In-Network Evaluation of Spatial Aggregation in Sensor Networks

Dina Goldin, Ayferi Kutlu
University of Connecticut
Storrs, CT, USA

Abstract

We consider the problem of evaluating spatial aggregation in sensor networks over geo-aware sensors, communicating via a routing tree. Given a set of spatial regions, such as a spatial database relation, spatial aggregation involves the aggregation of dynamic sensor readings over each of these regions simultaneously. Nested spatial aggregation involves one more level of aggregation, combining these aggregates into a single aggregate value.

We propose to maintain additional rivet information with the routing tree, which enables us to determine when an aggregate record for any spatial region is ready to be evaluated. In the absence of this information, evaluation cannot be done until a record has reached the root node. In some important scenarios, such as nested spatial aggregation and filtering predicates, this allows us to reduce the amount of communication involved in query evaluation, by pushing more processing into the network. Preliminary results show that rivet information can reduce in-network communication by up to 16%, without any drop in accuracy.

1 Introduction

Unlike traditional database applications, where spatial considerations are often irrelevant (except as expressed by traditional attributes such as address or zip code), it is expected that most applications of sensor networks will involve queries that combine spatial data with spatial data [RSV01, W99]. Furthermore, it is expected that in many sensor networks, the sensors will be geo-aware, i.e. they will know their location in some local or global coordinate system; there are several ways to accomplish geo-awareness [BHE01, BP00, PCB00].

We assume that sensors are geo-aware, and we focus on spatial querying over geo-aware sensor networks. As is common for the sensor network query setting, we assume a central processor which provides an entry point for querying the sensor network. In our setting, it also serves as a host for a spatial database: queries may involve both the data collected at sensors as well as data from this database. Finally, we assume that a routing tree is maintained over the sensors, whose root communicates directly with the central processor; examples of similar systems are TAG and Cougar.

To avoid information overload, it is natural to aggregate sensor data. In-network aggregation strategies offer great communication savings by reducing the number of records that need to be propagated up to the root. For these reasons, the issue of aggregation has emerged as an important one for sensor network querying [MFHH02, HHM03]. We identify spatial aggregation as a particularly important class of queries for geo-aware spatial sensor networks.

Spatial aggregation involves the computation of some arbitrary aggregate function [Klu82] for each spatial region in some set of regions co-located with the network, such as:

- Find average temperature for each region.
- Find maximum humidity for each region.

To make the processing of spatial aggregation queries more efficient, we propose to tag the routing tree with rivet information. For each region $R$, a single sensor node is designed as its rivet node. If $X$ is the rivet node for $R$, this means that the aggregation record for $R$ is known to be complete once it is processed at $X$; i.e., it will not need to be merged further. By contrast, in the absence of rivet information, we cannot be sure that any aggregate record is complete until it is processed at the root of the routing tree; i.e., the root acts as the rivet node for all the regions.

The ability to process queries in-network is an important way of achieving savings in power consumption for sensor network querying. In the case of spatial aggregation, we show that rivet information allows us to push more processing into the network for many
query scenarios, and achieve significant savings as a result.

**Outline.** First, we define spatial aggregation and nested spatial aggregation, and show how to perform them in-network using existing techniques (section 2). We then define rivet information (section 3). We discuss three scenarios where rivet information can benefit in-network query processing (nested aggregation, filtering predicates, and data pruning), by allowing us to push more computation into the network (section 4). We then discuss how to compute rivet information for a given routing tree, presenting two approaches: exact riveting and approximate riveting (section 5). We conclude with experimental results that motivate the usefulness of rivet information (section 6).

2 Spatial aggregation in geowarn sensor networks

In this section, we introduce spatial aggregation, an important class of aggregation queries for geosensor networks, as well as its variant nested spatial aggregation.

2.1 The spatial aggregation query

In the case of spatial querying over geowarn sensor networks, we often want to aggregate over some set of spatial regions rather than over the entire network. We refer to this operation as spatial aggregation.

**Definition 2.1** Given the relations

- spatial (RegionID, RegionGeometry)
- sensors (SensorID, SensorLocn, Value)

where Value is the dynamic attribute representing the sensor reading of some value of interest (e.g., temperature or humidity), and some aggregation function \( f \), we want to compute the relation

\[ \text{agg}(\text{RegionID}, \text{AggregateValue}) \]

where:

- RegionID is the key of both spatial and agg, and the set of RegionID's is the same for both relations:
  \( \pi_{\text{RegionID}} \text{agg} = \pi_{\text{RegionID}} \text{spatial} \)
- a tuple \((R, v)\) belongs to agg if and only if: given the set \( S_R \) of tuples in sensors which represents sensors lying inside region \( R \), \( v \) is the aggregate of Value's in \( S_R \), with \( f \) as the aggregate function.

For formal semantics of applying aggregate functions, see [Klu82]. Examples of spatial aggregation were presented in the introduction.

Note that we are making an assumption that for each region, there exists at least one sensor lying within that region; if this were relaxed, then the first item above would be changed as follows:

\( \pi_{\text{RegionID}} a_{99} \subseteq \pi_{\text{RegionID}} a_{99} \)

**Example.** Let spatial consist of two regions: \((R_1, \text{RGeom}_1), (R_2, \text{RGeom}_2)\) and sensors consist of four sensors:

\((s_1, \text{locn}_1, 80), (s_2, \text{locn}_2, 70), (s_3, \text{locn}_3, 55), (s_4, \text{locn}_4, 70)\)

where \text{locn}_1 and \text{locn}_2 lie inside region \( R_1 \), and \text{locn}_3 and \text{locn}_4 lie inside region \( R_2 \). Let average be the aggregate function of interest; then the output relation is as follows:

\[(R_1, 75), (R_2, 65)\]

2.2 In-network spatial aggregation

A framework for computing aggregation in sensor networks was originally proposed in [MFHH02]. In this framework, aggregation is performed in-network at the sensor nodes as the data is transmitted up the routing tree; only the output of the aggregate operation reaches the central processor. In-network aggregation is characterized by the following three functions:

- **initializer** \( i \): sensor \( \rightarrow \) state record
  \( i \) produces a state record for a single sensor value; it is applied at the leaves of the routing tree.
- **merger** \( f \): \{state records\} \( \rightarrow \) state record
  \( f \) computes a new state record at an internal node of the routing tree, by combining the state records of all the children.
- **evaluator** \( e \): state record \( \rightarrow \) aggregate value
  \( e \) takes the state record arriving at the root and computes the value of the aggregate.

**Example.** Suppose we are computing the average temperature for all sensors. At each sensor \( X \), \( i(X) = (1, t) \), where \( t \) is the sensor’s temperature. At each internal node \( Y \) of the routing tree, if \( \{(n_1, t_1), \ldots, (n_k, t_k)\} \) are the state records of \( X \)'s children, \( f(X) = (n_0, t_0) \), where \( n_0 \) is the sum of \( \{n_1, \ldots, n_k\} \) and \( t_0 \) is the sum of \( \{t_1, \ldots, t_k\} \). At the root \( Z \), if \( \{(n, t)\} \) is the completed record obtained by having merged all state records, \( e(Z) = t/n \), which is the value of the average temperature for all sensors.

Spatial aggregation can be performed in-network by simultaneously applying the in-network aggregation approach of [MFHH02] to all the regions. This is described next.

**In-network spatial aggregation.**

1. We need an additional attribute, region ID, in each state record; we refer to such a state record as region aggregate record (RAR).
2. The initializer creates a separate RAR for each region to which the sensor belongs, assigning the ID of that region to the record. These records are sent up.
3. The merger only merges RARs that share the same region ID; in effect, we execute many mergers in parallel, one for each region ID appearing among the input records. As a result of all merges, a complete aggregate record for each region arrives at the root of the routing tree.

4. The evaluator computes a separate aggregate value for each complete RAR; the output is a list of tuples with region IDs and aggregate values, as expected.

2.3 Nested Spatial Aggregation

We now consider the case when a query consists of two levels of aggregation, where the lower level represents spatial aggregation; we call such a query nested spatial aggregation. Examples are:

- Find the region with most sensors in it.
- Find the region with minimum maximum temperature.
- Which region has the highest ozone value?

In each example, spatial aggregation is performed first, to find an aggregate value of the regions. The second aggregation determines an aggregate over the results. For example, the first query involves the following aggregation steps:

1. apply `count` to the sensor readings in network, grouped by region;
2. apply `max` to the result in 1.

Note that, while the first level of aggregation can be performed in-network, the second level must be done at the root. This is because we must wait until a RAR has been evaluated at the end of the first level of aggregation, i.e. at the root, before it can be used as input for the second level of aggregation.

2.4 Complete Region Attribute Records

Unlike the general case of in-network aggregation, state records for in-network spatial aggregation are often ready to be evaluated long before they reach the root. For example, if only one of the root’s children has any descendants that are in some region R, then the RAR for R is ready to be evaluated at that child, if not before.

We refer to RARs that are ready to be evaluated as complete, to distinguish them from partial records. Note that whether a record is complete does not depend either on the aggregate function or on the sensor readings. It only depends on the topology of the sensor network, i.e. which nodes lie within the region for that record. In the rest of the paper, we capitalize on this observation.

3 Rivet Information

We now define rivet information and show how this information allows us to push the computation of nested spatial aggregation, as well as of filtering operations, into the network.

3.1 Rivet Lists

To make the processing of spatial aggregation queries more efficient, we introduce a new technique of tagging the routing tree with rivet information.

Definition 3.1 (X rivets R) We say that a sensor node X rivets a region R if all sensors that lie within R are descendants of X in the routing tree.

By definition, for any region R, the set of all sensor nodes that rivet it consists of the common ancestors of all sensors that lie in R. Figure 1 illustrates the three cases when a node S rivets a region: (a) it is outside a region, (b) it is inside, but not a leaf node, (c) it is inside and a leaf node. Clearly, the root of the routing tree rivets all regions, since it is the ancestor of all sensor nodes.

![Figure 1: 3 cases when node S rivets a region](image)

It is easy to show that if X is rivets R, then the aggregation record for R is known to be complete once it is processed at X:

Proposition 3.1 Let q be an RAR whose region ID is R, and let X be a sensor node that rivets R. Then, q is complete at X, i.e. will not be modified (by merging) by any of the ancestors of X.

To make use of this theorem is useful for improving in-network query processing, rivet information needs to be stored in the routing tree.

Definition 3.2 (Rivet information) Given a set of regions \( R = \{R_1, \ldots, R_k\} \), and a query routing tree \( T \), we say that \( T \) is tagged with rivet information if:

- each node \( X \) in \( T \) is associated with a (possibly empty) subset of those regions from \( R \) that are rivet by \( X \); this is known as the rivet list of \( X \);
- the non-empty rivet lists in \( T \) form a partition of \( R \).

We say that \( X \) is a rivet node of \( R \) if \( R \) appears in the rivet list of \( X \).
Figure 2 illustrates a tree with rivet information. The single nodes are labeled with the region id of the region they belong to. The bold labels are rivet lists. While the root is the rivet node for some of the regions, others have rivet nodes deeper in the routing tree.

3.2 Discussion

Rivet information allows us to know when a record is complete without waiting for it to reach the root. This follows from proposition 3.1.

Corollary 3.1 Let $q$ be an RAR whose region ID is $R$, and let $X$ be the rivet node of $R$; i.e., $R$ is listed in $X$’s rivet list. Then, $q$ is complete at $X$.

This corollary implies that once an RAR for any region has been merged at its rivet node, we can evaluate it right away. This increases the amount of processing that can be done in-network, and immediately results in savings in communication when the evaluation function reduces the size of the record, for example in the case of average aggregation function.

Additional efficiency can be realized by recognizing that a complete record, evaluated or not, never needs to be merged again on its way to the central processor. Thus, if a node receives a state record which is marked complete, this record can be moved directly from the input buffer to the output buffer, with no need to pass it to the merge operation or store it in the CPU.

We have identified several other scenarios where rivet information allows us to improve query processing by pushing more computing into the network; they are discussed in section 4. Algorithms for finding rivet information are discussion in section 5.

4 Applications of Rivet Information

In this section, we assume that the routing tree is decorated with rivet information as defined in section 3, and that RARs are evaluated as soon as they are complete. We discuss three scenarios where this allows us to improve query processing, by pushing more computing into the network: nested aggregation, filtering predicates and data pruning.

4.1 (Re)evaluating Nested Aggregation

Nested Spatial Aggregation was discussed in section 2.3. Rivet information allows us to improve the evaluation of nested spatial aggregation in the following way:

1. As soon as an RAR from the first level of aggregation is deemed complete, it is evaluated.
2. Immediately, this evaluated record is input to the initializer for the second level of aggregation, and a “2nd level” state record is created.
3. Whenever two or more “2nd level” records arrive at a sensor node, they are merged into a single “2nd level” aggregate record.

As a result of this modification to nested spatial aggregation, the number of records to be transmitted to the base station is reduced. By contrast, when rivet information is not available, all RARs have to be transmitted to the root.

4.2 Filtering Predicates

Often, sensor readings and aggregates are extracted with certain predicates, such as specified in the where and having clause of an SQL query [MFHH02]. An example is:

```
SELECT avg(volume) FROM Sensors
GROUP BY region
HAVING avg(volume) > threshold
```

These predicates act as filters on records; records that do not satisfy the having constraint do not have to be transmitted to the base station.

In the general case, the network has to make sure it has merged all records from a group before it can apply a filtering predicate. (This requirement can be relaxed for special predicates over monotonic aggregates such as described in [MFHH02].) Up to now, this could only be ensured if the RAR of a group has reached the root node. Now, we can apply the predicate as soon as a RAR has been evaluated, which may be long before it has reached the root. This reduces the amount of communication needed for query execution. If the predicate is highly selective, the significant communication savings to the sensor network are very significant.

4.3 Data Pruning

In sensor networks, if the data readings at the sensors do not fluctuate much from epoch to epoch (such as temperature). Aggregate values may fluctuate even
less than the individual readings at sensors, such as in the case of average. When data records have values that are the same, or close, to previous records, we may want to avoid transmitting such records; we call this data pruning. Some strategies for data pruning this are TINA [SBL03], delta gathering [GSK03], and duplicate suppression [IGE02].

However, none of the data pruning strategies could be applied to aggregate values, because these values were not evaluated in-network. In the case of spatial aggregation, rivet information allows us to evaluate RARs in-network, and duplicate removal strategies can be applied to the evaluated records with any of the above strategies.

5 Computing Rivet Information

In this section, we discuss how to compute rivet information for a given routing tree, presenting two approaches: exact riveting and approximate riveting. While the exact approach is more effective (i.e., it allows more in-network processing), the approximate approach is easier to compute and maintain.

5.1 Exact Riveting

Clearly, for any region $R$, the deeper its rivet node $X_R$ is located, the larger the potential savings from identifying its RAR as complete when it reaches $X_R$. In the best case, $X_R$ is the least common ancestor (LCA) of the sensors that lie in $R$; this is the deepest node that satisfies definition 3.1. We call this case exact riveting:

**Exact rivet node of region $R$:** LCA of all the nodes that lie in $R$.

We now present an in-network LCA algorithm for computing exact riveting information. This is a new algorithm; distributed LCA algorithms can be found in the literature, but they are not appropriate in our routing tree context.

**Exact Riveting.** The algorithm traverses the sensor network bottom up; we assume all nodes know their region.

1. Each node $X$ with ID $X.id$, located in region $X.r$, sends its parent a message $\langle X.id, X.r \rangle$.
2. If $X$ internal, it checks messages of the form $\langle Y.id, Y.r \rangle$ coming from its children:
   (a) if $Y.r = X.r$, the message is suppressed;
   (b) otherwise, if $Y.r$ appears in only one message, it is passed up;
   (c) otherwise (two or more messages have the same region ID $Y.r$, where $Y.r \neq X.r$), $X$ suppresses them both, sending on a message $\langle X.id, Y.r \rangle$ instead.

At the end of this bottom-up pass, the root will receive one message for every region ID, indicating the sensor node that is its LCA. This information is disseminated back into the network to notify the nodes of their rivet regions.

5.2 Approximate Riveting

Note that this bottom-up traversal requires synchronization, i.e., nodes should wait for all messages from their children, so they can know how many of them have the same region ID. It also involves two passes (bottom-up and top-down), and has high maintenance costs. Next, we discuss a “cheaper”, approximate algorithm for computing rivet lists. The rivet nodes it finds may be closer to the root than LCAs, but they still allow us to apply the techniques in section 4 to achieve savings in communication.

Our approximate rivet algorithm assumes that we are keeping, at every sensor node, the bounding box (or the convex hull) for itself and its children:

**Approximate rivet node of region $R$:** the first node from bottom up, whose bounding box (or convex hull) encloses all bounding boxes that intersect $R$.

Sometimes, a non-empty intersection of $R$ and some bounding box may contain no sensor nodes. Thus, the approximate rivet node is a common ancestor but not necessarily the least.

Georouting trees are routing trees augmented with such bounding box or convex hull information [GSK03]. Georouting trees allow a significant savings in communication during geocasting, when localized spatial data (or query) is broadcast into the network. This is accomplished by using the bounding box information to route the data to those sensors that lie in the region of interest. We therefore assume that a georouting tree is being maintained in our network.

Approximate rivet lists are computed during region broadcasting, when the IDs and the spatial extents for the set of regions $\{R_1, \ldots, R_k\}$ broadcast through the sensor network. The computation is the same for all regions, in parallel; we describe it for a representative region $R$.

**Approximate Riveting.** Before broadcasting $R$, we associate with it a rivet flag $R.b$, initially set to FALSE.

1. As $R$ is propagated down the tree with a FALSE $R.b$, it will eventually reach some node $X$ such that either (a) $X$ is inside $R$, or (b) $X$ is outside $R$, but it has more than one child whose bounding box (or convex hull) intersects $R$. Figure 1 illustrates the possible cases.

2. At this point, $R.b$ is set to TRUE and $R$ is added to $X$’s rivet list (alternatively, $R.b$’s record can be suppressed if not needed for anything else).

Note that space limitations prevent us from presenting complexity or correctness analyses for the algorithms presented in this section and from discussing
the dynamic maintenance of rivet lists. This will be done in the full version of the paper.

6 Experimental Results

In our last section, we provide motivation for the rest of the paper by showing the potential savings from using rivet information. Clearly, each of the techniques for using rivet information, presented in section 4, will have its own performance characteristics. However, we can get a general sense of the potential savings by assuming that for each of these techniques, some percentage of records will be dropped after identified as complete. In the case of filtering predicates, for example, this percentage directly corresponds to the selectivity of the predicate.

We computed \( N_{\text{saved}} \), the number of record transmissions that would be saved by using rivet information if we applied a filtering predicate with selectivity 100%. \( N_{\text{saved}} \) is the sum, for all nodes \( X \) in the network, of depth\((X) \times \text{size(rivet-list}(X)) \). This is to establish a base line for comparison; in general, the number of transmissions saved would be some fraction of this number.

We compared \( N_{\text{saved}} \) to \( N_{\text{total}} \), the total number of transmissions that would be required if no rivet information were used (per epoch). \( N_{\text{total}} \) is the sum, for all nodes \( X \) in the network, of depth\((X) \). Sensor networks of different sizes were generated randomly; the number of regions in the set \( \{R_1, \ldots, R_k\} \) was adjusted at each trial, so that the average number of sensors per region remained constant. The ratios of \( N_{\text{saved}} \) to \( N_{\text{total}} \) was measured for each trial; figure 3 shows this ratio, plotted against network size.

![Potential Savings from Rivet Information](image)

Figure 3: Potential Savings from Rivet Information

This figure contains three different plots, corresponding to three different algorithms for computing rivet lists: exact, approximate with bounding boxes, and approximate with convex hulls (section 5). Not surprisingly, exact riveting is better than approximate, and convex hulls are better than bounding boxes.

It is clear that the effectiveness of using rivet lists increases as the network (and the number of regions) grows. As the number of regions in the network grows, more and more regions have their rivet nodes further and further from the root. These preliminary results show that rivet information can reduce in-network communication by up to 16%, without any drop in accuracy.

References


