University of Alberta

An Analysis of Join Processing in Sensor Networks

by

Alexandru Coman
Mario A. Nascimento
Jörg Sander

Technical Report TR 06-24
November 2006

DEPARTMENT OF COMPUTING SCIENCE
University of Alberta
Edmonton, Alberta, Canada

Abstract

Wireless sensor networks have received much attention recently. Given their autonomy, flexibility and large range of functionality, they can be used as an effective and discrete means for monitoring data in many domains. Typically the network autonomy implies a limited and relatively small amount of energy for its operation. Hence, an important challenge they pose is how to process queries, i.e., manage and communicate data, in an energy-efficient manner within the network. In this paper we consider the problem of how to process join queries in a wireless sensor network. Unlike other types of queries, join queries have received little attention in the literature, despite their importance. We propose a few strategies for processing join queries, focusing on where (which sensor node(s)) to process data, and investigate their performance across several scenarios. Not surprisingly, our experiments show that no single strategy can be considered competitive for all scenarios. In order to avoid the potential high cost of using a fixed strategy for processing all queries, we develop a cost-based model that can be used to select the best join strategy for the query at hand. Our results confirm that, given a set of queries, selecting the join strategy based on the cost model is always better than using any fixed strategy for all queries.

1 Introduction

Recent technological advances, decreasing production costs and increasing capabilities have made sensor networks suitable for many applications, including environmental monitoring, warehouse management and battlefield surveillance. Today's sensors are no longer just simple sensing devices wired to a central monitoring site, but they have computation, storage and wireless communication capabilities. Most of these devices are battery operated, which highly constrains their life-span. In addition, it is often not possible to replace the power source of thousands of sensors. Energy efficient data processing and networking protocols must be developed in order to make the long-term use of such devices possible. Hence, our focus is on energy efficient processing of queries over sensor networks.

Users query the sensor network to retrieve the collected data on the monitored environment. The most popular form for expressing queries in a sensor network is using a declarative language [23, 15] such as SQL. The data collected in the sensor

network, be it stored or collected on demand, can be seen as one relation distributed over the sensor nodes, called the sensor relation in the following. The queries typically accept one or more of the following operators [25]: selection, projection, union, grouping and aggregations. Continuous queries also allow special operators that specify the duration of the query [23, 15] and, sometimes, the frequency of sensing. The join operation was mostly avoided. We present two applications where joins are an important operation for satisfying the users' information need.

In this paper we focus on the processing of the join operator in sensor networks. Recently, a few works tackled some aspects of the join processing problem. Bonfi ls and Bonnet [2] consider the problem of placing a correlation operator (i.e., a special join) at a node in the network. Pandit and Gupta [16] propose two algorithms for processing a range-join operator in the network and Yu at al. [24] propose an algorithm for processing equi-joins. These works study the *self-join* problem where subsets of the sensor relation are joined. Abadi et al. [1] propose several solutions for the *external join* problem, where the sensor relation is joined with a relation stored at the user station. A third type of join is the *internal join* where the sensor relation is joined with relations stored locally at the nodes, such as historical statistics or pre-loaded relations.

Since the energy required by sensing and computation is three to four orders of magnitude less than the energy used for communication [15, 25], we are interested in minimizing the energy cost of communication during query processing. We study this problem in a peer-to-peer sensor network environment where each sensor node is only aware of the existence of the other sensor nodes located within its wireless communication range, and the query can be initiated (or introduced) locally at any sensor node.

1.1 Motivating Applications

Environmental Monitoring. National Parks administration is interested in long-term monitoring of the fauna and flora in the managed park. A sensor network is deployed over the park, with the task of monitoring various phenomena (e.g.: temperature and humidity) and well as observe the animals (sound, video or RFID sensing). Park rangers patrol the park and, upon observing certain patterns, they

Pattern	Query	Query type
An animal is found dead	What other animals have been in the surrounding region around the (estimated) time of death?	Historical spatiotemporal range query
Animals are found dead in several regions	What animals have been in all the regions around the (estimated) times of death?	Historical spatiotemporal range query with self-join
A forest patch is affected by an illness	What were the conditions in the region before the illness occurred?	Historical spatiotemporal range query
Several forest patches are affected by an illness	What were the conditions common in all the regions before the illness occurred?	Historical spatiotemporal range query with self-join
Certain conditions are discovered	What other regions have had these conditions?	Continuous/historical temporal range query with external join

Table 1: Sample queries for environmental monitoring

query the sensor network to find information of interest. For instance, upon finding two animals killed, the rangers need to find what animal, possibly ill of rabies, has killed them. The query could be *"What animal has been around both killed animals before the estimated time of death?"*. If joins cannot be processed in-network, then two, possibly long, lists of animals appearing in each region will be retrieved to be joined at the user station. On the other hand, if the join is processed in-network, only one animal ID is retrieved, saving the communication cost for retrieving all animals appearing in the two regions. Other instances of patterns, the respective queries and their type are summarized in Table 1.

Traffic Monitoring. The city police and administration are interested in extending the monitoring of the criminal activities over most of the city. As the cost of installing power and communication lines is very high, they are interested in an autonomic wireless solution. Each monitoring device consists of a camera for sensing, computation and storage, specialized hardware for image processing, equipment for wireless communication and a solar-rechargeable power source. The cameras could take snapshots periodically and store them locally. The collected snapshots would be automatically discarded after a while. The monitoring devices would form a distributed network. While the nodes of this sensor network have more capabilities than those used in the environmental monitoring application, they are still constrained on the energy resource available at any given time. Table 2 presents several queries and their type for this environment.

Our query examples cover several types of queries. We believe that real sensor network deployments will have multiple roles, ranging from long-term monitoring to event alerts. It is unlikely that a processing solution could perform best for all

Query	Query type
What vehicles passed area X when the accident/crime happened?	Historical spatiotemporal range query
Have any of the vehicles in the list passed area X last week?	Historical spatiotemporal range query with external join
Alert when any of the vehicles in a given list pass area X.	Continuous spatial range query with external join
What vehicles have passed area X on Sunday and area Y on Tuesday?	Continuous spatiotemporal range query with self-join
What vehicles have passed area X within 2 minutes from the moment the noise level exceeded the threshold?	Continuous/historical spatiotemporal range query with internal join

Table 2: Sample queries for traffic monitoring

query types. As the energy efficiency is of primary concern, each type of query should be optimized differently and several processing solutions will coexist in the network.

1.2 Our Contributions

In this paper, our contributions are three-fold:

- We analyze several strategies for processing join queries. We investigate their suitability for certain scenarios (i.e., a combination of query and network characteristics) and their performance under various conditions.
- We develop cost models to estimate the processing cost of each strategy. We use these models in a query optimizer to dynamically select the most energy efficient processing strategy for a given query and sensor network.
- In an extensive experimental evaluation we show that each processing strategy performs best under certain conditions. We also show that dynamic strategy selection by the model-based query optimizer outperforms every processing strategy alone. Moreover, the optimizer makes close to optimal strategy selections in most cases.

1.3 Organization

The remainder of this paper is organized as follows. Section 2 presents an overview of sensor networks, as well as the characteristics of the data. Section 3 details the problem statement and presents four solutions for processing the join operator in

the sensor network. We also build cost models for each of the presented solutions in this section. We discuss several optimizations and extensions in Section 4. The evaluation of the investigated solutions and a discussion of results is presented in Section 5. Section 6 describes some of the research work related to ours and Section 7 concludes the paper.

2 Preliminary Background

2.1 Sensor Networks

In spite of the relative novelty of the architecture and the small number of real-life deployments, sensor networks are considered a highly promising technology that will change the way we interact with our environment [19]. Typical sensor networks will be formed by hundreds to tens of thousands of small, radio-enabled, sensing nodes. Each node is capable of observing the environment, storing the observed values, processing them and sharing them through wireless communication. While most of these capabilities are expected to rapidly grow in the near future, the energy source, be it either a battery or some sort of energy harvesting [10, 17], will remain the main limitation of these devices due to the relatively slow progress in energy storage and harvesting technologies. Nowadays sensor nodes have a large variance in capabilities, ranging from the bulky and powerful Sensoria nodes [21] to the small but limited Mica Motes [5].

In our work we consider a sensor network formed by thousands of fixed nodes. Each node has several sensing units (e.g., temperature, humidity, RFID reader), a small processor, few megabytes of flash memory, a fixed-range wireless radio and it is battery operated. These characteristics encompass a wide range of sensor node hardware, which extends the applicability of our research. Further on, we consider that each node is aware of its location, which is periodically refreshed through GPS [6] or a localization algorithm [20] to account for any variation in a node's position. Each node is aware of the nodes located within its wireless range, which form its neighborhood. A node communicates with nodes other than its neighbors using multi-hop routing over the wireless network.

As sensor nodes are not designed for user interaction, users access the sensor

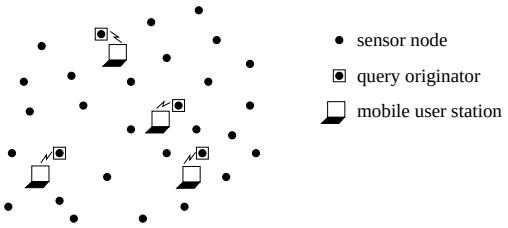


Figure 1: User interaction with the sensor network

network through user stations, which connect to one of the sensor nodes in their vicinity (Figure 1). The sensor node acts as a gateway in the sense that the node receives the user queries from the user station and returns the answer to it, without other nodes being aware of the user station. We call this node *query originator* in the following.

2.2 Data Model

Nodes acquire observations periodically and the observations are stored locally for future querying. The data stored in the sensor nodes forms a virtual relation over all sensors, denoted R^*. As nodes store the acquired data locally, each node holds the values of the observations recorded by its sensing units and the time when each recording was performed. On a conceptual level, the schema of the relation stored at a node N_j is $R_j(loc_j, time, s_1, \ldots, s_S)$[1], where loc_j denotes the location of the sensor node N_j, s_i is a recorded value for sensing unit i and S is the number of sensing units. The virtual relation R^* is the union of the node relations $R^* \leftarrow \bigcup R_j$, $j = 1..N$, where N represents the number of sensors nodes. Thus, the virtual relation R^* stores spatio-temporal data.

2.3 Query Model

In this paper we analyze the join processing problem in sensor networks for join queries having the sensor relation R^* as one of the joined relations, while the other relation could be R^* (*self join*), a relation stored at the user station (*external join*)

[1] An actual implementation may use a different organization. For instance, loc_j should be stored only once in N_j.

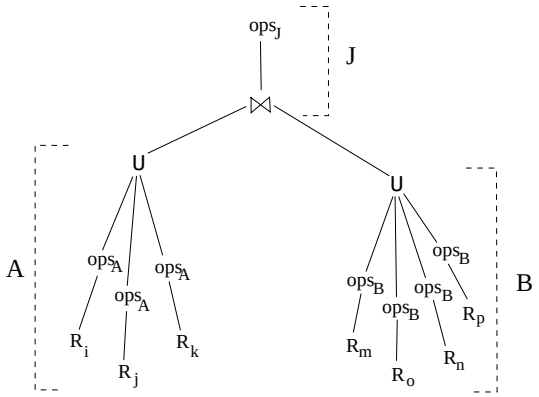


Figure 2: Query tree and notations

or a relation stored in the network (*internal join*). For clarity of presentation, we consider *self join* queries only in the following. In Section 4 we show that our analysis of the *self join* problem applies to the *external* and *internal joins* as well.

We impose no restrictions on the join conditions, that is, any tuple from a relation could match any tuple of the other relation. For each occurrence of the R^* relation we consider that the query contains a spatial selection predicate constraining the tuples of the relation to belong to a region[2]. For instance, the query *"What animals have been in both regions around the times of interest?"* from our example in Section 1 can be expressed in an SQL-like language as:

```
SELECT   S.animalID
FROM     R* as S, R* as T
WHERE    S.location IN Region A
    AND  T.location IN Region B
    AND  S.time IN TimeRange A
    AND  T.time IN TimeRange B
    AND  S.animalID = T.animalID
```

Let us denote with A the restriction of R^* to the sensor nodes in Region A and with B the restriction of R^* to the sensor nodes in Region B. In our presentation we will refer to the joined relations as A and B, but they are in fact restrictions of the R^* relation to the respective areas as specified in the query. The query may also

[2]The sensor relation R^* can be expressed as R^* constrained to the entire sensor network area.

contain other operators (selection, projection, etc.) on each tuple of R^* or the result of the join. Since our focus is on join processing, we consider the relations A and B as the resulting relations after the operators that can be applied on each node's relation have been applied. We consider operators that can be processed locally by each sensor node N_j on its stored relation R_j and thus they do not involve any communication. We denote with J the result of the join of relations A and B, including any operators on the join result required by the query: $J = ops_J(A \bowtie B)$. We consider operators on the join result can be processed in a pipelined fashion immediately following the join of two tuples. Figure 2 shows a general query tree and the notations we use.

3 Strategies for Processing Join Queries

3.1 Problem Statement

We are interested in evaluating the merits of several join processing strategies and deriving the conditions under which a strategy performs best. In our analysis we categorize the strategies based on the location where the join operation is carried and if a semi-join is used. We analyze the following processing strategies:

- **External Join**: the join is processed externally;
- **Internal Join**: the join is processed at the location of one of the joined relations;
- **Mediated Join**: the join is processed in the network at a location other than the locations of one of the joined relations;
- **Local Semi-Join**: the join is processed using a semi-join at the location of one of the joined relations;
- **Mediated Semi-Join**: the join is processed using a semi-join at a location other than the locations of one of the joined relations.

Before we proceed on discussing the join processing strategies, let us model the energy cost E of transmitting data in the sensor networks. These costs will be used

as building blocks for modelling the cost of each strategy. Let us first introduce the following definitions:

Definition 1: The distance between sensor nodes N_i and N_j in the sensor network area is the Euclidean distance between their locations L_i and L_j. We denote this distance with $d_{N_iN_j}$.•

Definition 2: The distance between a sensor node N_i and a region R_A is the average distance between N_i and a node in R_A: $d_{N_iR_A} = \frac{\int_{R_A} d_{N_iN_j} dR_A}{Area(R_A)}$.•

For the large scale sensor networks considered in our work, most queries will involve the sensor relation constrained to relatively small regions from the network. We approximate the location of a relation by the location of the centroid C_A of the region R_A where the relation is distributed. Thus, we have $d_{N_iR_A} = d_{N_iC_A}$.

Definition 3: The location L_A of a relation A distributed over the sensor nodes in an region R_A is the centroid C_A of the region R_A.•

Definition 4: The distance between two relations A and B in the sensor network area is the Euclidean distance between their locations L_A and L_B. We denote this distance with d_{AB}.•

The cost of transmitting data D from node N_i to node N_j using unicast multi-hop routing is directly proportional to the size of the data s_D, the energy cost to transmit (E_t) and receive (E_r) one bit of information over one hop and the number of hops between the two nodes $h_{N_iN_j}$:

$$E(N_i, N_j, s_D) = (E_t + E_r)\ s_D\ h_{N_iN_j}. \tag{1}$$

The number of hops is equal to the distance $d_{N_iN_j}$ between nodes N_i and N_j divided by the average advance towards destination over one hop a_{hop}. We denote with k_u the terms independent of N_i and N_j, $k_u = (E_t + E_r)/a_{hop}$. Note that k_u is independent of the query and it network specific. We have:

$$E(N_i, N_j, s_D) = k_u\ s_D\ d_{N_iN_j}. \tag{2}$$

To transmit relation A distributed over R_A to node N_j, we transmit the subset of A stored at the nodes in R_A to N_j. We have:

$$E(A, N_j, s_A) = \sum_{i=1}^{N_A} k_u \; s_{A_i} \; d_{N_i N_j}, \tag{3}$$

where N_A is the number of sensor nodes in region R_A. We approximate $d_{N_i N_j}$ with d_{AN_j} (see *Definition 2*) and we obtain:

$$E(A, N_j, s_A) = k_u \sum_{i=1}^{N_A} s_{A_i} \; d_{AN_j} = k_u \; s_A \; d_{AN_j}. \tag{4}$$

Finally, to transmit relation A to the nodes in R_B, we multicast relation A over region R_B: we unicast A to the centroid of the region R_B and distribute A from there over R_B using broadcasting. Each node in R_B transmits A once and receives it from every neighbor during broadcasting. Let ν be the average number of neighbors and N_B the number of nodes in R_B. Thus, we have:

$$E(A, B, s_A) = E(A, C_B, s_A) + (E_t + \nu E_r) \; s_A \; N_B. \tag{5}$$

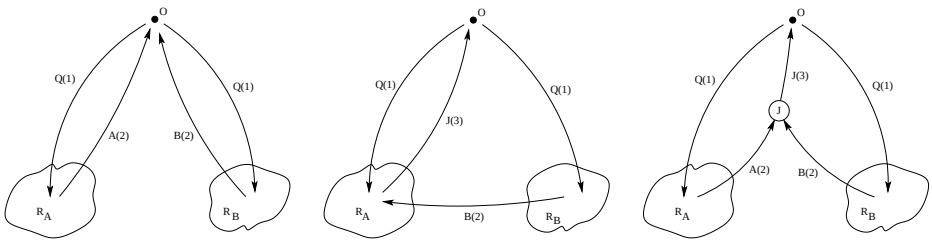
Let $k_b = (E_t + N_n E_r)$, which is independent of A and B. Note that k_b is independent of the query and network specifi c. Using *Definitions 3* and *4* we have:

$$E(A, B, s_A) = k_u \; s_A \; d_{AB} + k_b \; s_A \; N_B. \tag{6}$$

We will use this notations when estimating the cost of each join processing strategy.

3.2 External Join

Most query processing solutions proposed in the literature do not consider the join among their operators. Rather they focus on processing effi ciently the selection, projection and aggregation operators in the network, with the resulting data collected at the user station. For these solutions we can process a join by separately processing the query over the two relations, collecting the results (i.e., A and B) at the user station and performing the join externally. Figure 3(a) shows the data



(a) External Join (b) Local Join (c) Mediated Join

Figure 3: Join alternatives (w/o semi-join) - data flow and steps

flow (query Q and relation A and B) and the processing steps for this solution. The solution is a straightforward way for extending the current query processors to handle joins and it would require no (or very little) modifications to existing algorithms. We denote with O the location of the query originator node. The cost E_{Ext} of processing the join is equal to the sum of the costs of processing the two queries:

$$E_{Ext} = E(A, O, s_A) + E(B, O, s_B) = k_u s_A d_{AO} + k_u s_B d_{BO}. \quad (7)$$

The external join is advantageous when the size of the relation J resulting from the join is much larger than the sum of the two relations A and B or the data extracted from the sensor network is re-used for other tasks, such as external storing, other joins or building a model of the monitored environment. However, if the join selectivity factor is low (highly selective join), the network wastes energy for transmitting unnecessary tuples to the originator node and user station.

3.3 Local Join

An alternative for processing the join is transmitting one of the relations to the location of the other relation, performing the join locally and returning the join result to the originator node. At first, it may seem that it is advantageous to move the smaller relation to the location of the larger one. However, as discussed next, this may not be the most efficient case due to the cost of returning the join result to the originator node. Let us formally analyze the problem. When joining the two relations at the location of relation A, B is moved to the location of relation A, the

relations are joined locally at each node, and the result it transmitted to the query originator. The energy cost is:

$$E_{Loc}(A) = E(B, A, s_B) + E(A, O, s_J) = k_u s_B d_{AB} + k_b s_B N_A + k_u s_J d_{AO}. \quad (8)$$

Figure 3(b) shows the flow of data (query Q, relation B and join result J) and the steps of the *Local Join* solution. Similarly, when performing the join at the location of relation B we have:

$$E_{Loc}(B) = E(A, B, s_A) + E(B, O, s_J) = k_u s_A d_{AB} + k_b s_A N_B + k_u s_J d_{BO}. \quad (9)$$

When the relation located closer to the originator node is also larger in size, it is more efficient to process the join at its location to minimize the cost. Otherwise, the two costs must be estimated and compared to decide the best join location. We discuss in Section 4 how the originator node (or user station) may estimate the parameters used in the cost models and how the accuracy of the estimates affects the decision. For now, we assume these estimates are available. If the distances between the originator node and the two relations are large compared to the extent of each query region, the term expressing the cost of broadcasting in the query region can be neglected, thus further simplifying the cost model.

Various algorithms could be used for joining relations A and B at the location of A, respectively, B. However, this is beyond the scope of our analysis. In our evaluation, we consider the following algorithm for processing the join at the relation A: relation B is multicast to the nodes in the region R_A; each node performs locally the join between its local partition of the relation A and relation B and the resulting tuples are sent using geographic routing to the originator node. Note that the local joins can be performed in a distributed, pipelined fashion; as soon as a node in R_A receives a packet containing B's tuples it joins them with the local partition of A and can send the result. Thus, only two buffers for the received and outgoing packets are required at each node to process the join. The join is processed at the location of B similarly. We do not include the cost of the local processing in the total cost as it does not involve communication.

3.4 Mediated Join

A third alternative for processing the join is performing the join at a location different than the location of the originator node or the locations of the involved relations. To process the join, relations A and B are collected at location R_J where they are joined and the resulting relation J is transmitted to the originator node. Figure 3(c) shows the data flow and the order of the processing steps. The cost of processing the join at the intermediate location R_J is:

$$E_{Med} = E(A, J, s_A) + E(B, J, s_B) + E(J, O, s_J) \quad (10)$$

$$= k_u s_A d_{AJ} + k_u s_B d_{BJ} + k_u s_J d_{JO}. \quad (11)$$

Note that the *External Join* is in fact an instance of the *Mediated Join* where locations R_J and O coincide. In the general case, the challenge is to find the optimal position for the join location such that the cost of processing the join is minimized. We need to locate the optimal R_J such that it minimizes the weighted sum of the distances from R_J to A, B and O, where the weights are the sizes of the data involved in the join. This problem is known as the the weighted Fermat[3] problem, where one wants to find the point with the property that the weighted sum of the distances from the point to the vertexes of a triangle is minimized. To find the optimal join location, we use the solution proposed by Greenberg and Robertello [8]. The main points of the solution are:

- The locations of A, B and O form a triangle where each location has assigned a weight equal to the amount of data it sends (s_A or s_B) or receives (s_J).
- If the weight at a location is greater than the sum of the weights at the other locations, then the join should be processed at that location.
- If the weights are equal and one of the angles of the triangle is larger than $2\pi/3$, the join location is at the vertex where the angle occurs.

[3]This problem is also know as the three factory problem, the three villages problem and the weighted Steiner problem. Steiner has analyzed it in a general context involving three or more locations.

- Otherwise, the location R_J lies in the triangle. The derivation of the optimal location involves non-trivial trigonometry and analytical geometry, but the terms expressing the optimal join location are computationally inexpensive. For further details see [8].

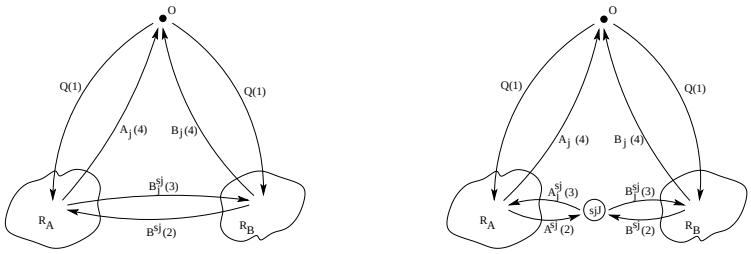
In [4, 16] the authors also investigate finding the optimal join location for this scenario. They consider that the optimal join location is the weighted centroid of the triangle formed by A, B and O. The centroid has the property that it minimizes the weighted sum of the *squared* distances, and thus it is not optimal.

Once the optimal location R_J is known, we need an algorithm to process the join. Pandit et al. propose an index-based and a hash-based join algorithm for processing range-join queries [16]. Our work is general with respect of the join condition and thus their algorithms are not suitable for our problem. In addition, we are interested in analyzing the alternative join locations rather than studying various algorithms for a particular join location.

In our evaluation, we use a simple block-nested-loop-based algorithm for processing the join at the intermediate location, which works as follows. As one of the join relations must be stored at the join location to process the join, we use the smaller of the two relation for this purpose. Let us assume that the smaller relation is A. If A fits into one node, then we store it at the node located closest to location R_J. Otherwise, A is stored at the nodes located in region R_J around the optimal location. Each node in R_J stores a partition of A. Next, B is multicast to the nodes in R_J, and each node locally joins its partition of A with B. Again, the processing proceeds in a distributed, pipelined fashion. Once a packet of joined tuples is created or the join finishes, the joined tuples are transmitted to the query originator. As R_J is much smaller than R_A, the cost of distributing A at the nodes in R_J is negligible and it is not included for simplicity in Equation 11.

3.5 Join Processing with Semi-Joins

For join conditions with low selectivity it is often the case that many tuples of one relation will not match any tuple of the other relation. Since transporting the tuples over the network is costly, one wants to avoid transporting tuples that do not join. A technique commonly used in distributed databases for reducing the cost of moving



(a) Local Semi-Join (b) Mediated Semi-Join

Figure 4: Join alternatives with semi-join - data flow

non-matching tuples is the semi-join [12]. In a semi-join, for each tuple only the attributes appearing in the join condition together with a tuple identifier are used for evaluating the join. Only this subset of a relation must be transported over the network to evaluate the join. Once the join is evaluated, the tuple identifiers for the joined tuples are returned to the original relation. The full tuples for the joined tuples are then transmitted to the join location or the query result destination. The semi-join technique assumes that the size of the sub-set of attributes transmitted plus the size of the identifiers for the joined tuples is much smaller than the size of the original relation.

As most sensor nodes have several sensing units, sensor tuples tend to have a large number of attributes. If the join condition involves only some of these attributes, it may be more cost efficient to employ the semi-join technique when processing joins over the sensor data. In the following we discuss how semi-joins can be used with local and mediated join processing.

3.5.1 Local Semi-Join

We consider first the case when the join is performed locally at the relation A. If the selectivity of the join condition is low, sending the entire relation B at R_A may be unnecessary and expensive. When using semi-joins, part of each tuple of B is sent to R_A, where it is joined with the tuples of A as for the *Local Join*. For each semi-tuple of B matching a tuple of A, its identifier is returned to B. To obtain the query result at the query originator, the entire tuples of the matching semi-tuple of B must reach the originator. It is more efficient to send these entire tuples

directly to the originator node than through A. After the semi-join has been fully processed at R_A, the tuples of A that have joined one or more of the semi-tuples of B are also sent to the query originator. Once the joined tuples from both A and B have reached the query originator, they can be joined externally at the user station. Figure 4(a) shows the data flow and the processing steps. The cost of the processing is:

$$\begin{aligned} E_{sjLoc}(A) &= E(B, A, s_{B^{sj}}) + E(A, B, s_{B_j^{sj}}) + E(A, O, s_{A_j}) + && (12)\\ & \quad E(B, O, s_{B_j}) && (13)\\ &= k_u s_{B^{sj}} d_{AB} + k_b s_{B^{sj}} N_A + k_u s_{B_j^{sj}} d_{AB} + k_b s_{B_j^{sj}} N_B + && (14)\\ & \quad k_u s_{A_j} d_{AO} + k_u s_{B_j} d_{BO} && (15) \end{aligned}$$

where B^{sj} represents the partition of B required for the semi-join, B_j^{sj} represents the tuple identifiers for the tuples of B^{sj} joined with tuples of A, and A_j and B_j are the tuples of A, respectively B, that joined during the semi-join. Similarly, if the join is performed at B, the processing cost is:

$$\begin{aligned} E_{sjLoc}(B) &= E(A, B, s_{A^{sj}}) + E(B, A, s_{A_j^{sj}}) + E(B, O, s_{B_j}) + && (16)\\ & \quad E(A, O, s_{A_j}) && (17)\\ &= k_u s_{A^{sj}} d_{AB} + k_b s_{A^{sj}} N_B + k_u s_{A_j^{sj}} d_{AB} + k_b s_{A_j^{sj}} N_A + && (18)\\ & \quad k_u s_{B_j} d_{BO} + k_u s_{A_j} d_{AO} && (19) \end{aligned}$$

Note that the difference in cost between $E_{sjLoc}(A)$ and $E_{sjLoc}(B)$ is given by the semi-join part of the cost, as the cost of sending matching tuples to the query originator is the same. As the size of the tuple identifiers for the semi-joined tuples is much smaller than the semi-join partitions, the cost difference is determined mostly by A^{sj} and B^{sj}. If A^{sj} is larger than B^{sj}, then A should be the semi-join region, and B should be the semi-join region otherwise.

3.5.2 Mediated Semi-Join

In this approach both relations A and B send semi-tuples to an intermediate location R_J where these tuples are joined. Once the semi-tuples are joined, the identi-

fiers of the tuples participating in the join result are returned to the locations of the joined relations. Then the nodes send the tuples contributing to the join result to the query originator, where they are joined again to generate the query result. Figure 4(b) shows the data flow and the processing steps. The cost of the processing is:

$$
\begin{aligned}
E_{sjMed} &= E(A, J, s_{A^{sj}}) + E(B, J, s_{B^{sj}}) + E(J, A, s_{A_j^{sj}}) + && (20)\\
&\quad E(J, B, s_{B_j^{sj}}) + E(A, O, s_{A_j}) + E(B, O, s_{B_j}) && (21)\\
&= k_u s_{A^{sj}} d_{AJ} + k_u s_{B^{sj}} d_{BJ} + k_u s_{A_j^{sj}} d_{AJ} + && (22)\\
&\quad k_u s_{B_j^{sj}} d_{BJ} + s_{B_j} d_{BO} + s_{A_j} d_{AO}. && (23)
\end{aligned}
$$

To obtain the optimal join location that minimizes the cost of processing we need to minimize E_{sjMed}. The costs of sending the joined tuples from A and B to O is independent of the join location. Thus, the cost we need to minimize is:

$$
\texttt{min}(E_{sjMed}) = \texttt{min}(s_{A^{sj}} d_{AJ} + s_{B^{sj}} d_{BJ} + s_{A_j^{sj}} d_{AJ} + s_{B_j^{sj}} d_{BJ}) \quad (24)
$$

As only locations R_A and R_B are involved, it is easy to see that the optimal join location is on the line between A and B. Thus, we have that $d_{AJ} + d_{BJ} = d_{AB}$. We obtain that the join location should be at R_A if $s_{A^{sj}} + s_{A_j^{sj}} \geq s_{B^{sj}} + s_{B_j^{sj}}$ and the join location should be R_B otherwise. Since the optimal join location is either A or B, this approach is similar to the *Local Semi-Join* approach.

4 Discussion

In this section we discuss some issues that are relevant to our work, namely: performing the join at a mediated location; the estimation of the join selectivity; reliability of routing; region approximation; and processing external and internal joins.

Mediated Join. For the *Mediated Join* solution we have presented the problem in terms of finding the best location for performing the join. A first issue is that nodes are located at discrete location and there may be no node located at the best location. This issue is trivially solved by performing the join at the closest node to the best location. The nodes located in regions R_A and R_B do not actually need

to know the location of this node as the geographic routing algorithm [11, 3] will route the packets destined for the best join location to the nearest located node. A second issue is the amount of storage the node performing the join has available. If the relations to be joined are small (or at least one of them), the node may store locally the smaller relation and perform a block-nested loop-like join in a pipelined fashion, in which case only very little buffer space is required for the second relation and the join result. However, if the relations to be joined are large, more nodes must participate in the join processing, and the node neighboring the best join node will also participate in the join processing. Algorithms for performing the join in such cases are beyond the scope of this paper, and we refer the reader to [16] for two such algorithms. Even when the size of the join relations is sufficiently small so that the best join node may perform the join alone, it may be useful the employ the node's neighbors to improve the reliability for highly unreliable networks. For instance, the smaller relation could be stored not only at the best join node, but at the next few nodes farther away from the best join location. If the best join node fails during query processing, the geographic routing will automatically route the data packets from the larger relation to the next closest node to the join location.

Estimation of Join Selectivity. Accurate estimation of join selectivity is important for any query processor as the query optimizer uses the estimate to choose the most cost-effective processing plan. For our problem in particular, the estimation errors may lead to using an expensive solution for processing the query, which, in turns, would reduce unnecessary the network lifetime. The cost of obtaining the estimation itself must also be considered, and it is typically a trade-off of estimation accuracy. In our context, a communication-free solution is using past query answers to estimate the join cardinality for new queries, but it may not be very accurate. A more accurate solution is using samples of the query relevant data, at the added cost of transferring these samples to the query originator node or user station. End-biased samples [7] is a particularly attractive solution as it provides highly accurate estimations with small sample sizes for correlated data, a typical characteristic of sensor data. In any case, the cost of estimating the join selectivity is very low compared to the cost of query processing, considering that very few data must be communicated for the estimation.

Reliability. In this work no routing tree is built and maintained, but rather ge-

ographic routing is used for routing data. This effectively means that every data packet is sent to a destination location rather than a specific node and data stops at the nearest located node. In addition, data sent from node N_i to node N_j could follow a different route than data send back from N_j to N_i as no routes are maintained. This approach alleviates the network reliability issue in part as a node on the route from N_i to a destination could die, but another route to the same destination will be discovered and used when the next data packet is sent. We employ GPSR/GFG [11, 3] for geographic routing which guarantee packet delivery if a route exist. A heartbeat technique [14] ensures that the neighbor lists are updated regularly to account for transient or permanent node failures.

Region Approximation. In this work we have assumed that the query regions are much smaller than the distance between the regions and the originator. As such, we approximated each region by the location of its centroid when building the cost model. For larger query regions or small distances to the originator node such an approximation may not be sufficient. Using more realistic approximations of the query regions and their effect on join processing and the the cost model is an open problem.

External and Internal Joins. In out presentation we have focused on processing the join between the data located a two regions in the network area. Nevertheless, we analyzed four general techniques that should apply equally well if only one of the relations is a subset of the sensor relation R^*. The other relation could be located *externally* at the user station or *internally* at one or more of the sensor nodes. Let us consider that relation A is the subset of the sensor relation R^* and B is the external or internal relation. If B is an external relation, we have that $d_{BO} = 0$. Fitting this case into the cost models it is easy to see that the external relation B should be moved in the network and the join should be performed at the location of relation A if the size of A is larger than the size of the external relation B plus the join result J. REED [1] discusses several situations for joining the sensor relation with an external relation. In the case of the join with an internal relation stored at the sensor nodes (different than the sensor relation R^*), we have two cases. If the subset A of the sensor relation and the internal relation are located at the same set of nodes, the join can be performed in the common region. Even more, if the join involves equality conditions on the spatial attribute, the join is trivially performed

locally at each node and no data needs to be communicated during the processing (except for the join result). If the subset A of the sensor relation and the internal relation are located in different regions, the processing is similar to the one analyzed in this paper.

5 Evaluation

We implemented a sensor network simulator in C++ to study the performance of the solutions and evaluate the cost models. The sensor nodes' placement follows a uniform distribution over a two dimensional region. The query consists of a join operation over the data from two distinct query regions from the network area. We express the size of the query regions as a percentage of the size of the monitored region. The query originator is one of the sensor nodes selected at random and the query regions are distributed at random in the network area. A summary of query and sensor network parameters and their default values used in our evaluations are presented in Table 3.

A parameter particularly important in the evaluation is the ratio of the sizes of relations A, B and J. We consider that the query selects a constant number of tuples from each relevant node's database (e.g., a temporal selection for a constant size interval). At first it may seem that this setting results in relations A and B having the same size due to the uniform node distribution. This is true only in average. Otherwise, for a particular query with default sizes of query regions, the ratio of the sizes of the relations A and B is up to 3 due to the small size of the query regions and the sparseness of the nodes in the network area. The default value for the selectivity of the join operator is 0.001 which results in the size of the join relation J being close to the sum of the sizes of relation A and B for the default query parameters. We further detail this aspect when we discuss the impact of the join selectivity factor on the performance of the solutions.

We compare the solutions in terms of the average energy used per network node for communication while processing a query. According to [18], the energy used to transmit and receive one bit of information in wireless communication is given by $E_t = \alpha + \gamma \times d^n$ and $E_r = \beta$, where d is the distance to which a bit is being transmitted, n is the path loss index, α and β capture the energy dissipated by

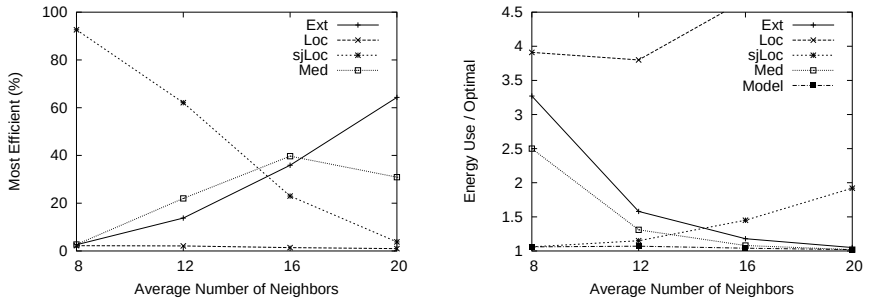
Parameter	Value (default)
Network area	1000x1000
Wireless range	50
Number of neighbors	12 (1650 nodes)
Link quality (%)	100
Size of each query region	0.5% (70x70)
Number of tuples per node	100
Number of attributes per tuple	6
Join selectivity factor(JSF)	0.001
JSF estimation error	0
Number of attributes per join tuple	6
Number of attributes per semi-join tuple	2

Table 3: Network and query parameters

the communication electronics and γ represents the energy radiated by the power-amp. In our experiments, we use the following values for these parameters [9]: $\alpha = \beta = 50nJ/bit$, $n = 2$, and $\gamma = 10\ pJ/bit/m^2$. All measurements are averaged over 100 randomly generated sensor networks, with 10 random queries over each network. We focus on the energy efficiency of the query processing solutions only, making the measurements independent of the characteristics of the MAC layer (for instance 802.11 radios consume as much energy in idle mode as for receive mode, while other radios may switch to a low-energy state when idle).

We evaluate the performance of the *External Join* (Ext), *Local Join* (Loc), *Mediated Join* (Med) and *Local Semi-Join* (sjLoc) solutions. In addition, we evaluate the cost of the *Model-Based Join* solution (Model) that uses our simple cost models to choose and execute the most cost-efficient solution among the four. We compare the cost of the investigated solutions against an *Optimal Join* solution (Optimal) that would process every query using the most efficient of the four solutions.

We evaluate the impact of several parameters on the performance of the solutions. Two of the parameters are characteristics of the sensor network: the network density and the packet delivery success rate. We also investigate the effect of two query characteristics on the performance of the algorithms: the size of the query's spatial range and the selectivity of the join operator.

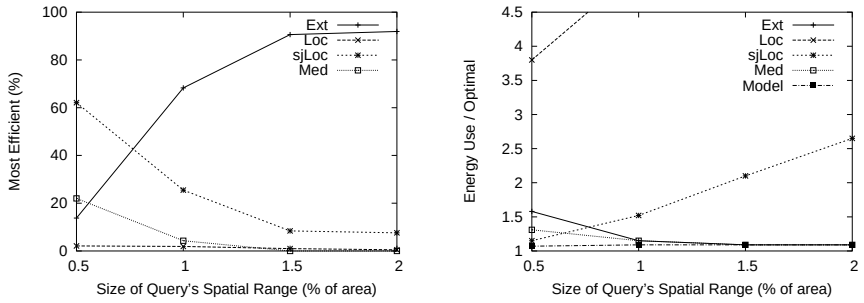


(a) (b)

Figure 5: Number of Neighbors

5.1 Impact of Network Density

We investigate first the effect of network density on the performance of the join processing solutions. Figure 5(a) shows the percentage of queries for which each solution performs best. The *Local Semi-Join* processing performs substantially better than the other solutions when the network density is small. When the network density increases, the *External Join* and *Mediated Join* perform better than the *Local Semi-Join*. For very dense networks with 20 or more neighbors per node, the *External Join* becomes the most efficient one for a majority of queries. This is due to the larger number of nodes in the query regions. As more data participates in the join and more join tuples are generated, it becomes more efficient to send the relevant data to the user station and process the join there. In addition, in the case of the *Local Join* and *Local Semi-Join* solutions, the cost of distributing the semi-join tuples to the nodes in the query regions increases substantially for higher network density. This effect can be better seen in Figure 5(b), where we show the cost ratio of each processing solution against the cost of the *Optimal Join*. In spite of the simple cost models, the *Model-Based Join* performs best for all network densities, choosing a solution close or equal to the most efficient one for processing the join. In fact, the cost of the *Most-Based Join* solution is within 7% of the cost of the *Optimal Join* for all network densities, while the *External Join* performs up to 327% worse, the *Median Join* up to 250% worse and the *Local Semi-Join* up to 192% worse than the cost of the *Optimal Join*. The *Local Join* performs poorly for all network densities, in average between 380% and 602% worse than the cost of



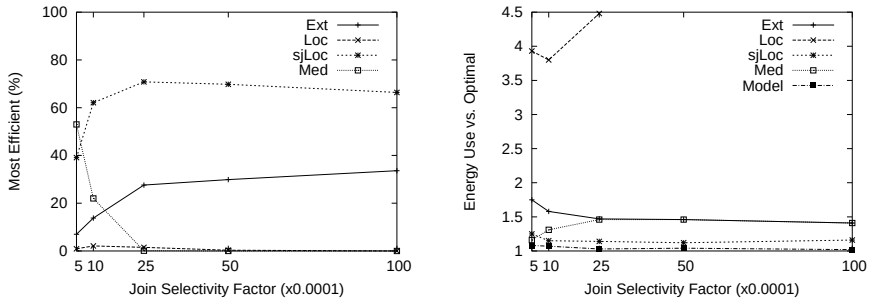
(a) (b)

Figure 6: Size of Query's Spatial Range

the *Optimal Join*, performing best for less than 2% of the queries (Figure 5(a)).

5.2 Impact of Size of Query Regions

We varied the size of the query regions between 0.5% and 2% of the network area for each query region (Figure 6). The increase in the size of query's range has a strong effect on the performance of the algorithms. For queries with regions larger than 1% of the network area the *External Join* performs best for a majority of queries (Figure 6(a)). The *Model-Based Join* solution outperforms all solutions for all query sizes (Figure 6(b)), but for query regions of 1.5% or larger its cost is very close to the cost of the *External Join* solution. As the *External Join* performs best for most queries with query regions larger than 1.5% of the network area and the *Model-Based Join* solution captures this behavior, the performance of the two solutions converge. The cost of the *Mediated Join* converges as well to the same value, since the best mediator position approaches or matches the position of the originator node for large query regions. The increase in the size of the query regions causes more nodes to be relevant to the query, and thus the amount of data that participates in and is generated by the join increases. Therefore it becomes more efficient to send the data to the user station over the shortest path (as in the *External Join* solution) compared to moving it in the network over longer paths. In addition, as we do not vary the the join selectivity factor in this experiment, the query reaches a point where the size of the data resulting after the join becomes larger than the participating relations. For our default join selectivity ratio (0.001),



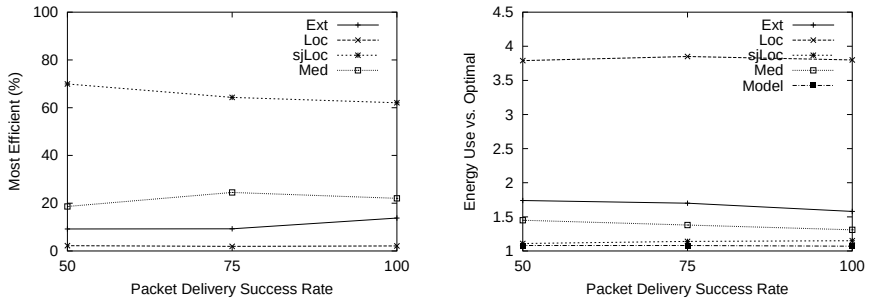
(a) (b)

Figure 7: Join Selectivity Factor

this effect occurs when each query region covers 20 or more nodes. It is easy to see that in this situation sending the data to the user station is the best solution, behavior well captured by the *Mediated Join* which moves the join location at the originator node.

5.3 Impact of Join Selectivity Factor

To evaluate the effect of the join selectivity factor on the performance of the solutions, we varied the factor between 0.0005 and 0.01. The lower range corresponds to 5 tuples being generated by the join of the data from each pair of nodes from the two query regions, while the higher range corresponds to 100 tuples being generated for each pair of nodes. Considering our default size of the query regions and network density, this effectively translates overall into the size of the join result being smaller than the joined data for the lower join selectivity factor (highly selective join condition), and larger for the higher selectivity factor. Also note that the join selectivity factor has no effect on the cost of the *External Join* as the join is performed at the user station, and only a small effect on the cost of the *Local Semi-Join* as the joining tuples from each query region are joined at the user station as well. The solution most affected by the size of the join selectivity factor is the *Local Join* since the join is performed at one of the query regions and the join data is then transferred to the user station. While the *Mediated Join* also performs the join at a location in the network, the mediated location is dynamically selected at query time and it coincides for some queries with the location of the



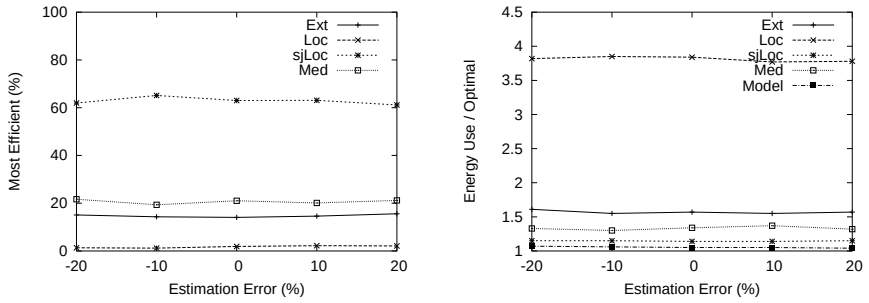
(a) (b)

Figure 8: Packet Delivery Success Rate

query originator when the size of the join data grows larger. When this effect occurs the *Mediated Join* and the *External Join* behave similarly (Figure 7(b)), and in the efficiency graph (Figure 7(a)) we consider that the *External Join* is the most efficient solution between the two. The experimental evaluation also shows that the *Mediated Join* performs best for small selectivity factors, closely followed by the *Local Semi-Join*. With the increase in the join selectivity factor (less selective join condition), the *Mediated Join* approached the *External Join* in behavior and performance, and the *Local Semi-Join* is only slightly affected. The *Model-Based Join* solution is able to pick these effects on the solutions, performing best and within 8% of the cost of the *Optimal Join* for all join selectivity factor sizes. The performance of the *Local Semi-Join* decreases slowly for increasing selectivity sizes, and in our setup it is the most efficient of the four solutions for a majority of queries when the size of the join selectivity factor is larger than 0.001.

5.4 Impact of Realistic Communications

Up to this point in our experiments we have considered a reliable communication environment, where no messages (packets) are lost during transmission. This assumption allowed us to investigate the performance of the solutions independent of the communication environment. Unfortunately, the typical environments where sensor networks operate affect the quality of transmission negatively, with packet delivery failing at times. To capture this unreliable conditions, we have set each communication link between two nodes with a packet delivery success rate. In



(a) (b)

Figure 9: Join Selectivity Factor - Estimation Error

addition, the quality of communication links is typically not symmetric, so we varied the delivery rate with up to 10% for the two directions of each communication link. When a packet does not reach its destination in a one hop transmission, the source node re-transmits it until it receives acknowledgement of receival. Figure 8 represents the performance of each solution under three packet delivery success rates. Each success rate represents the lower bound in terms of success rate for the links between two nodes located within the wireless communication range of each other, while the higher bound is the 100% delivery (no packet loss). The delivery success rates are randomly distributed in this interval. While the energy cost of each solution increases for decreasing delivery rates, the relative performance of the solutions remains unchanged. This suggests that the relative performance of the solutions is not affected by unreliable communication mediums and the cost model can be used to capture the relative performance of the solutions for unreliable communication environments as well.

5.5 Impact of the Estimation Accuracy for the Join Selectivity Factor

The cost models used in all solutions, except the *External Join*, use the join selectivity factor to estimate the size of the resulting join relation. In our previous experiments we have considered that this factor is estimated correctly to reduce the influence of its error (if any) on the effects of the other parameters. In this experiment we consider that the estimation of the factor is not accurate, and we investigate the effect that the estimation error has on the performance of the so-

lutions. We consider both the underestimation and overestimation errors, varying the error of the estimated join selectivity factor from -20% to $+20\%$ from the actual factor. Figure 9 shows the effect of this variation on the investigated solutions. While quantitatively the performance of the solutions varies slightly from the performance for accurate estimation, the relative performance of the investigated solutions remains the same in spite of the estimation error.

5.6 Impact of the Location of the Query Regions and Originator Node

Our experiments so far have assumed a setup where the query originator could be located anywhere in the network and there is no restriction on the location of the query regions. We have investigated the performance of the solutions on three more setups. In one of these setups the query originator is located in the upper-left corner of the network area and the query regions can be anywhere (a typical setup when using one fixed base-station). In the other two setups considered, one with a randomly distributed originator and the other with a corner originator, we restricted the locations of the query regions so that they are far away from each other and the originator. The experimental evaluation on these three setups has shown qualitatively similar behaviors of the solutions as for the setup discusses in details above. This suggests that the relative performance of the solutions is not affected by the relative locations of the query originator and query regions and that the cost models, while simple, are sufficient for capturing the relative performance of the algorithms in a variety of application scenarios.

5.7 Summary

Overall, the evaluation shows that no join processing solution performs best for all queries. The *Local Semi-Join* is especially suitable when the query regions are small and the network density is low. The *External Join* performs best for dense networks and large query regions, while the *Local Join* does not perform well under any investigated condition. The *Mediated Join* adapts well to the query characteristics and it is a good alternative to the *External Join* and *Local Semi-Join* solutions if performing the join at the user station is not acceptable. In any case, as energy is a vital resource of the network, one should not settle to use only one

solution for processing all queries, but rather select for each query the best solution. We have shown that using simple cost models to capture the relative performance of the investigated solutions is an effective way of selecting an efficient solution for each query, and it outperforms by a large margin processing all queries with the same solution.

6 Related Work

Over the past few year, much research has focused on how to minimize the cost of processing queries involving the sensor relation [14, 23, 22, 13]. The queries typically accept one or more of the following operators: selection, projection, union, grouping and aggregations. In addition, the continuous queries allow special operators that specify the duration of the query [23, 15] and, sometimes, the frequency of sensing. The join operation was mostly avoided either due to its complex nature or due to the belief that it is of little importance in a sensor network. Recently, several works tackled some aspects of the join processing problem.

Bonfils and Bonnet [2] consider the problem of processing a correlation operator (i.e., a special join) at a node in the network. The solution starts with a random placement of the operator at a network node. The position is progressively refined by moving the operator to the nodes with lower processing cost during the lifetime of the continuous query. Two important assumptions are that the operator can be fully processed on one node and that the lifetime of the query is sufficiently long to refine the operator position from a random location to an optimal one. An advantage of the refinement is that the operator placements adapts to the change is data during the query lifetime. For short continuous queries their solution would perform much worse than the optimal cost due to the initial random placement, while the solution is inadequate for historical queries.

Chowdhary and Gupta [4] propose an algorithm for performing the *self-join* in-network over a processing region with several sensor nodes participating in the join. The processing algorithm, called distribute-broadcast join, is a form of distributed block-nested loop join, where each node in the join area holds one block of the outside relation while the inside relation is broadcasted over the join region. The algorithm is similar in spirit with the algorithm we use for the *Mediated Join*.

Different from us, the authors consider a special shape for the join region and argue that this region is optimal. Along the same line, Pandit and Gupta [16] propose two algorithms for in-network processing of the range-join operator. One algorithm is a distributed form of a hash-join algorithm, while the other is a distributed form of index-join and uses a B-tree structure distributed at the sensor nodes. Both works [4, 16] consider that the optimal join location is the weighted centroid of the triangle. The centroid has the property that it minimizes the weighted sum of the *squared* distances, and thus it is not optimal.

Yu et al. [24] also propose a strategy for processing *self-join* queries in sensor network. Their solution is targeted for processing equi-join queries over historical data stored at nodes. The solutions constructs a synopsis (e.g., a histogram) of each relation involved in the join. The synopsis are transmitted to a mediated location, where they are used for finding which tuples of the two relation will certainly not join. This information is returned to the sensor nodes storing the relation, which will use will it to select only the join relevant tuples to participate in the join. The join is then performed in network at a mediated location. The solution performs best when the join selectivity is small and it is closest in spirit to our *Mediated Semi-Join*.

The *external join* problem where the sensor relation is joined with a relation stored at the user station is studied by Abadi et al. [1]. The external relation is basically a relation containing filters to be applied on the sensor tuples. If the external relation is small, it is flooded in the network and the join occurs locally at each node. When the external relation is too large to be stored in the network, the authors propose three techniques (bloom filters, partial joins and cache diffusion) that help filter part of the sensor tuples. Non-filtered tuples are then join externally after reaching the base station. An intermediate situation is when the external relation fits into a group of nodes. While multiple groups of nodes a formed in REED, this solution is closest in spirit with *Local Join*.

A research area closely related to our problem is distributed query processing in traditional database systems. Kossmann [12] surveys the state-of-the-art in the area. Some major differences for our problem is that in a sensor network the database relations are distributed over devices with limited capabilities and hard constrains on the energy resources, that the amount of information available at

each sensor node about the other partitions of the data (data location, distribution of values, etc.) is limited (if not null), and that the nodes communicate wirelessly, with all the issues this communicating environment brings into play. Nevertheless, there are also a number of similarities: the query processor needs to select the partitions of the relation that must be used to construct the query answer; the query optimizer must choose the location (server, node or neighborhood) where each operator needs to be processed; and the optimizer has to use cost models to choose the best plan and estimations affect these models.

7 Conclusions and Future Directions

While the technological advances have lead to sensors with reduced sizes and increased capabilities, the sensor data management is still in its incipient stages. Many works have focused on processing queries over the sensor network, but they limited their focus on processing the selection and aggregation operators over the sensor relation R^*. Such queries were mostly filtering the sensor data in-network, with any further processing done off-line. In this paper we have argued that the join operator allows one to pose important queries on the sensor data. We have also shown that the join operator can be pushed in-network together with the other operators previously studied. We analyze several solutions for in-network processing of the join between two relations, where at least one of the relations is the sensor relation. We show empirically that no join processing solution performs best for all queries. Using our cost models to choose at query time the most efficient processing solution, we are able to reduce the cost of join processing with up to 83% compared to processing every query with the same solution and perform within 7% of the optimal processing cost.

In this paper we have investigated the processing of queries with one join operation. An interesting open problem is in-network processing of queries joining multiple sensor relation. The challenge is finding the best order for joining the relations and, for each join, the most energy efficient processing solution. Any decision on what solution to use for processing a particular join operator in the query tree should consider its effect on processing of the other join operators in the tree.

References

[1] D. Abadi, S. Madden, and W. Lindner. REED: robust, efficient filtering and event detection in sensor networks. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, pages 769–780, 2005.

[2] B.J. Bonfils and P. Bonnet. Adaptive and decentralized operator placement for in-network query processing. In *Proceedings of the International Conference on Information Processing in Sensor Networks (IPSN)*, pages 47–62, 2003.

[3] P. Bose, P. Morin, I. Stojmenovic, and J. Urrutia. Routing with guaranteed delivery in ad-hoc wireless networks. *Wireless Networks*, 7(6):609–616, 2001.

[4] V. Chowdhary and H. Gupta. Communication-efficient implementation of join in sensor networks. In *Proceedings of the 10th International Conference on Database Systems for Advanced Applications (DASFAA)*, pages 447–460, 2005.

[5] Crossbow Technology Inc. MICA sensor platform. www.xbow.com.

[6] P.H. Dana. Global positioning system overview. The Geographer's Craft Project, University of Colorado at Boulder. Available at http://www.colorado.edu/geography/gcraft/notes/gps/gps.html, 1994-2000.

[7] C. Estan and J.F. Naughton. End-biased samples for join cardinality estimation. In *Proceedings of the 22nd International Conference on Data Engineering (ICDE)*, pages 20–31, 2006.

[8] I. Greenberg and R.A. Robertello. The three factory problem. *Mathematics Magazine*, 38(2):67–72, 1965.

[9] W. Heinzelman. *Application-Specific Protocol Architectures for Wireless Networks*. PhD thesis, MIT, 2000. http://www-mtl.mit.edu/research/icsystems/uamps/pubs/.

[10] X. Jiang, J. Polastre, and D. Culler. Perpetual environmentally powered sensor networks. In *Proceedings of the International Conference on Information Processing in Sensor Networks (IPSN)*, 2005.

[11] B. Karp and H.T. Kung. Greedy perimeter stateless routing for wireless networks. In *Proceedings of the International Conference on Mobile Computing and Networking (MobiCom)*, pages 243–254, 2000.

[12] D. Kossmann. The state of the art in distributed query processing. *ACM Computing Surveys*, 32(4):442–469, 2000.

[13] Y. Kotidis. Snapshot queries: towards data-centric sensor networks. In *Proceedings of the International Conference on Data Engineering (ICDE)*, 2005.

[14] S. Madden, M.J. Franklin, and J.M. Hellerstein. TAG: a tiny aggregation service for ad-hoc sensor networks. In *Proceedings of the Symposium on Operating Systems Design and Implementation (OSDI)*, pages 131–146, 2002.

[15] S. Madden, M.J. Franklin, J.M. Hellerstein, and W. Hong. The design of an acquisitional query processor for sensor networks. In *Proceedings of the SIGMOD Conference on Management of Data (SIGMOD)*, pages 491–502, 2003.

[16] A. Pandit and H. Gupta. Communication-effi cient implementation of range-join in sensor networks. In *Proceedings of the 11th International Conference on Database Systems for Advanced Applications (DASFAA)*, 2006.

[17] V. Raghunathan, A. Kansal, J. Hsu, J. Friedman, and M. Srivastava. Design considerations for solar energy harvesting wireless embedded systems. In *Proceedings of the International Conference on Information Processing in Sensor Networks (IPSN)*, 2005.

[18] T. Rappaport. *Wireless Communications: Principles and Practice*. Prentice-Hall Inc., 1996.

[19] A. Ricadela. Sensors everywhere. *Information Week*, Jan. 24, 2005.

[20] A. Savvides, M. Srivastava, L. Girod, and D. Estrin. Localization in sensor networks. *Wireless sensor networks*, pages 327–349, 2004.

[21] Sensoria Corp. WINS sensor platform. www.sensoria.com.

[22] M.A. Sharaf, J. Beaver, A. Labrinidis, and P.K. Chrysanthis. Balancing energy effi ciency and quality of aggregate data in sensor networks. *VLDB Journal*, 13(4):384–403, 2004.

[23] Y. Yao and J. Gehrke. The Cougar approach to in-network query processing in sensor networks. *SIGMOD Record*, 31(3):9–18, 2002.

[24] H. Yu, E.P. Lim, and J. Zhang. On in-network synopsis join processing for sensor networks. In *Proceedings of 7th International Conference on Mobile Data Management*, pages 32–39, 2006.

[25] F. Zhao and L. Guibas. *Wireless sensor networks: an information processing approach*. Morgan Kaufmann, 2004.