been to install filters with non-zero width in the initialization phase only to active nodes of the network. In our algorithms we initialize the widths of the error filters similarly to the *uniform allocation* method. For example, in the case when the aggregate function is the function SUM, then each of the $T-1$ nodes in the aggregation tree besides the Root node is assigned the same fraction $E\_Global/(T-1)$ of the error E_Global that the application is willing to tolerate.

In each epoch, the node obtains a measurement (V_Curr) related to the observed quantity only if it is an active node, and then waits for messages from its children nodes containing updates to their measured aggregate values. We here note that each

node computes a partial aggregate based on the values reported by its children nodes in the tree. This is a recursive procedure which ultimately results in the evaluation of the aggregate query at the $Root$ node. After waiting for messages from its children nodes, the current node computes the new value of the partial aggregate based on the most current partial aggregate values $LastReceived[i]$ it has received from its children.

The new aggregate is calculated using a $Combine$ function, which depends on the aggregate function specified by the query. In Table 2 we provide its implementation for the most common aggregate functions. In the case of the AVG aggregate function, we calculate the sum of the values observed at the active nodes, and then the $Root$ node will divide this value with the number of active nodes participating in the query.

After calculating the current partial aggregate, the node must decide whether it needs to transmit a measurement to its parent node or not. This depends on the operation mode being used. In a *non-residual* mode, the node would have to transmit a message either when the value of the measured quantity at the node itself lies outside its filter, or when at least one of the subtrees has transmitted a message and the new changes do not exactly cancel out each other. This happens because in the *non-residual* mode (e.g. the original algorithm of [11]) the error filters are applied to the values measured by each node, and not to the partial aggregates of the subtree. On the contrary, in a *residual* mode of operation, which is the mode used in our algorithms, the node transmits a message only when the value of the new partial aggregate lies outside the node's filter. In both modes of operation the algorithm that distributes the available error enforces that for any node N_i, its calculated partial aggregate will never deviate by more than E_Sub_i from the actual partial aggregate of its subtree (ignoring propagation delays and lost messages). At each epoch the node also updates some statistics which will be later used to adjust the widths of the filters. The *cumulative gain* of the node, which is a single value, is the only statistic propagated to the parent node at each transmission. This adjustment phase is performed every Upd epochs. The first step is for all nodes to shrink the widths of their filters by a shrinking factor $shrinkFactor$ ($0 \leq shrinkFactor < 1$). After this process, the $Root$ node has an error budget of size $E\_Global \times (1 - shrinkFactor)$, where E_Global is the maximum error of the application, that it can redistribute recursively to the nodes of the network. Details of the adjustment process will be given later in this section.

5.2 Calculating the Gain of Each Node

Our algorithm updates the width of the filter installed in each node by considering the potential gain of increasing the error threshold at a sensor node, which is defined as the amount of messages that we expect to save by allocating more resources to the node. The result of using this gain-based approach is a robust algorithm that respects the hierarchy imposed by the aggregation tree and, at the same time, is able to identify volatile data sources and eliminate them from consideration. This computation of potential gains, as we will show, requires only local knowledge, where each node simply considers statistics from its children nodes in the aggregation tree. In Fig. 2 we show the expected behavior of a sensor node N_i, varying the width of the its filter W_i. The y-axis plots the number of messages sent from this node to its parent in the aggregation tree in a period of Upd epochs. Assuming that the measurement on the node is not constant, a zero width filter ($W_i = E_i = 0$) results in one message for each of the Upd epochs. By increasing the

Table 2. Definition of the `Combine` function

Aggregate Function	Implementation of Combine Function
SUM	$V\_Curr + \sum_i LastReceived[i]$
AVERAGE	$V\_Curr + \sum_i LastReceived[i]$
MAX	$\max\{V\_Curr, \max_i\{LastReceived[i]\}\}$
MIN	$\min\{V\_Curr, \min_i\{LastReceived[i]\}\}$

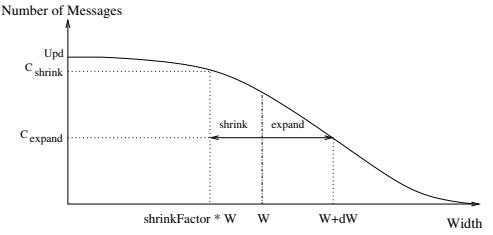

Fig. 2. Potential Gain of a Node

width of the filter, the number of messages is reduced, up to the point that no messages are required. Of course, in practice, this may never happen as the width of the filter required may exceed the global error constraint E_Global. Some additional factors that can make a node deviate from the typical behavior of Fig. 2 also exist. As an example, the measurement of the node may not change for some period of time exceeding Upd. In such a case, the curve becomes a straight line at y=0 and no messages are being sent (unless there are changes on the subtree rooted at the node). In such cases of very stable nodes, we would like to be able to detect this behavior and redistribute the error to other, more volatile nodes. At the other extreme, node N_i may be so volatile that even a filter of considerable width will not be able suppress any messages. In such a case the curve becomes a straight line at y=Upd. Notice that the same may happen because of a highly volatile node N_j that is a descendant of N_i in the aggregation tree.

In principle, we can not fully predict the behavior of a node N_i unless we take into account its interaction will all the other nodes in its subtree. Of course, a complete knowledge of this interaction is infeasible, due to the potentially large amounts of information that are required, as described in Sect. 4. We will thus achieve this by computing the potential gains of adjusting the width of the node's filter W_i, using simple, *local* statistics that we collect during the query evaluation.

Let W_i be the width of the filter installed at node N_i at the last update phase. The node also knows the $shrinkFactor$ that is announced when the query is initiated. Unless the adaptive procedure decides to increase the error of the node, its filter's width is scheduled to be reduced to $shrinkFactor \times W_i$ in the next update phase, which takes place every Upd epochs. The node can estimate the effects of this change as follows. At the same time that the node uses its filter W_i to decide whether or not to send a message to its parent, it also keeps track of its decision assuming a filter of a smaller width of $shrinkFactor \times W_i$. This requires a single counter C_{shrink} that will keep track of the number of messages that the node would have sent if its filter was reduced. C_{shrink} gives as an estimate of the negative effect of reducing the filter of N_i. Since we would also like the node to have a chance to increase its filter, the node also computes the number of messages C_{expand} in case its filter was increased by a factor dW to be defined later.[1]

[1] Even though this computation, based on two anchor points, may seem simplistic, there is little more that can truly be accomplished with only local knowledge, since the node cannot possibly know exactly which partial aggregates it would have received from its children in the case of either a smaller or a larger filter, because these partial aggregates would themselves depend on the corresponding width changes in the filters of the children nodes.

Our process is demonstrated in Fig. 2. Let $\delta G \geq 0$ be the reduction in the number of messages by changing the width from $shrinkFactor \times W_i$ (which is the default in the next update phase) to $W_i + dW$. The *potential gain* for the node is defined as:

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

It is significant to note that our definition of the *potential gain* of a node is independent on whether the node is active or not, since the algorithm for deciding whether to transmit a message or not is only based on the value of the partial aggregate calculated for the node's entire subtree. Moreover, the value of dW is not uniquely defined in our algorithms. In our implementation we use the following heuristic for the computation of gains:

- For leaf nodes, we use $dW = \frac{E\_Global}{N_{active}}$, N_{active} being the number of active nodes in the aggregation tree.
- For non-leaf nodes, in the residual mode, we need a larger value of dW, since the expansion of the node's filter should be large enough to allow the node to coalesce negative correlations in the changes of the aggregates on its children nodes. As a heuristic, we have been using $dW = num\_children_i \times \frac{E\_Global}{N_{active}}$, where $num\_children_i$ is the number of children of node N_i.

These values of dW have been shown to work well in practice on a large variety of tested configurations. We need to emphasize here that these values are used to give the algorithm an estimate on the behavior of the sensor and that the actual change in the widths W_i of the filters will also be based on the amount of "error budget" available and the behavior of all the other nodes in the tree.

Computation of Cumulative Gains. The computation of the potential gains, as explained above, may provide us with an idea of the effect that modifying the size of the filter in a node may have, but is by itself inadequate as a metric for the distribution of the available error to the nodes of its subtree. This happens because this metric does not take into account the corresponding gains of descendant nodes in the aggregation tree. Even if a node may have zero potential gain (this may happen, for example, if either the node itself or some of its descendants are very volatile), this does not mean that we cannot reduce the number of transmitted messages in some areas of the subtree rooted at that node.

Because of the top-down redistribution of the errors that our algorithm applies, if no budget is allocated to N_i by its parent node then all nodes in the subtree of N_i will not get a chance to increase their error thresholds and this will eventually lead to every node in that subtree to send a new message on each epoch, which is clearly an undesirable situation. Thus, we need a way to compute the *cumulative gain* on the subtree of N_i and base the redistribution process on that value. In our framework we define the cumulative gain on a node N_i as:

$$CumGain_i = \begin{cases} Gain_i & N_i \text{ is a leaf node} \\ Gain_i + \sum_{N_j \in children(N_i)} CumGain\_Sub[j] & \text{otherwise} \end{cases} .$$

This definition of the cumulative gain has the following desirable properties: (1) It is based on the computed gains ($Gain_i$) that is purely a local statistic on a node N_i; and (2)

The recursive formula can be computed in a bottom-up manner by having nodes piggyback the value of their cumulative gain in each message that they transmit to their parent along with their partial aggregate value. This is a single number that is being aggregated in a bottom-up manner, and thus poses a minimal overhead.[2] Moreover, transmitting the cumulative gain is necessary only if its value has changed (and in most cases only if this change is significant) since the last transmission of the node.

5.3 Adjusting the Filters

The algorithm for adjusting the widths of the filters is based on the cumulative gains calculated at each node. Every Upd epochs, we mentioned before that all the filters shrink by a factor of $shrinkFactor$. This results in an error budget of $E\_Global \times (1 - shrinkFactor)$ which the $Root$ node can distribute to the nodes of the tree. Each node N_i has statistics on the potential gain of allocating error to the node itself ($Gain_i$), and the corresponding cumulative gain of allocating error to each of its subtrees. Using these statistics, the allocation of the errors is performed as follows, assuming that the node can distribute a total error of $E\_Additional$ to itself and its descendants:

1. For each subtree j of node N_i, allocate error E_Sub_j proportional to its cumulative gain: $E\_Additional_j = \frac{E\_Additional \times CumGain\_Sub[j]}{Gain_i + \sum_{N_j \in children(N_i)} CumGain\_Sub[j]}$. This distribution is performed only when this quantity is at least equal to E_Global/N_{active}.
2. The remaining error budget is distributed to the node itself.

The fraction of the error budget allocated to the node itself and to each of the subtrees is analogous to the expected benefit of each choice. The only additional detail is that in case when the error allocated to a subtree of node N_i is less than the E_Global/N_{active} value, then we do not allocate any error in that subtree, and allocate this error to node N_i itself. This is done to avoid sending messages for adjusting the filters when the error budget is too small.

6 Experiments

We have developed a simulator for sensor networks that allows us to vary several parameters like the number of nodes, the topology of the aggregation tree, the data distribution etc. The synchronization of the sensor nodes is performed, for all algorithms, as described in TAG [9]. In our experiments we compare the following algorithms:

1. BBA (Burden-Based Adjustment): This is an implementation of the algorithm presented in [11] for the adaptive precision setting of cached approximate values.
2. Uni: This is a static setting where the error is evenly distributed among all active sensor nodes, and therefore does not incur any communication overhead for adjusting the error thresholds of the nodes.

[2] In contrast, the algorithm of [11], requires each node to propagate the burden scores of all of its descendant nodes in the aggregation tree whose observed values was outside their error filters.

3. PGA (Potential Gains Adjustment): This is our precision control algorithm, based on the potential gains (see Sect. 5), for adjusting the filters of the sensor nodes.

For the PGA and BBA algorithms we made a few preliminary runs to choose their internal parameters (*adjustment period*, *shrink percentage*). Notice that the *adjustment period* determines how frequently the precision control algorithm is invoked, while the *shrink percentage* determines how much of the overall error budget is being redistributed. Based on the observed behavior of the algorithms, we have selected the combination of values of Table 3(a) as the most representative ones for revealing the "preferences" of each algorithm. The first configuration (Conf1) consistently produced good results, in a variety of tree topologies and datasets, for the PGA algorithm, while the second configuration (Conf2) was typically the best choice for the BBA algorithm. In the BBA algorithm we also determined experimentally that distributing the available error to 10% of the nodes with the highest burden scores was the best choice for the algorithm.

The initial allocation of error thresholds was done using the uniform policy. We then used the first 10% of epochs as a warm-up period for the algorithms to adjust their thresholds and report the results (number of transmitted messages) for the later 90%. We used synthetic data, similar in spirit to the data used in [11]. For each simulated active node, we generated values following a random walk pattern, each with a randomly assigned step size in the range $(0 \ldots 2]$. We further added in the mix a set of "unstable nodes" whose step size is much larger: $(0 \ldots 200]$. These volatile nodes allow us to investigate how the different algorithms adapt to noisy sensors. Ideally, when the step-size of a node is comparable to the global error threshold, we would like the precision control algorithm to restrain from giving any of the available budget to that node at the expense of all the other sensor nodes in the tree. We denote with $P_{unstable}$ the probability of an active node being unstable.

$P_{unstable}$ describes the volatility of a node in terms of the magnitude of its data values. Volatility can also be expressed in the orthogonal temporal dimension. For instance some nodes may not update their values frequently, while others might be changing quite often (even by small amounts, depending on their step size). To capture this scenario, we further divide the sensor nodes in two additional classes: *workaholics* and *regulars*. Regular sensors make a random step with a fixed probability of 1% during an epoch. Workaholics, on the other hand, make a random step on every epoch. We denote with $P_{workaholic}$ the probability of an active node being workaholic.

We used three different network topologies denoted as T_{leaves}, T_{all} and T_{random}. In T_{leaves} the aggregation tree was a balanced tree with 5 levels and a fan-out of 4 (341 nodes overall). For this configuration all active nodes were at the leaves of the tree. In T_{all}, for the same tree topology, all nodes (including the $Root$) were active. Finally in T_{random} we used 500 sensor nodes, forming a random tree each time. The maximum fan-out of a node was in that case 8 and the maximum depth of the tree 6. Intermediate nodes in T_{random} were active with probability 20% (all leaf nodes are active by default).

In all experiments presented here, we executed the simulator 10 times and present here the averages. In all runs we used the SUM aggregate function (the performance of AVG was similar).

Benefits of Residual Operation and Sensitivity Analysis. The three precision control algorithms considered (Uni, PGA, BBA) along with the mode of operation (residual: Res, non-residual: $NoRes$) provide us with six different choices (Uni+Res, Uni+$NoRes$ ...). We note that BBA+$NoRes$ is the original algorithm of [11] running over TAG, while BBA+Res is our extension of that algorithm using the residual mode of operation. The combination PGA+Res denotes our algorithm. In this first experiment we investigate whether the precision control algorithms benefit from the use of the residual mode of operation. We also seek their preferences in terms of the values of parameters *adjustment period* and *shrink percentage*.

We used a synthetic dataset with $P_{unstable}$=0 and $P_{workaholic}$=0.2. We then let the sensors operate for 40,000 epochs using a fixed error constraint E_Global=500. The average value of the SUM aggregate was 25,600, meaning that this E_Global value corresponds to a relative error of about 2%. In Table 3(b) we show the total number of messages in the sensor network for each choice of algorithm and tree topology and each selection of parameters. We also show the number of messages for an exact computation of the SUM aggregate using one more method, entitled as (E_Global=0)+Res, which places a zero width filter in every node and uses our residual mode of operation for propagating changes. Effectively, a node sends a message to its parent only when the partial aggregate on its subtree changes. This is nothing more than a slightly enhanced version of TAG. The following observations are made:

- Using a modest E_Global value of 500 (2% relative error), we are able to reduce the number of messages by 7.6-9.9 times (in PGA+Res) compared to (E_Global=0)+Res. Thus, error-tolerate applications can significantly reduce the number of transmissions, resulting in great savings on both bandwidth and energy consumption.
- Algorithm PGA seems to require fewer invocations (larger *adjustment period*) but with a larger percentage of the error to be redistributed (a smaller *shrink percentage* results in a wider reorganization of the error thresholds). In the table we see that the number of messages for the selection of values of Conf1 is always smaller. Intuitively, larger adjustment periods, allow for more reliable statistics on the computation of potential gains. On the contrary, BBA seems to behave better when filters are adjusted more often by small increments. We also note that BBA results in a lot more messages than PGA, no matter which configuration is used.
- The PGA algorithm, when using the residual operation (PGA+Res), results in substantially fewer messages that all the other alternatives. Even when using the non-residual mode of operation, PGA outperforms, significantly, the competitive algorithms.
- BBA seems to benefit only occasionally from the use of the residual operation. The adjustment of thresholds based on the burden of a node can not distinguish on the true cause of a transmission (change on local measurement or change in the subtree) and does not seem to provide a good method of adjusting the filters with respect to the tree hierarchy.

In the rest of the section we investigate in more details the performance of the algorithms based on the network topology and the data distribution. For PGA we used the residual mode of operation. For BBA we tested both the residual and non-residual modes and present here the best results. (as seen on Table 3(b), the differences were very

small) We configured PGA using the values of Conf1 and BBA using the values of Conf2 that provided the best results per case.

Sensitivity on Temporal Volatility of Sensor Measurements. We here investigate the performance of the algorithms when varying $P_{workaholic}$ and for $P_{unstable}$=0. We first fixed $P_{workaholic}$ to be 20%, as in the previous experiment. In Fig. 3 we plot the total number of messages in the network (y-axis) for 40,000 epochs when varying the error constraint E_Global from 100 to 2,000 (8% in terms of relative error). Depending on E_Global, PGA results in up to 4.8 times fewer messages than BBA and up 6.4 times fewer than Uni. These differences arise from the ability of PGA to place, judiciously, filters on passive intermediate sensor nodes and exploit negative correlations on their subtree based on the computed potential gains. Algorithm BBA may also place filters on the intermediate nodes (when the residual mode is used) but the selection of the widths of the filters based on the burden scores of the nodes was typically not especially successful in our experiments.

Figures 4 and 5 repeat the experiment for the T_{all} and T_{random} configurations. For the same global error threshold, PGA results in up to 4 times and 6 times fewer messages than BBA and Uni respectively. In Fig. 6 we vary $P_{workaholic}$ between 0 and 1 for T_{all} (best network topology for BBA). Again PGA outperforms the other algorithms. An important observation is that when the value of $P_{workaholic}$ is either 0 or 1, all the methods behave similarly. In this case all the nodes in the network have the same characteristics, so it is not surprising that Uni performs so well. The PGA and BBA algorithms managed to filter just a few more messages than Uni for these cases, but due to their overhead for updating the error thresholds of the nodes, the overall number of transmitted messages was about the same for all techniques.

Sensitivity in Magnitude of Sensor Measurements. In Figs. 7, 8 we vary the percentage of unstable nodes (nodes that make very large steps) from 0 to 100% and plot the total number of messages for T_{all} and T_{random} ($P_{workaholic}$=0, E_Global=500). For $P_{unstable}$=1 the error threshold (500) is too small to have an effect on the number of messages and all algorithms have practically the same behavior. For smaller values of $P_{unstable}$, algorithm PGA results in a reduction in the total number of messages by a factor of up to 3.8 and 5.5 compared to BBA and Uni respectively.

7 Conclusions

In this paper we proposed a framework for in-network data aggregation that supports the evaluation of aggregate queries in error-tolerant applications over sensor networks. Unlike previous approaches, our algorithms exploit the tree hierarchy that messages follow in such applications to significantly reduce the number of transmitted messages and, therefore, increase the lifetime of the network. Our algorithms are based on two key ideas that we presented in this paper. Firstly, the residual mode of operation for nodes in the aggregation tree allows nodes to apply their error filters to the partial aggregates of their subtrees and, therefore, potentially suppress messages from being transmitted towards the root node of the tree. A second key idea is the use of simple

Table 3. (a) Used Configurations; (b) First number is total number of messages (in thousands) in the network when using parameters of Conf1, second for Conf2. Uni does not use these parameters. Best numbers for each algorithm in bold

	Configuration	
Parameters	Conf1	Conf2
Upd shrinkFactor	50 0.6	20 0.95
Invocations	Fewer	Frequent
Error Amount Redistributed	Significant	Smaller

(a)

	T_{leaves}	T_{all}	T_{random}
PGA+Res	**423** / 978	**479** / 903	**677** / 1,207
PGA+$NoRes$	463 / 924	558 / 894	830 / 1,454
BBA+Res	2,744 / 1,654	2,471 / **1,426**	3,775 / 2,657
BBA+$NoRes$	3,203 / **1,394**	2,967 / 1,481	4,229 / **2,474**
Uni+Res	**2,568**	**2,451**	**3,906**
Uni+$NoRes$	**2,568**	2,642	4,044
(E_Global=0)+Res	**4,176**	**4,176**	**5,142**

(b)

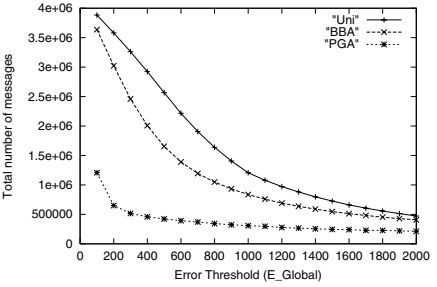

Fig. 3. Messages vs. E_Global for T_{leaves}

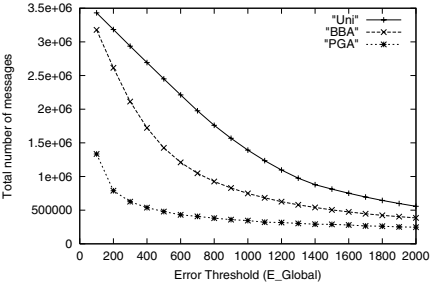

Fig. 4. Messages vs. E_Global for T_{all}

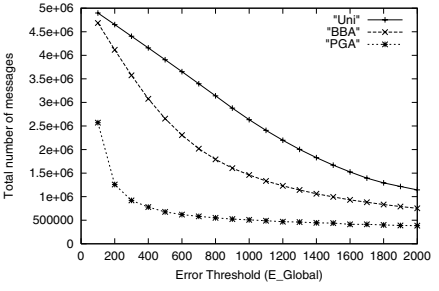

Fig. 5. Messages vs. E_Global for T_{random}

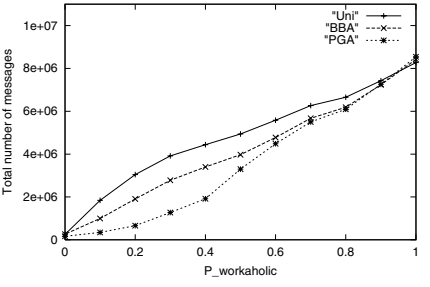

Fig. 6. Messages vs. $P_{workaholic}$ for T_{all}

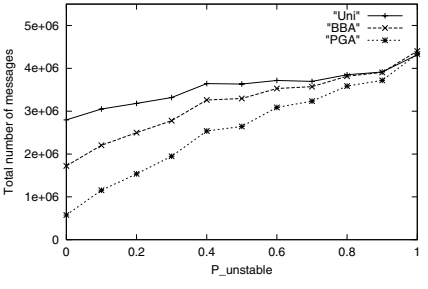

Fig. 7. Messages vs. $P_{unstable}$ for T_{all}

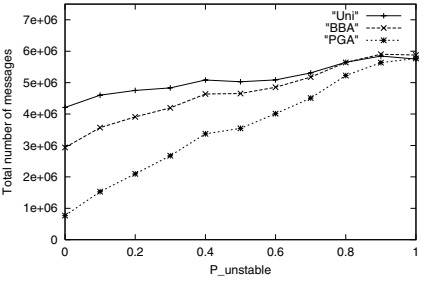

Fig. 8. Messages vs. $P_{unstable}$ for T_{random}

and local statistics to estimate the potential gain of allocating additional error to nodes in a subtree. This is a significant improvement over previous approaches that require a large amount of information to be continuously transmitted to the root node of the tree, therefore defeating their purpose, namely the reduction in the amount of transmitted information in the network. Through an extensive set of experiments, we have shown in this paper that while the distribution of the error based on the computed gains is the major factor for the effectiveness of our techniques compared to other approaches, the fusion of the two ideas provides even larger improvements.

References

1. D. Barbará and H. Garcia-Molina. The Demarcation Protocol: A Technique for Maintaining Linear Arithmetic Constraints in Distributed Database Systems. In *EDBT*, 1992.
2. A. Cerpa and D. Estrin. ASCENT: Adaptive Self-Configuring sEnsor Network Topologies. In *INFOCOM*, 2002.
3. J. Chen, D.J. Dewitt, F. Tian, and Y. Wang. NiagaraCQ: A Scalable Continuous Query System for Internet Databases. In *ACM SIGMOD*, 2000.
4. R. Cheng, D. V. Kalashnikov, and S. Prabhakar. Evaluating Probabilistic Queries over Imprecise Data. In *ACM SIGMOD Conference*, pages 551–562, 2003.
5. J. Considine, F. Li, G. Kollios, and J. Byers. Approximate Aggregation Techniques for Sensor Databases. In *ICDE*, 2004.
6. D. Estrin, R. Govindan, J. Heidermann, and S. Kumar. Next Century Challenges: Scalable Coordination in Sensor Networks. In *MobiCOM*, 1999.
7. J. Heidermann, F. Silva, C. Intanagonwiwat, R. Govindanand D. Estrin, and D. Ganesan. Building Efficient Wireless Sensor Networks with Low-Level Naming. In *SOSP*, 2001.
8. C. Intanagonwiwat, D. Estrin, R. Govindan, and J. Heidermann. Impact of Network Density on Data Aggregation in Wireless Sensor Networks. In *ICDCS*, 2002.
9. S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. Tag: A Tiny Aggregation Service for ad hoc Sensor Networks. In *OSDI Conf.*, 2002.
10. S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. The Design of an Acquisitional Query processor for Sensor Networks. In *ACM SIGMOD Conf*, June 2003.
11. C. Olston, J. Jiang, and J. Widom. Adaptive Filters for Continuous Queries over Distributed Data Streams. In *ACM SIGMOD Conference*, pages 563–574, 2003.
12. M. A. Sharaf, J. Beaver, A. Labrinidis, and P. K. Chrysanthis. TiNA: A Scheme for Temporal Coherency-Aware in-Network Aggregation. In *MobiDE*, 2003.
13. N. Soparkar and A. Silberschatz. Data-value Partitioning and Virtual Messages. In *Proceedings of PODS*, pages 357–367, Nashville, Tennessee, April 1990.
14. D.B. Terry, D. Goldberg, D. Nichols, and B.M. Oki. Continuous Queries over Append-Only Databases. In *ACM SIGMOD*, 1992.
15. Y. Yao and J. Gehrke. The Cougar Approach to In-Network Query Processing in Sensor Networks. *SIGMOD Record*, 31(3):9–18, 2002.