Communication-Efficient Implementation of Join in Sensor Networks

Vishal Chowdhary and Himanshu Gupta

SUNY, Stony Brook, NY 11754.
vishal,hgupta@cs.sunysb.edu

Abstract. A sensor network is a wireless ad hoc network of resource-constrained sensor nodes. In this article, we address the problem of communication-efficient implementation of the SQL "join" operator in sensor networks. We design an optimal join-implementation algorithm that provably incurs minimum communication cost under certain reasonable assumptions. In addition, we design a much faster suboptimal heuristic that empirically delivers a near-optimal solution. We evaluate the performance of our designed algorithms through extensive simulations.

1 Introduction

A sensor network consists of sensor nodes with a short-range radio and on-board processing capability forming a multi-hop network of an irregular topology. Each sensor node can sense certain physical phenomena like light, temperature, or vibration. There are many exciting applications [3, 13, 14] of such sensor networks, including monitoring and surveillance systems in both military and civilian contexts, building smart environments and infrastructures such as intelligent transportation systems and smart homes. In a sensor network, sensor nodes generate data items that are simply readings of one or more sensing devices on the node. Thus, a sensor network can be viewed as a distributed database system where each sensor node generates a stream of data tuples. Appropriately enough, the term *sensor database* is increasingly being used in research literature. Like a database, the sensor network is queried to gather and/or process the sensed data tuples. Database queries in SQL are a very general representation of queries over data, and efficient implementation of SQL queries is of great significance because of the enormous amount of data present in a typical sensor network. Since sensor nodes have limited battery energy, the distributed implementation of SQL queries in sensor networks must minimize the communication cost incurred, which is the main consumer of battery energy [31].

In this article, we address how to efficiently execute database queries in a sensor network, when the data distributed across sensors in a sensor network is viewed as relational database tables. In particular, we address communication-efficient in-network processing of the join operator, which is essentially a cartesian product of the operand tables followed by a predicate selection. We design

an optimal algorithm for a join operation that provably incurs minimum communication cost in dense sensor networks under some reasonable assumptions of communication cost and computation model. We also design a much faster suboptimal heuristic that empirically performs very close to the optimal algorithm, and results in significant savings over the naive approaches.

The rest of the paper is organized as follows. We start with modeling the sensor network as a database in Section 2. In Section 3, we present various algorithms for in-network implementation of the join operator, along with certain generalizations. We present our experiment results in Section 4. Related work is discussed in Section 5, and concluding remarks presented in Section 6.

2 Sensor Network Databases

A sensor network consists of a large number of sensors distributed randomly in a geographical region. Each sensor has limited on-board processing capability and is equipped with sensing devices. A sensor node also has a radio which is used to communicate directly with some of the sensors around it. Two sensor nodes S_1 and S_2 can directly communicate with each other if and only if the distance between them is less than the *transmission radius*. Sensor nodes may indirectly communicate with each other through other intermediate nodes – thus, forming a multi-hop network. We assume that each sensor node in the sensor network has a limited storage capacity of m units. Also, sensors have limited battery energy, which must be conserved for prolong unattended operation. Thus, we have focused on minimization of communication cost (hence, energy cost) as the key performance criteria of the join implementation strategies.

2.1 Modeling the Sensor Network as a Database

In a sensor network, the data generated by the sensor nodes is simply the readings of one or more sensing devices on the node. Thus, the data present in a sensor network can be modeled as relational database tables, wherein each sensor produces data records/tuples of a certain format and semantics. In some sense, a relational database table is a collection of similar-typed tuples from a group of sensors in the network. Due to the spatial and real-time nature of the data generated, a tuple usually has `timeStamp` and `nodeLocation` as attributes. In a sensor network, relational database tables are typically stream database tables [2] partitioned horizontally across (or generated by) a set of sensors in the network.

In-network Implementation. A plausible implementation of a sensor network database query engine could be to have an external database system handle all the queries over the network. In such a realization, all the data from each sensor node in the network is sent to the external system that handles the execution of queries completely. Such an implementation would incur very high communication costs and congestion-related bottlenecks. Thus, prior research has proposed

query engines that would execute the queries within the network with little external help. In particular, [18] shows that in-network implementation of database queries is fundamental to achieving energy-efficient communication in sensor networks. The focus of this article is communication-efficient in-network implementation of the join operator. As selection and projection are unary operators and operate on each tuple independently, they could be efficiently implemented using efficient routing and topology construction techniques. Union operation can be reduced to duplicate elimination, and the difference and intersection operations can be reduced to the join operation. Implementation of other database operators (aggregation, duplicate elimination, and outerjoins) is challenging and is the focus of our future work.

Querying and Cost Model. A query in a sensor network is initiated at a node called *query source* and the result of the query is required to be routed back to the query source for storage and/or consumption. A stream database table may be generated by a set of sensor nodes in a closed geographical region. The optimization algorithms, proposed in this article, to determine how to implement the join operation efficiently, are run at the query source. As typical sensor network queries are long running, the query source can gather all the catalogue information needed (estimated sizes and locations of the operand relations, join selectivity factor to estimate the size of the join result, density of the network) by initially sampling the operand tables. As mentioned before, we concentrate on implementations that minimize communication cost. We define the total communication cost incurred as the total data transfer between neighboring sensor nodes.

3 In-network Implementation of Join

In this section, we develop communication-efficient algorithms for implementation of a join operation in sensor networks. We start with assuming that the operand tables are static (non-streaming). Later in the section, we describe how our algorithms can be generalized for stream database tables, as data in sensor network is better represented as data stream tables.

The SQL join operator is used to correlate data from multiple tables, and can be defined as a selection predicate over the cross-product of a pair of tables; a join of R and S tables is denoted as $R \bowtie S$. Consider a join operation, initiated by a query source node Q, involving two static (non-streaming) tables R and S distributed horizontally across some geographical regions $\mathcal{R}$ and $\mathcal{S}$ in the network. We assume that the geographic regions are disjoint and small relative to the distances between the query source and the operand table regions (see [10] for a discussion on relaxation of this assumption). If we do not make any assumptions about the join predicates involved, each data tuple of table R should be paired with every tuple of S and checked for the join condition. The joined tuple is then routed (if it passes the join selection condition) to the query source Q where all the tuples are accumulated or consumed. Given that each

sensor node has limited memory resources, we need to find out appropriate regions in the network that would take the responsibility of computing the join. In particular, we may need to store and process the relations at some intermediate location before routing the result to the query source.

A simple nested-loop implementation of a join used in traditional databases is to generate the cross product (all pairs of tuples), and then extract those pairs that satisfy the selection predicate of the join. More involved implementations of a join operator widely used in database systems are merge-sort and hash-join. These classical methods are unsuitable for direct implemention in sensor networks due to the limited memory resources at each node in the network. Moreover, the traditional join algorithms focus on minimizing computation cost, while in sensor networks the primary performance criteria is communication cost. Below, we discuss various techniques for efficient implementation of the join operation in sensor networks.

Naive Approach. A simple way to compute $R \bowtie S$ could be to route the tuples of S from their original location $\mathcal{S}$ to the region $\mathcal{R}$, broadcast the S-tuples in the region $\mathcal{R}$, compute the join within the region $\mathcal{R}$, and then route the joined tuples to the query source Q. We refer to this approach as the *Naive* approach. Note that the roles of the tables R and S can be interchanged in the above approach.

Centroid Approach. Centroid approach is to compute the join operation in a circular region around some point C in the sensor network. In particular, let P_c be the smallest circular region around C such that the region P_c has at least $|R|/m$ sensor nodes to store the table R. First, we route both the operand table to C. Second, we distribute R and broadcast S in the region P_c around C. Lastly, we compute the join operation, and route the resulting tuples of $(R \bowtie S)$ to the query source Q. Since the communication cost incurred in the second step is independent of the choice of C, it is easy to see that the communication cost incurred in the above approach is minimized when the point C is the weighted centroid of the triangle formed by R, S, and Q. Here, the choice of the centroid point C is weighted by the sizes of R, S, and $(R \bowtie S)$.

3.1 Optimal Join Algorithm

In this section, we present an algorithm that constructs an optimal region for computing the join operation using minimum communication cost. We assume that the sensor network is sufficiently dense that we can find a sensor node at any point in the region. To formally prove the claim of optimality, we need to restrict ourselves to a class of join algorithms called *Distribute-Broadcast Join Algorithms* (defined below). In effect, our claim of optimality states that the proposed join algorithm incurs less communication cost than any distribute-broadcast join algorithm.

Definition 1. *A join algorithm to compute $R \bowtie S$ in a sensor network is a* distribute-broadcast join algorithm *if the join is processed by first distributing*

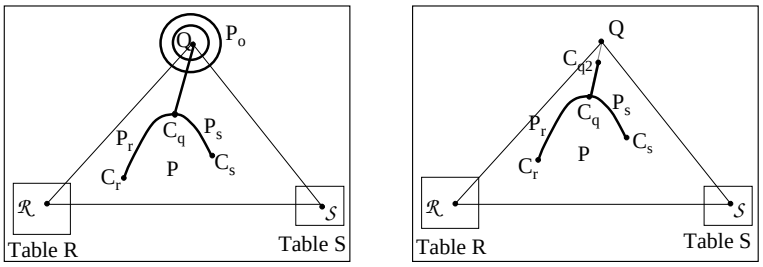


Fig. 1. Possible Shape of an Optimal Join-Region.

the table R in some region P (other than the region $\mathcal{R}$ storing R)[1] *of the sensor network followed by broadcasting the relation S within the region P to compute the join. The joined tuples are then routed from each sensor in the region P to the query source.*

As before, consider a query source Q and regions $\mathcal{R}$ and $\mathcal{S}$ that store the static operand tables R and S in a sensor network. The key challenge in designing an optimal algorithm for implementation of a join operation is to select a region P for processing the join in such a way that the total communication cost is minimized. We use the term *join-region* to refer to a region in the sensor network that is responsible for computing the join.

Shape of an Optimal Join-Region. Theorem 1 (see [10] for proof) shows that the join-region P that incurs minimum communication cost has a shape as shown in Figure 1 (a) or (b). In particular, the optimal join-region P is formed using three point C_r, C_s, and C_q in the sensor network (typically these points will lie within the $\triangle\mathcal{RSQ}$). More precisely, given three points C_r, C_s, and C_q in the sensor network, the region P takes one of the following forms:

1. Region P is formed of the paths $P_r = (C_r, C_q)$ and $P_s = (C_s, C_q)$, the line segment $\overline{C_qQ}$, and a circular region P_O of appropriate radius around Q. See Figure 1 (a).
2. Region P is formed of the paths $P_r = (C_r, C_q)$ and $P_s = (C_s, C_q)$, and a *part* of the line segment $\overline{C_qQ}$. See Figure 1 (b).

The total number of sensors in the region P is $l = |R|/m$, where $|R|$ is the size of the table R which will be distributed over the region P, and m is the memory size of each sensor node.

Theorem 1. *The shape of the join-region P used by a distribute-broadcast join algorithm that incurs optimal communication cost is as described above or as depicted in Figure 1 (a) or (b).* ∎

[1] Else, the algorithm will be identical to one of the Naive Approaches.

Theorem 1 restricts the shape of an optimal join-region. However, there are still an infinite number of possible join-regions of shapes depicted in Figure 1. Below, we further restrict the shape of an optimal join-region by characterizing the equations of the paths P_r and P_s, which connect C_r and C_s respectively to Q. We start with a definition.

Definition 2. *The* sensor length *between a region $\mathcal{X}$ and a point y in a sensor network plane is denoted as $d(\mathcal{X}, y)$ and is defined as the average weighted distance, in terms of number of hops/sensors, between the region $\mathcal{X}$ and the point y. Here, the distance between a point $x \in \mathcal{X}$ and y is weighted by the amount of data residing at x.*

Optimizing Paths P_r and P_s. Consider an optimal join-region P that implements a join operation using minimum communication cost. From Theorem 1, we know that the region P is of the shape depicted in Figure 1 (a) or (b). The total communication cost T incurred in processing of a join using the region P is

$$|R|d(\mathcal{R}, C_r) + |S|d(\mathcal{S}, C_s) + |R \bowtie S|d(P, Q) + |R||P|/2 + |S||P|,$$

where the first two terms represent the cost of routing R and S to C_r and C_s respectively, the third term represents the cost of routing the result $R \bowtie S$ from P to Q, and the last two terms represent the cost of distributing R and broadcasting S in the region P. Here, we assume that lack of global knowledge about the other sensors' locations and available memory capacities preclude the possibility of distributing or broadcasting more efficiently than doing it in a simple linear manner. Now, the only component of cost T that depends on the *shape* of P is $|R \bowtie S|d(P, Q)$. Let $P' = P - P_r - P_s$, i.e., the region P without the paths P_r and P_s. Since the result $|R \bowtie S|$ is evenly spread along the entire region P, we have $d(P, Q) = \frac{1}{|P|}(|P'|d(P', Q) + |P_r|d(P_r, Q) + |P_s|d(P_s, Q))$, where the notation $|B|$ for a region B denotes the number of sensor nodes in the region B. For a given set of points C_r, C_s, and C_q, the total communication cost T is minimized when the path P_r is constructed such that $|P_r|d(P_r, Q)$ is minimized. Otherwise, we could reconstruct P_r with a smaller $|P_r|d(P_r, Q)$, and remove/add sensors nodes from the end[2] of the region P' to maintain $|P| = |R|/m$. Removal of sensor nodes from P' will always reduce T, and it can be shown that addition of sensor nodes to the end of the region P' will not increase the cost more than the reduction achieved by optimizing P_r. Similarly, the path P_s could be optimized independently.

We now derive the equation of the path P_r that minimizes $|P_r|d(P_r, Q)$ for a given C_r and C_q. Consider an arbitrary point $R(x, y)$ along the optimal path P_r. The length of an infinitesimally small segment of the path P_r beginning at $R(x, y)$ is $\sqrt{(dx)^2 + (dy)^2}$, and the average distance of this segment from Q is $\sqrt{x^2 + y^2}$, if the coordinates of Q are $(0, 0)$. Sum of all these distances over the path P_r is $F = \int_{x_1}^{x_2} \sqrt{x^2 + y^2}\sqrt{(1 + (y')^2}\ dx$. To get the equation for the path

[2] Here, by the end of the region P', we mean either the circular part P_O or the line segment $\overline{C_q C_{q2}}$ depending on the shape.

P_r, we would need to determine the extremals of the above function F. Using the technique of calculus of variations [15], we can show that the extremal values of F satisfy the Euler-Lagrange differential equation. The equation of the path P_r can thus be computed as (we omit the details):

$$\beta = x^2 \cos\alpha + 2xy \sin\alpha - y^2 \cos\alpha$$

where the constants α and β are evaluated by substituting for coordinates of C_r and C_q in the equation.

Optimal Join Algorithm. Given points C_r, C_s, C_q, and Q, let P_r and P_s be the optimized paths connecting C_r and C_s to C_q respectively as described above. For a **given** triplet of points (C_r, C_s, C_q), the optimal join-region P is as follows. Let $l = |R|/m$ and $l_Y = |P_r| + |P_s| + |\overline{C_q Q}|$.

- When $l_Y < l, P = P_r \cup P_s \cup \overline{C_q Q} \cup P_O$, where P_O is a circular region around Q such that $|P_O| = l - (|\overline{C_q, Q}| + |P_r| + |P_s|)$. See Figure 1 (a).
- When $l_Y \geq l, P = P_r \cup P_s \cup \overline{C_q C_{q2}}$, where C_{q2} is such that $|\overline{C_q C_{q2}}| = l - (|P_r| + |P_s|)$. See Figure 1 (b).

Now, we can construct an optimal join-region to compute a join operation for tables R and S and the query source Q, by considering all possible triples of points C_r, C_s, and C_q in the sensor network, and picking the triplet (C_r, C_s, C_q) that results in a join-region P (as describe above) with minimum communication cost. The time complexity of the above algorithm is $O(n^3)$, where n is the total number of sensor nodes in the sensor network.

Suboptimal Heuristic. The high time complexity of the optimal algorithm described above makes it impractical for large sensor networks.

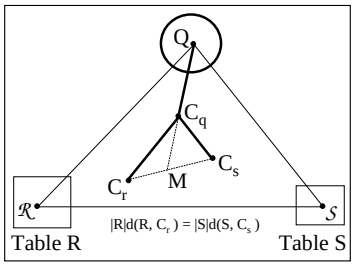


Fig. 2. Heuristic

Thus, we propose a suboptimal heuristic that runs in $O(n^{3/2})$ time, and incidentally performs very well in practice. Essentially, for a given C_r, we stipulate that C_s should be symmetrically ($|R|d(\mathcal{R}, C_r) = |S|d(\mathcal{S}, C_s)$) located in the $\triangle\mathcal{R}Q\mathcal{S}$. In addition, we approximate paths P_r and P_s to be straight line segments, and choose the point C_q on the median of the $\triangle C_r C_s Q$. See Figure 2. Thus, for each point as C_r in the sensor network, we determine C_s and search for the best C_q on the median of $\triangle C_r C_s Q$.

3.2 Join Implementation for Stream Database Tables

In the previous subsection, we discussed implementation of the join operation in a sensor network for static database tables. Since, sensor network data is better represented as stream database tables, we now generalize the algorithms to handle stream database tables. First, we start with presenting our model of stream database tables in sensor networks.

Data Streams in Sensor Networks. As for the case of static tables, a stream database table R corresponding to a data stream in a sensor network is associated with a region $\mathcal{R}$, where each node in $\mathcal{R}$ is continually generating tuples for the table R. To deal with the unbounded size of stream database tables, the tables are usually restricted to a finite set of tuples called the *sliding window* [1, 12, 27]. In effect, we expire or archive tuples from the data stream based on some criteria so that the total number of stored tuples does not exceed the bounded window size. We use W_R to denote the sliding window for a stream database table R.

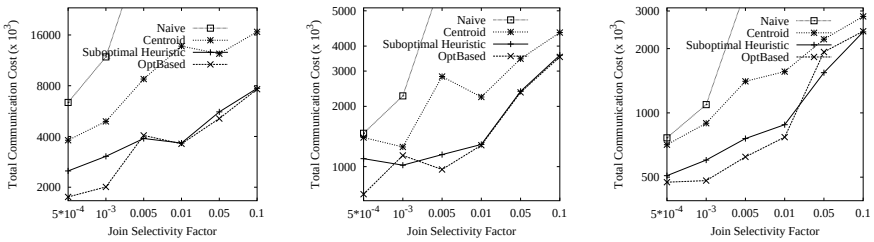
Naive Approach for Stream Tables. In the Naive Approach, we use the region $\mathcal{R}$ (or $\mathcal{S}$) to store the windows W_R *and* W_S of the stream tables R and S.[3] Each sensor node in the region $\mathcal{R}$ uses $W_R/(|W_R| + |W_S|)$ fraction of its local memory to store tuples of W_R, and the remaining fraction of the memory to store tuples of W_S. To perform the join operation, each newly generated tuple (of R or S) is broadcast to all the nodes in the region $\mathcal{R}$, and is also stored in some node of $\mathcal{R}$ with available memory. Note that the generated data tuples of S need to be first routed from the region $\mathcal{S}$ to the region $\mathcal{R}$. The resulting joined tuples are routed from $\mathcal{R}$ to the query source Q.

Generalizing Other Approaches. The other approaches viz. Centroid Approach, Optimal Algorithm, and Suboptimal Heuristic, use a join-region that is separate from the regions $\mathcal{R}$ and $\mathcal{S}$. These algorithms are generalized to handle stream database tables as follows. First, the strategy to choose the join-region P remains the same as before for static tables, except for the size of the join-region. For stream database tables, the chosen join-region is used to store W_R as well as W_S, with each sensor node in the join-region using $W_R/|W_R| + |W_S|$ fraction of its memory to store tuples of W_R, and the rest to store tuples of W_S. Each newly generated tuple (of R or S) is routed from its source node in $\mathcal{R}$ or $\mathcal{S}$ to the join-region P, and broadcast to all the nodes in P. The resulting joined tuples are then routed to Q. As part of the broadcast process (without incurring any additional communication cost), each generated tuple of R (or S) is also stored at some node in P with available memory.

4 Performance Evaluation

In this section, we compare the performance of Naive Approach, Centroid Algorithm, Optimal Algorithm, and Suboptimal Heuristic. In our previous discussion, we have assumed dense sensor networks where we can find a sensor node at any desirable point in the region. On real sensor networks, we use our proposed algorithms in conjunction with the trajectory based forwarding (TBF) routing technique [28], which works by forwarding packets to nodes closest to the intended path/trajectory. More specifically, to form the P_r, P_s, and $\overline{C_qQ}$ (or $\overline{C_qC_{q2}}$) parts of the join-region, we use nodes that are closest to uniformly spaced points on the geometrically constructed paths. In addition, each algorithm is generalized

[3] If the total memory of the nodes in $\mathcal{R}$ is not sufficient to store W_R and W_S, then the region $\mathcal{R}$ is expanded to include more sensor nodes.



(a) $t = 0.13$ units (b) $t = 0.15$ units (c) $t = 0.18$ units

Fig. 3. Total communication cost for various transmission radii (t), and fixed $\triangle\mathcal{RSQ}$.

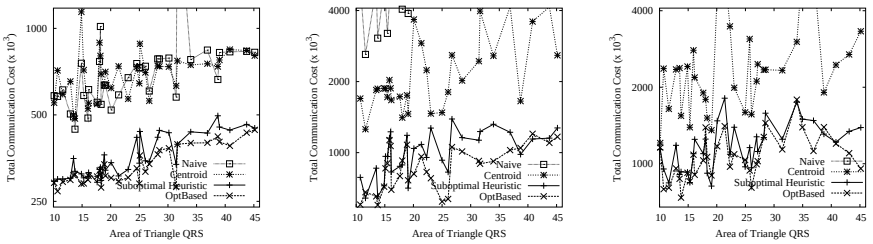
for stream database tables as discussed in Section 3.2. We refer to the generalized algorithms as Naive, Centroid, *OptBased*, and Suboptimal Heuristic respectively.

Definition 3. *Given instances of relations R and S and a join predicate, the* join-selectivity factor (f) *is the probability that a random pair of tuples from R and S will satisfy the given join predicate. In other words, the join selectivity factor is the ratio of the size of $R \bowtie S$ to the size of the cartesian product, i.e., $f = |R \bowtie S|/(|R||S|)$.*

Parameter Values and Experiments. We generated random sensor networks by randomly placing 10,000 sensors with uniform transmission radius (t) in an area of 10×10 units. For the purposes of comparing the performance of our algorithms, varying the number of sensors is tantamount to varying the transmission radius. Thus, we fix the number of sensors to be 10,000 and measure performance for different transmission radii. Memory size of a sensor node is 300 tuples, and the size of each of the sliding windows W_R and W_S of stream tables R and S is 8,000 tuples. For simplicity, we chose uniform data generation rates for R and S streams. In each of the experiments, we measure communication cost incurred in processing 8000 newly generated tuples of R and S each, after the join-region is already filled with previously generated tuples. We use the GPSR [19] algorithm to route tuples. Catalogue information is gathered for non-Naive approaches by collecting a small sample of data streams at the query source. In the first set of experiments, we consider a fixed $\triangle\mathcal{RSQ}$ and calculate the total communication cost for various transmission radii and join-selectivity factors. Next, we fix the transmission radius and calculate the total communication cost for various join-selectivity factors and various shapes/sizes of the $\triangle\mathcal{RSQ}$.

Fixed Triangle $\mathcal{RSQ}$. In this set of experiments (Figure 3), we fix the locations of regions R, S, and query source Q and measure the performance of our algorithms for various values of transmission radii and join-selectivity factors. In particular, we choose coordinates (0,0), (5,9.5), and (9.5,0) for R, Q, and S respectively.

We have looked at three transmission radii viz. 0.13, 0.15, and 0.18 units. Lower transmission radii left the sensor network disconnected, and the trend observed



(a) $f = 10^{-4}$ (b) $f = 5 * 10^{-3}$ (c) $f = 10^{-2}$

Fig. 5. Total communication cost for various $\triangle\mathcal{RSQ}$. Here, $t = 0.15$.

for these three transmission radii values is sufficient to infer behavior for larger transmission radii (see Figure 4). From Figure 3 (a)-(c), we can see that the Suboptimal Heuristic performs very close to the OptBased Algorithm, and significantly outperforms (upto 100%) the Naive and Centroid Approaches for most parameter values. The performance of the Naive approach worsens drastically with the increase in the join-selectivity factor, since the routing cost of the joined tuples from the join region ($\mathcal{R}$ or $\mathcal{S}$) to the query source Q becomes more dominant.

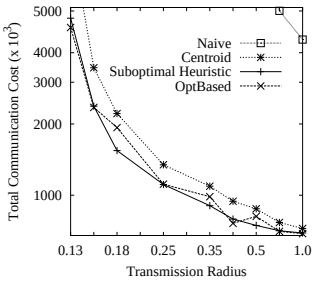


Fig. 4. Here, $f = 0.05$.

Fixed Transmission Radius (0.15 units). We also observe the performance of various algorithms for different size and shapes of $\triangle\mathcal{RSQ}$. In particular, we fix the transmission radius of each sensor node in the network to be 0.15 units, and generate various $\triangle\mathcal{RSQ}$'s as follows. We fix locations of regions $\mathcal{R}$ and $\mathcal{S}$, and select many locations of the query source Q with the constraint that the area of the $\triangle\mathcal{RSQ}$ is between 10% to 50% of the total sensor network area. For each such generated $\triangle\mathcal{RSQ}$, we run all the four algorithms for three representative join-selectivity factor values viz. $10^{-4}, 5 * 10^{-3}$, and 10^{-2}. See Figure 5. Again we observe that the Suboptimal Heuristic performs very close to the OptBased Algorithm, and incurs much less communication cost than the Naive and Centroid Approaches for all join-selectivity factor values.

Summary. From the above experiments, we observe that the Suboptimal Heuristic performs very close to the OptBased Algorithm, but performs substantially better than the Centroid and Naive Approaches for a wide range of sensor network parameters. The savings in communication cost reduce with the increase in join-selectivity factor and/or transmission radius. We expect the join-selectivity factor to be relatively low in large sensor networks because of large sizes of operand tables and data generated having only local spatial and temporal data

correlations. Moreover, since sensor nodes have the capability to adjust transmission power, effective topology control [30, 32] is used to minimize transmission radius at each node to conserve overall energy. Thus, the Suboptimal Heuristic is a natural choice for efficient implementation of join in sensor networks, and should result in substantial energy savings in practice.

5 Related Work

The vision of sensor network as a database has been proposed by many works [5, 16, 26], and simple query engines such as TinyDB [26] have been built for sensor networks. In particular, the COUGAR project [5, 33, 34] at Cornell University is one of the first attempts to model a sensor network as a database system. The TinyDB Project [26] at Berkeley also investigates query processing techniques for sensor networks. However, TinyDB implements very limited functionality [25] of the traditional database language SQL. A plausible implementation of an SQL query engine for sensor networks could be to ship all sensor nodes' data to an external server that handles the execution of queries completely [21]. Such an implementation would incur high communication costs and congestion-related bottlenecks. In particular, [18] shows that in-network implementation of database queries is fundamental to conserving energy in sensor networks. Thus, recent research has focussed on in-network implementation of database queries. However, prior research has only addressed limited SQL functionality – single queries involving simple aggregations [22, 24, 34] and/or selections [25] over single tables [23], or local joins [34]. So far, it has been considered that correlations such as median computation or joins should be computed on a single node [4, 25, 34]. In particular, [4] address the problem of operator placement for in-network query processing, assuming that each operator is executed locally and fully on a single sensor node. The problem of distributed and communication-efficient implementation of join has not been addressed yet in the context of sensor networks.

In addition, there has been a large body of work done on efficient query processing in data stream processing systems [6, 8, 9, 27]. In particular, [11] approximates sliding window joins over data streams and [17] has designed join algorithms for joining multiple data streams constrained by a sliding time window. However, a data stream processing system is not necessarily distributed and hence, minimizing communication cost is not the focus of the research. There has been a lot of work on query processing in distributed database systems [7, 20, 29], but sensor networks differ significantly from distributed database systems because of their multi-hop communication cost model and resource limitations.

6 Conclusions

Sensor networks are capable of generating large amounts of data. Hence, efficient query processing in sensor networks is of great importance. Since sensor nodes

have limited battery power and memory resources, designing communication-efficient distributed implementation of database queries is a key research challenge. In this article, we have focussed on implementation of the join operator, which is one of the core operators of database query language. In particular, we have designed an Optimal Algorithm that incurs minimum communication cost for implementation of join in sensor networks under certain reasonable assumptions. Moreover, we reduced the time complexity of the Optimal Algorithm to design a Suboptimal Heuristic, and showed through extensive simulations that the Suboptimal Heuristic performs very close to the Optimal Algorithm. Techniques developed in this article are shown to result in substantial energy savings over simpler approaches for a wide range of sensor network parameters.

References

1. D. J. Abadi, D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik. Aurora: a new model and architecture for data stream management. *The VLDB Journal*, 12(2):120–139, 2003.
2. B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS)*, 2002.
3. B. Badrinath, M. Srivastava, K. Mills, J. Scholtz, and K. Sollins, editors. *Special Issue on Smart Spaces and Environments*, IEEE Personal Communications, 2000.
4. B. Bonfils and P. Bonnet. Adaptive and decentralized operator placement for in-network query processing. In *Proceedings of the International Workshop on Information Processing in Sensor Networks (IPSN)*, 2003.
5. P. Bonnet, J. Gehrke, and P. Seshadri. Towards sensor database systems. In *Proceeding of the International Conference on Mobile Data Management*, 2001.
6. D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, G. Seidman, M. Stonebraker, N. Tatbul, and S. Zdonik. Monitoring streams - A new class of data management applications. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, 2002.
7. S. Ceri and G. Pelagatti. *Distributed Database Design: Principles and Systems*. MacGraw-Hill (New York NY), 1984.
8. S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. R. Madden, F. Reiss, and M. A. Shah. TelegraphCQ: Continuous dataflow processing. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, 2003.
9. J. Chen, D. J. DeWitt, F. Tian, and Y. Wang. NiagaraCQ: a scalable continuous query system for internet databases. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, 2000.
10. V. Chowdhary and H. Gupta. Communication-efficient implementation of join in sensor networks. Technical report, SUNY, Stony Brook, Computer Science Department, 2004.
11. A. Das, J. Gehrke, and M. Riedewald. Approximate join processing over data streams. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, 2003.
12. L. Ding, N. Mehta, E. Rundensteiner, and G. Heineman. Joining punctuated streams. In *Proceedings of the International Conference on Extending Database Technology*, 2004.
13. D. Estrin, R. Govindan, and J. Heidemann, editors. *Special Issue on Embedding the Internet*, Communications of the ACM, volume 43, 2000.
14. D. Estrin, R. Govindan, J. S. Heidemann, and S. Kumar. Next century challenges: Scalable coordination in sensor networks. In *Proceedings of the International Conference on Mobile Computing and Networking (MobiCom)*, 1999.
15. I. Gelfand and S. Fomin. *Calculus of Variations*. Dover Publications, 2000.
16. R. Govindan, J. Hellerstein, W. Hong, S. Madden, M. Franklin, and S. Shenker. The sensor network as a database. Technical report, University of Southern California, Computer Science Department, 2002.
17. M. Hammad, W. Aref, A. Catlin, M. Elfeky, and A. Elmagarmid. A stream database server for sensor applications. Technical report, Purdue University, Department of Computer Science, 2002.
18. J. S. Heidemann, F. Silva, C. Intanagonwiwat, R. Govindan, D. Estrin, and D. Ganesan. Building efficient wireless sensor networks with low-level naming. In *Symposium on Operating Systems Principles*, 2001.
19. B. Karp and H. Kung. Gpsr: greedy perimeter stateless routing for wireless networks. In *Proceedings of the International Conference on Mobile Computing and Networking (MobiCom)*, 2000.
20. D. Kossmann. The state of the art in distributed query processing. *ACM Computing Surveys*, 32(4), 2000.
21. S. Madden and M. Franklin. Fjording the stream: An architecture for queries over streaming sensor data. In *Proceedings of the International Conference on Database Engineering (ICDE)*, 2002.
22. S. Madden, M. Franklin, J. Hellerstein, and W. Hong. TAG: A tiny aggregation service for ad-hoc sensor networks. In *Proceedings of the Symposium on Operating Systems Design and Implementation (OSDI)*, 2002.
23. S. Madden and J. M. Hellerstein. Distributing queries over low-power wireless sensor networks. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, 2002.
24. S. Madden, R. Szewczyk, M. Franklin, and D. Culler. Supporting aggregate queries over ad-hoc wireless sensor networks. In *Workshop on Mobile Computing and Systems Applications*, 2002.
25. S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. The design of an acquisitional query processor for sensor networks. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, 2003.
26. S. R. Madden, J. M. Hellerstein, and W. Hong. TinyDB: In-network query processing in tinyos. http://telegraph.cs.berkeley.edu/tinydb, Sept. 2003.
27. R. Motwani, J. Widom, A. Arasu, B. Babcock, S. Babu, M. Datar, G. Manku, C. Olston, J. Rosenstein, and R. Varma. Query processing, approximation, and resource management in a data stream management system. In *Proceedings of the International Conference on Innovative Data Systems Research (CIDR)*, 2003.
28. B. Nath and D. Niculescu. Routing on a curve. In *Proceedings of the Workshop on Hot Topics in Networks*, 2002.
29. M. T. Ozsu and P. Valduriez. *Principles of Distributed Database Systems*. Prentice Hall, 1999.
30. J. Pan, Y. T. Hou, L. Cai, Y. Shi, and S. X. Shen. Topology control for wireless sensor networks. In *Proceedings of the International Conference on Mobile Computing and Networking (MobiCom)*, 2003.
31. G. Pottie and W. Kaiser. Wireless integrated sensor networks. *Communications of the ACM*, 43, 2000.
32. R. Ramanathan and R. Rosales-Hain. Topology control in multihop wireless networks using transmit power adjustment. In *Proceedings of the IEEE INFOCOM*, 2000.
33. Y. Yao and J. Gehrke. The cougar approach to in-network query processing in sensor networks. In *SIGMOD Record*, 2002.
34. Y. Yao and J. Gehrke. Query processing for sensor networks. In *Proceedings of the International Conference on Innovative Data Systems Research (CIDR)*, 2003.