Singapore Management University

Institutional Knowledge at Singapore Management University

On in-network synopsis join processing for sensor networks

Hai YU

Ee Peng LIM
Singapore Management University, eplim@smu.edu.sg

Jun ZHANG

Citation

In-Network Join Processing for Sensor Networks

Hai Yu, Ee-Peng Lim, and Jun Zhang

Center for Advanced Information Systems,
Nanyang Technological University, Singapore
yuhai@pmail.ntu.edu.sg
{aseplim, jzhang}@ntu.edu.sg

Abstract. Recent advances in hardware and wireless technologies have led to sensor networks consisting of large number of sensors capable of gathering and processing data collectively. Query processing on these sensor networks has to consider various inherent constraints. While simple queries such as select and aggregate queries in wireless sensor networks have been addressed in the literature, the processing of join queries in sensor networks remains to be investigated. In this paper, we present a synopsis join strategy for evaluating join queries in sensor networks with communication efficiency. In this strategy, instead of directly joining two relations distributed in a sensor network, synopses of the relations are firstly joined to prune those data tuples that do not contribute to join results. We discuss various issues related to the optimization of synopsis join. Through experiments, we show the effectiveness of the synopsis join techniques in terms of communication cost for different join selectivities and other parameters.

1 Introduction

The emergence of wireless technologies has enabled the development of tiny, low-power, wireless *sensors* capable of sensing physical phenomena such as temperature, humidity, etc.. Sensor networks have been adopted in various scientific and commercial applications [1, 2, 3]. Data collection in a sensor network is achieved by modeling it as a distributed database where sensor readings are collected and processed using queries [4, 5, 6].

In this paper, we address in-network join query processing in sensor networks. Join is an important operation in sensor networks for correlating sensor readings since a single sensor reading may not provide enough information representing a meaningful event or entity. Consider a sensor network covering a road network. Each sensor node can detect the ID's of vehicles in close vicinity, record the timestamps at which the vehicles are detected, and keep the timestamped records for a fixed duration, say 1 hour. Suppose N_R and N_S represent two sets of sensor nodes located at two regions of a road segment, Region1 and Region2, respectively. To gather the necessary data for determining the speeds of vehicles traveling between the two regions, the following join query can be expressed.

```
SELECT s1.vehId, s1.time, s2.time FROM s1, s2
WHERE  s1.loc IN Region1 AND s2.loc IN Region2 AND s1.vehId = s2.vehId
```
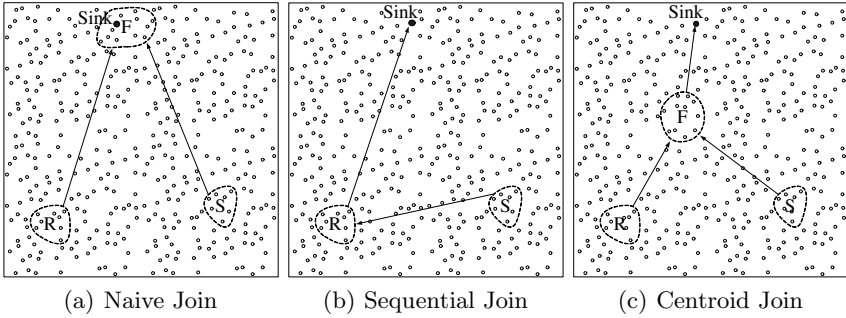
(a) Naive Join   (b) Sequential Join   (c) Centroid Join

Fig. 1. General Strategies

To evaluate the above query, sensor readings from Region1 and Region2 need to be collected and joined on the vehId attribute. We focus on addressing join scenarios whereby the join selectivity is so low that is it not cost-effective to ship source tuples to the sink for join. Therefore, efficient in-network join algorithms are required.

2   Preliminaries

Suppose a sensor network consisting of N sensor nodes. We assume there are two virtual tables in the sensor network, $\mathcal{R}$ and $\mathcal{S}$, containing sensor readings distributed in sensors. Each sensor reading is a tuple with two mandatory attributes, `timestamp` and `sensorID`, indicating the time and the sensor at which the tuple is generated. A sensor reading may contain other attributes that are measurements generated by a sensor or multiple sensors, e.g., temperature. We are interested in the evaluation of static one-shot *binary equi-join* (BEJ) queries in sensor networks. A BEJ query for sensor networks is defined as follows.

Definition 1. *Given two sensor tables* $\mathcal{R}(A_1, A_2, \ldots, A_n)$ *and* $\mathcal{S}(B_1, B_2, \ldots, B_m)$, *a binary equi-join (BEJ) is*

$$\mathcal{R} \bowtie_{A_i = B_j} \mathcal{S} \qquad (i \in \{1, 2, \ldots, n\}, j \in \{1, 2, \ldots, m\}) ,$$

where A_i *and* B_j *are two attributes of* $\mathcal{R}$ *and* $\mathcal{S}$ *respectively, which have the same domain.*

We assume that $\mathcal{R}$ and $\mathcal{S}$ are stored in two sets of sensor nodes N_R and N_S located in two distinct regions known as R and S, respectively. A BEJ query can be issued from any sensor node called *query sink*, which is responsible for collecting the join result. Due to limited memory, the query sink cannot perform the join by itself. A set of nodes is required to process the join collaboratively, referred to as *join nodes*. The join processing can be divided into three stages, *query dissemination*, *join evaluation*, and *result collection*.

In the query dissemination stage, the sink sends a BEJ query to one of the N_R (and N_S) nodes using a location-based routing protocol such as GPSR [7].

Once the first N_R (and N_S) node receives the query, the node broadcasts the query among the N_R (N_S) nodes. The query dissemination cost is therefore $O(\sqrt{N} + |N_R| + |N_S|)$. When sensors receive the query, they send their local data to the join nodes which are either determined in the query dissemination phase, or adaptively selected according to the network conditions in the join evaluation phase. Once the join results are ready, the query sink collects the join results from the join nodes [1].

Our objective is to minimize the total communication cost for processing a given BEJ query in order to prolong the sensor network lifetime. In addition, the join scheme has to ensure that the memory space needed by the join operation on each join node does not exceed the available memory space. In the next section, we present several general strategies for performing in-network join. In Section 4 we describe a *synopsis join strategy* in which unnecessary data transmission is reduced by an additional synopsis join process.

3   General Strategies

In general strategies, *join nodes* N_F are selected to join tuples of $\mathcal{R}$ and $\mathcal{S}$, without attempting to firstly filter out tuples that are not involved in the join results (referred to as *non-candidate tuples*).

When a join query is issued, a *join node selection* process is initiated to find a set of join nodes N_F to perform the join. $\mathcal{R}$ tuples are routed to a *join region* F where the join nodes N_F reside in. Each join node $n_f \in N_F$ stores a *horizontal partition* of the table $\mathcal{R}$, denoted as $\mathcal{R}_f$. $\mathcal{S}$ tuples are transmitted to and broadcast in F. Each join node n_f receives a copy of $\mathcal{S}$ and processes local join $\mathcal{R}_f \bowtie \mathcal{S}$. The query sink obtains the join results by collecting the partial join results at each n_f. Note that though $\mathcal{S}$ could be large, the local join $\mathcal{R}_f \bowtie \mathcal{S}$ at n_f can be performed in a pipelined manner to avoid memory overflow [8].

The selection of N_F is critical to the join performance. Join node selection involves selecting the number of nodes in N_F, denoted by $|N_F|$, and the location of the join region F. To avoid memory overflow, assuming $\mathcal{R}$ is evenly distributed in N_F, $|N_F|$ should be at least $|\mathcal{R}|/m$, where $|\mathcal{R}|$ denotes the number of tuples in $\mathcal{R}$ and m denotes the maximum number of $\mathcal{R}$ tuples a join node n_f can store.

Depending on the location of the join region, we have at least three join strategies, namely, *naive join*, *sequential join*, and *centroid join* (see Figure 1).

Naive Join. In naive join, sensor nodes around the sink are selected as the join nodes N_F, so that the cost of routing join results to the sink can be minimized (Figure 1(a)). The communication cost involves routing tables $\mathcal{R}$ and $\mathcal{S}$ to the join region F, broadcasting $\mathcal{S}$ to the join nodes, and sending the join results from N_F to the sink (shown in Equation 1)[2]. Naive join establishes a basis

---

[1] Note that although the query sink may not be able to evaluate the join, it is able to consume the join results since it can retrieve and process the join results tuple by tuple with little memory usage.

for performances of all join strategies, since any join strategy should at least perform better than naive join in terms of total communication cost in order to be a reasonable join strategy.

$$C_{\text{naive}-\text{join}} = |\mathcal{S}| \operatorname{dist}(S, F) + |\mathcal{S}| |N_F| + \sum_{n_f \in N_F} (|\mathcal{R}_f| \operatorname{dist}(R, n_f) + \tag{1}$$
$$|\mathcal{R}_f \bowtie \mathcal{S}| \operatorname{dist}(n_f, \text{sink})) .$$

Sequential Join. Sequential join minimizes the cost of routing and distributing $\mathcal{R}$ tuples to the join region by selecting the nodes N_R as N_F (see Figure 1(b)). In this case, $\mathcal{R}$ tuples remain in their respective nodes. $\mathcal{S}$ tuples are routed to the region R, and broadcast to all nodes N_R. Each node $n_i \in N_R$ performs the local join $\mathcal{R}_i \bowtie \mathcal{S}$ where $\mathcal{R}_i$ is the local table stored at n_i. Join results are delivered to the sink as shown in Figure 1(b). The communication cost of this strategy is:

$$C_{\text{seq}-\text{join}} = |\mathcal{S}| \cdot \operatorname{dist}(R, S) + |\mathcal{S}| \cdot |N_R| + \sum_{n_i \in N_R} |\mathcal{R}_i \bowtie \mathcal{S}| \cdot \operatorname{dist}(R, \text{sink}) . \tag{2}$$

Centroid Join. Centroid join selects an optimal join region within the triangle formed by R, S, such that the total communication cost are minimized (see Figure 1(c)). The communication cost is shown in Equation 3. Path-Join [9] is an example of this strategy, which tries to find an optimal join region by minimizing a target cost function. Note that naive join and sequential join are special cases of centroid join.

$$C_{\text{cen}-\text{join}} = \sum_{n_j \in N_S} |S_j| \cdot \operatorname{dist}(n_j, F) + |S| \cdot |N_F| + \sum_{n_i \in N_R} |R_i| \cdot \operatorname{dist}(n_i, F) . \tag{3}$$

The above three strategies can be further optimized for BEJ queries. A hash-based join can be applied in which both $\mathcal{R}$ and $\mathcal{S}$ are partitioned into a number of disjoint sub-tables, each with a join attribute value range. Each node n_f in N_F [3] is dedicated to join two subsets of $\mathcal{R}^v$ and $\mathcal{S}^v$ with the same join value range v. In this way, tuples with the same join attribute value are always joined at the same join node, and the broadcasting of $\mathcal{S}$ in N_F can be avoided.

The major problem associated with general strategies is the communication overhead for transmitting non-candidate tuples in $\mathcal{R}$ and $\mathcal{S}$, especially for queries with low join selectivity.

---

[2] dist(A,B) refers to the hop distance between A and B. If A, B are two nodes, dist(A,B) is the average hop distance of the routes selected by the routing protocol. If A, B are two regions, dist(A,B) is the average hop distance between any pair of nodes from A and B. If A is a region and B is a node, dist(A,B) refers to the average hop distance between B and all nodes in A.

[3] A subset of N_F is needed if one node does not have enough memory space for handling the join.
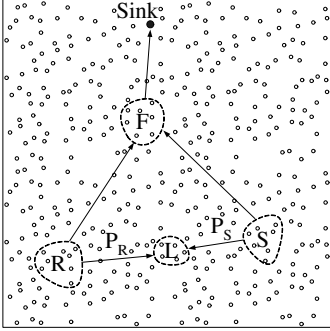
Timestamp	Vehicle-Type	Speed (km/h)
10:23:12	car	82
10:25:29	bus	69
10:30:48	car	85
10:31:31	lorry	70
10:36:07	lorry	62
10:36:40	car	78

(a) Original Table $\mathcal{R}$

Tuple-ID	Vehicle-Type	Count
t1	car	3
t2	bus	1
t4	lorry	2

(b) Synopsis $S(\mathcal{R})$

Fig. 2. Synopsis Join Strategy   **Fig. 3.** An example of synopsis

4 Synopsis Join Strategy

The synopsis join strategy prunes non-candidate tuples and only joins candidate tuples. The key to the pruning process is to keep the cost overhead as low as possible. The synopsis join strategy comprises three phases, *synopsis join*, *notification transmission* and *final join*.

4.1 Synopsis Join

The synopsis join phase performs an inexpensive synopsis join, aiming at reducing the number of $\mathcal{R}$ and $\mathcal{S}$ tuples to be transmitted for final join. The synopsis join phase comprises two steps: *synopsis generation*, *synopsis join*.

Synopsis Generation. A *synopsis* is a digest of a relation that is able to represent the relation to perform operations such as aggregation or join. We denote $S(\mathcal{R})$ as the synopsis of a table $\mathcal{R}$. A synopsis can be in any form such as histograms, wavelets, etc., which is generally smaller than the size of the corresponding table. In this paper, we adopt simple histograms as synopses where a synopsis is represented by the join attribute values of a table and their frequencies. For example, assume a sensor table $\mathcal{R}$ shown in Figure 3(a). Let the join attribute be `Vehicle-type`. The corresponding synopsis $S(\mathcal{R})$ consists of two attributes, the join attribute value, whose domain is all possible values of `Vehicle-Type`, and the number of tuples for each `Vehicle-Type`, as shown in Figure 3(b).

In synopsis generation, each sensor generates a synopsis of its local table. Consider a relation $\mathcal{R}$ distributed among N_R sensor nodes. Each node $n_i \in N_R$ stores a local table $\mathcal{R}_i$ that is part of $\mathcal{R}$. n_i generates a *local synopsis* $S(\mathcal{R}_i)$ by extracting the join column A_J of $\mathcal{R}_i$, and computing the frequencies of the distinct values in A_J. Assuming uniform data distribution, we can derive $|S(\mathcal{R}_i)|$ as:

$$|S(\mathcal{R}_i)| = |A_J| \left(1 - \left(1 - \frac{1}{|N_R|} \right)^{|R|/|A_J|} \right) . \tag{4}$$

Synopsis Join. In this stage, a set of *synopsis join nodes* N_L in the *synopsis join region* L is selected to join the synopses of $\mathcal{R}$ and $\mathcal{S}$ to determine the candidate tuples in $\mathcal{R}$ and $\mathcal{S}$ (see Figure 2). Once N_L nodes are determined, the local synopses are routed to N_L for synopsis join. For BEJ queries, each synopsis join node $n_l \in N_L$ is assigned a range v of join attribute values using a geographic hash function [10], so that only synopses with join attribute value in v are transmitted to n_l for synopsis join. For a node $n_i \in N_R$, the local synopsis $S(\mathcal{R}_i)$ is divided into $|N_L|$ partitions. A partition $S_l^v(\mathcal{R}_i)$ containing a synopsis of tuples with join attribute values in v is sent to n_l maintaining the range v.

Consider the example shown in Figure 3. Suppose there are two synopsis join nodes n_{l1} and n_{l2}. n_{l1} is dedicated to handle join attribute values `car`, while n_{l2} handles `bus` and `lorry`. When a sensor n_{i1} generates a local synopses as the one in Figure 3(b), it divides the synopses into two partitions, one partition $S_1(R_{n_{i1}})$ contains tuples $t1$ and $t3$, whose join attribute values are `car`, and the other partition $S_2(R_{n_{i1}})$ contains tuples $t2$ and $t4$ whose join attribute values are `bus` and `lorry`. Therefore $S_1(R_{n_{i1}})$ and $S_2(R_{n_{i1}})$ are sent to n_{l1} and n_{l2} for synopsis join, respectively.

The synopsis join nodes perform synopsis join as synopses from N_R and N_S nodes arrive. We denote a synopsis from a node $n_i \in N_R$ received by a synopsis join node n_l as $S_l(\mathcal{R}_i)$. A synopsis join operation performed at n_l is defined as follows.

$$\biguplus_{n_i \in N_R} S_l(\mathcal{R}_i) \bowtie \biguplus_{n_j \in N_S} S_l(\mathcal{S}_j) . \tag{5}$$

The $\uplus$ operator is a merge function which takes multiple synopses as inputs and produces a new synopsis. In particular, for our histogram synopsis, $\uplus$ is defined as a function that accumulates the frequency values if two input tuples are of the same join attribute value. The output of $\uplus$ is therefore the accumulation of the input histograms.

Synopsis Join Node Selection. The number of synopsis join nodes is determined by the sizes of local synopses N_L nodes receive. Specifically, suppose a node's memory space is m_s (number of synopsis tuples that can fit into a node), the number of synopsis join nodes is determined as follows.

$$|N_L| = \frac{1}{m_s} \sum_{n_i \in N_R} |S(\mathcal{R}_i)| . \tag{6}$$

The locations of N_L nodes are selected so that the communication cost for routing local synopses is minimized. The communication cost of sending local synopses from N_R and N_S nodes to N_L nodes can be expressed as:

$$\sum_{n_l \in N_L} \sum_{n_i \in N_R} |S_l(\mathcal{R}_i)| \cdot \mathrm{dist}(n_l, n_i) + \sum_{n_l \in N_L} \sum_{n_j \in N_S} |S_l(\mathcal{S}_j)| \cdot \mathrm{dist}(n_l, n_j) . \tag{7}$$

Assuming the synopsis join region L is small, we can simplify the above equation:

$$C_{\mathrm{synopsis-routing}} = |P_R| \sum_{n_i \in N_R} |S(\mathcal{R}_i)| + |P_S| \sum_{n_j \in N_S} |S(\mathcal{S}_j)| , \tag{8}$$

where $|P_R|$ (or $|P_S|$) is $\text{dist}(R, L)$ (or $\text{dist}(S, L)$) [4]. Given the above equation, the optimal set of synopsis join nodes that minimize $C_{\text{syno-join}}$ are located on the line connecting R and S. Therefore $|P_R| + |P_S| = \text{dist}(R, S)$. Assuming $\sum_{n_i \in N_R} |S(\mathcal{R}_i)| > \sum_{n_j \in N_S} |S(\mathcal{S}_j)|$, it is obvious that $C_{\text{synopsis-routing}}$ is minimized when $|P_R|$ is zero, and $|P_S|$ is $\text{dist}(R, S)$. Hence,

$$\min(C_{\text{synopsis-routing}}) = \text{dist}(R, S) \sum_{n_j \in N_S} |S(\mathcal{S}_j)| \ . \tag{9}$$

Therefore, the optimal set of synopsis join nodes N_L are chosen from nodes in N_R that are nearest to N_S, assuming the size of the region R is small compared to the distance between R and S. Optimal selection of N_L for arbitrary R and S regions are part of the future work.

4.2   Notification Transmission

Each sensor node in N_R and N_S needs to be notified of which are the candidate tuples. To achieve this, a synopsis join node n_l stores the ID of the sensor a local synopsis originates from. For each join attribute value a, it identifies two set of sensors N_R^a and N_S^a storing tuples with join attribute value a, and selects a *final join node* n_f to join these tuples, such that the communication cost of sending data tuples with join attribute value a from N_R^a and N_S^a to n_f, and sending the results from n_f to the sink is minimized. Therefore n_f is the node that minimizes the cost function in Formula 10.

$$\sum_{n_i \in N_R^a} |\mathcal{R}_i^a| \cdot \text{dist}(n_i, n_f) + \sum_{n_j \in N_S^a} |\mathcal{S}_j^a| \cdot \text{dist}(n_j, n_f)$$
$$+ \sum_{n_i \in l_r^a} |\mathcal{R}_i^a| \cdot \sum_{n_j \in l_s^v} |\mathcal{S}_j^a| \cdot \text{dist}(n_f, \text{sink}) \ , \tag{10}$$

where $|\mathcal{R}_i^a|$ and $|\mathcal{S}_j^a|$ denote the number of $\mathcal{R}$ tuples in n_i and $\mathcal{S}$ tuples in n_j with the join attribute value a, respectively.

In order to simplify the problem, the weighted centers of sensors in N_R^a and N_S^a are derived, respectively. The weighted center c of a set of sensors N storing a table $\mathcal{T}$ are defined in Equation 11, where $\mathcal{T}_i$ refers to the table stored in node n_i, and $\text{loc}(n)$ refers to the location of a node n.

$$\text{loc}(c) = \frac{1}{\sum_{n_i \in N} |\mathcal{T}_i|} \cdot \sum_{n_i \in N} |\mathcal{T}_i| \cdot \text{loc}(n_i) \ . \tag{11}$$

With Formula 11, the weighted centers c_r and c_s for sensors in N_R^a and N_S^a can be computed respectively. Since $\sum_{n_i \in N_R^a} |\mathcal{R}_i^a| = |\mathcal{R}^a|$ and $\sum_{n_j \in N_S^a} |\mathcal{S}_j^a| = |\mathcal{S}^a|$, we can rewrite Formula 10 as:

$$|\mathcal{R}^a| \cdot \text{dist}(c_r, n_f) + |\mathcal{S}^a| \cdot \text{dist}(c_s, n_f) + |\mathcal{R}^a| \cdot |\mathcal{S}^a| \cdot \text{dist}(n_f, \text{sink}) \ . \tag{12}$$

---

[4] P_R (or P_S) is the path connecting the centers of R (or S) and L.

Formula 12 is minimum when n_f is the *generalized Fermat's point* [11] of the triangle formed by c_r, c_s, and the sink. Note that there may not exist a sensor located at the derived generalized Fermat's point g. GPSR is used to select a node that is nearest to g as the final join node n_f.

The same operation is performed for all join attribute values handled by n_l. When synopsis join is completed, n_l obtains for each sensor node n_i a set of $\langle a, n_f \rangle$ pairs, which means tuples stored in n_i with the join attribute value a are to be sent to n_f for final join. The set of pairs are sent to n_i in a *notification message*. A notification message can be broken up into multiple ones if it cannot fit into one network packet. The communication cost for notification transmission is similar to Equation 9.

$$C_{\text{notification}} = \text{dist}(R, S) \sum_{n_i \in N_R \cup N_S} |d_i| \, , \qquad (13)$$

where $|d_i|$ denotes the total size of the notification messages sensor n_i receives.

4.3   Final Join

Upon receiving a notification message from a synopsis join node, each node in N_R or N_S sends the candidate tuples whose join attribute values are specified in the notification message to a final join node n_f. In the final join stage, a group of final join nodes N_F are selected to join the candidate tuples sent from R and S, as shown in Figure 2. The final join node n_f performs the join $\mathcal{R}^v \bowtie \mathcal{S}^v$, and sends the join results to the query sink. If n_f does not have enough memory space, it requests its neighbors to help in the join operation.

5   Experiments

In this section, we evaluate the performance of synopsis join strategy and compare it with other general join strategies, i.e., naive join, sequential join, and centroid join. Throughout the experiments, performance is measured by the total number of messages incurred for each join strategy. The control messages for synchronization and coordination among the sensors are negligible compared to the heavy data traffic caused by large tables. More realistic simulation and experiments will be included in our future work.

We varied the following parameters: join selectivity, network density, node memory capacity and synopsis size. Join selectivity δ is defined as $|\mathcal{R} \bowtie \mathcal{S}|/(|\mathcal{R}| \cdot |\mathcal{S}|)$. The join attribute values are uniformly distributed within the domain of the attribute. Network density affects the number of neighboring nodes within the communication range of a sensor node. We varied the communication radius of the sensors to achieve different network densities. The size of the synopsis is determined by the data width of join attribute. If the synopsis size is small, the number of messages needed for routing the synopses to the synopsis join nodes becomes small. If it is large, we expect a high communication overhead incurred due to the transmission of synopses.

Experiment Setup. We created a simulation environment with $10,000$ sensor nodes uniformly placed in a 100×100 grid. Each grid contains one sensor node located at the center of the grid. The sink is located at the right-top corner of the area, with coordinates $(0.5, 0.5)$. The regions R and S are located at the bottom-right and bottom-left corners of the network region, respectively, each covering 870 sensor nodes. Table $\mathcal{R}$ consists of 2000 tuples, while $\mathcal{S}$ consists of 1000 tuples. $\mathcal{R}$ and $\mathcal{S}$ tuples are uniformly distributed in R and S, respectively.

We assume a dense network with GPSR as the routing protocol. The number of hops required to route a message from a source node to a destination node is approximated using the distance between the two sensors and the communication radius. The simplification enables analysis of network traffic under ideal conditions where there is no message loss. In addition, the overhead of GPSR perimeter mode is avoided with the assumption of dense network. Simulations and experiments under real conditions using GPSR are part of our future work. We set a message size of 40 bytes, which is equal to the size of a data tuple. A tuple in the join result is 80 bytes since it is a concatenation of two data tuples.

Join Strategies. We evaluate and compare the performances of five different join strategies, namely, naive join, centroid join, sequential join, optimal join, and synopsis join.

The optimal join provides a lower bound on the total communication cost involved in the join operation. It assumes that the query sink has a complete knowledge about the distribution of $\mathcal{R}$ and $\mathcal{S}$. Hence, unlike centroid join, only candidate tuples are transmitted for the final join at N_F. Similar to the final join phase of the synopsis join strategy, for each join attribute value a, an optimal node n_f is selected such that the total cost of routing $\mathcal{R}^a$ and $\mathcal{S}^a$ to n_f and routing the result $\mathcal{R}^a \bowtie \mathcal{S}^a$ is minimized. The cost is expressed as in Equation 14. Since for any join strategy, the transmissions of candidate tuples and the join results cannot be avoided, the optimal join provides a lower bound on the total number of messages. Note that the assumption is impractical in real environment.

$$C_{\text{optimal–join}} = \sum_{n_f \in N_F} (|\mathcal{R}^a| \cdot \text{dist}(R, n_f) + |\mathcal{S}^a| \cdot \text{dist}(S, n_f) +$$
$$|\mathcal{R}^a \bowtie \mathcal{S}^a| \cdot \text{dist}(n_f, \text{sink})) \ . \tag{14}$$

5.1 Performance Evaluation

Performance vs. Join Selectivity. Figure 4(a) shows the total communication cost for different join selectivities while keeping the memory capacity, communication radius and synopsis size fixed at 250×40 bytes, 2 units and 10 bytes respectively. As shown in the figure, sequential join performs worse than all others due to the high cost of broadcasting $\mathcal{S}$ to all nodes in N_R. Therefore we exclude it from subsequent experiments. As expected, optimal join outperforms all other strategies for all selectivities. When selectivity is lower than 0.001, synopsis join outperforms naive join and centroid join. This is because non-candidate tuples can be determined in the synopsis join stage, and only a small portion of
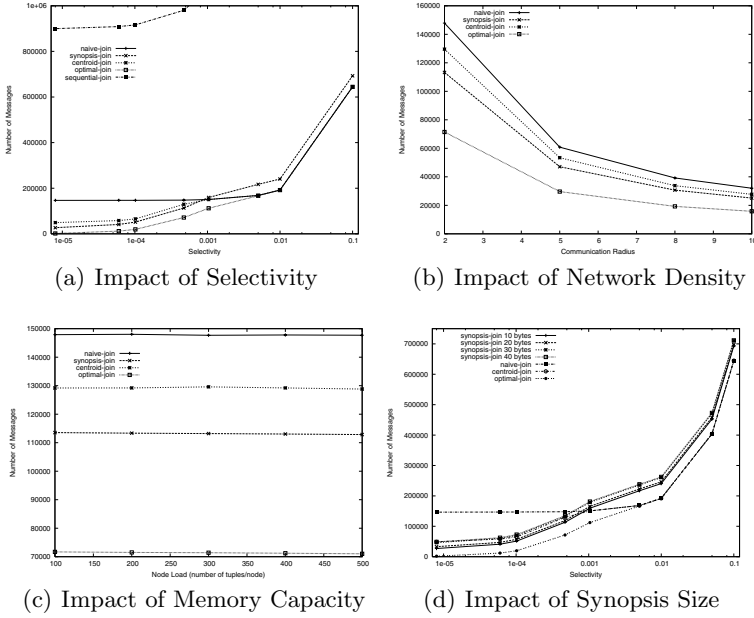
(a) Impact of Selectivity

(b) Impact of Network Density

(c) Impact of Memory Capacity

(d) Impact of Synopsis Size

Fig. 4. Experimental Results

data are transmitted during the final join. On the other hand, when selectivity is high, almost all data tuples are involved in the result. With large join result sizes, the final join nodes are centered around the sink. This explains why naive, centroid and optimal joins have the same communication cost. Moreover, synopsis join incurs unnecessary communication sending the synopses, making it less desirable for high selectivity joins.

Although there is an overhead of using synopsis join when selectivity is high, it accounts for a small portion of the total communication cost. The overhead when selectivity is 0.1 is only 7%. Only when the selectivity is 0.005 and 0.01, the synopsis overhead accounts for a significant portion (20% ∼ 30%) of the total cost. Many queries have small selectivities where synopsis join is more suitable. Consider our BEJ query example in Section 1, the BEJ query joining on the `Vehicle-ID` attribute has a maximum join selectivity of 0.0005, which favors the synopsis join.

Impact of Network Density. Figure 4(b) shows the scalability of the join strategies with varied network density. In this experiment, sensors have a memory capacity of 250×40 bytes. The join selectivity and synopsis size is 0.0001 and 10 bytes, respectively. As the network becomes denser, the total communication costs for all strategies reduce too. This is expected because with a larger communication range, fewer hops are needed to send a message across the network.

Impact of Memory Capacity. Figure 4(c) shows the total communication cost with different memory capacities. In this experiment, the communication

radius is 2. The synopsis size is 10 bytes, and the selectivity is 0.0001. It is shown that the communication costs of all strategies do not change much when the memory capacity increases. The change in the memory capacity only affects the number of join nodes (and the number of synopsis join nodes for synopsis join). When the memory capacity is larger, there are fewer join nodes selected (8 join nodes reduced to 1 in our experiment setup), and fewer messages are required for sending the result tuples to the sink. There is no reduction in the communication cost of sending data from R and S to the join nodes. Therefore we cannot see much reduction in the total communication cost.

Impact of Synopsis Size. Figure 4(d) shows the total communication cost with varied synopsis sizes and join selectivities. The memory capacity is 250×40 bytes. And the communication radius is 2 units. As shown in Figure 4(d), with the experiment setup, the smaller the synopsis size, the better the performance of the synopsis join. Small synopses results in lower communication overhead during the synopsis join stage. Therefore, it is beneficial for synopses with small join attribute width compared to the data tuple size. We also observe that the synopsis join performs slightly worse than the centroid join when the synopsis size is greater than 30 bytes, indicating that the overhead of sending the synopsis is greater than the cost savings in data tuple transmission.

6   Related Work

The popular aggregation-tree-based techniques for solving in-network aggregate queries [5, 6, 12] typically use an aggregation tree to progressively reduce data by merging partial results from child nodes so as to generate new results. The same data reduction technique cannot be directly applied to in-network join queries

Several solutions have been proposed to handle joins in sensor networks. TinyDB [13] supports only simple joins in a local node, or between a node and the global data stream. Join operations across arbitrary pairs of sensors are not supported. Chowdhary et al. [9] proposed a path-join algorithm to select an optimal set of join nodes to minimize transmission cost involved in the join. Our technique differs from path-join by pre-filtering non-candidate tuples using synopsis join. Ahmad et al. [14] proposed a join algorithm by utilizing the data and space locality in a network. Their focus is on optimizing the output delay instead of the communication cost. Recently Abadi et al. [15] designed techniques to perform event detection using distributed joins. The technique joins sensor data with external static tables, and does not address the problem of joining in-network sensor readings. Also related is the work from Bonfils et al. [16] addressing the problem of optimal operator placement in a sensor network. The join is limited on only a single node, which is prohibitive for large data tables.

7   Conclusions

In this paper, we present a synopsis join strategy for efficient processing of BEJ queries in sensor networks. Unlike the general strategies, the synopsis join strat-

egy executes a synopsis join step before performing a final join. The synopsis join step joins synopses generated by the sensors to filter out non-candidate data tuples and avoid unnecessary data transmission. As part of the synopsis join strategy, we have developed methods for determining the optimal set of synopsis join nodes and final join nodes. We have also performed cost analysis on synopsis join. Our preliminary experiments have shown that synopsis join performs well for joins with low selectivity and does not incur much overheads for high join selectivity.

References

1. Mainwaring, A., Culler, D., Polastre, J., Szewczyk, R., Anderson, J.: Wireless sensor networks for habitat monitoring. In: Proceedings of WSNA'02. (2002)
2. Estrin, D., Govindan, R., Heidemann, J.S., Kumar, S.: Next century challenges: Scalable coordination in sensor networks. In: Proceedings of MobiCom. (1999)
3. Estrin, D., Govindan, R., Heidemann, J.S., eds.: Special Issue on Embedding the Internet, Communications of the ACM. Volume 43. (2000)
4. Bonnet, P., Gehrke, J.E., Seshadri, P.: Towards sensor database systems. In: Proceedings of MDM, Hong Kong (2001)
5. Madden, S., Franklin, M.J., Hellerstein, J.M., Hong, W.: TAG: A Tiny AGgregation service for ad-hoc sensor networks. In: Proceedings of OSDI'02. (2002)
6. Yao, Y., Gehrke, J.E.: The cougar approach to in-network query processing in sensor networks. SIGMOD Record **31**(3) (2002) 9–18
7. Karp, B., Kung, H.T.: GPSR: Greedy perimeter stateless routing for wireless networks. In: Proceedings of MobiComm'00, Boston, USA (2000)
8. Lu, H., Carey, M.J.: Some experimental results on distributed join algorithms in a local network. In: Proceedings of VLDB, Stockholm, Sweden (1985)
9. Chowdhary, V., Gupta, H.: Communication-efficient implementation of join in sensor networks. In: Proceedings of DASFAA, Beijing, China (2005)
10. Ratnasamy, S., Karp, B., Li, Y., Yu, F., Estrin, D., Govindan, R., Shenker, S.: GHT: A geographic hash table for data-centric storage. In: Proceedings of WSNA'03, Atlanta, USA (2002) 56–67
11. Greenberg, I., Robertello, R.A.: The three factory problem. Mathematics Magazine **38**(2) (1965) 67–72
12. Nath, S., Gibbons, P.B., Seshan, S., Anderson, Z.R.: Synopsis diffusion for robust aggregation in sensor networks. In: Proceedings of SenSys '04, ACM Press (2004)
13. Madden, S.: The Design and Evaluation of a Query Processing Architecture for Sensor Networks. PhD thesis, UC Berkeley (2003)
14. Ahmad, Y., U.Cetintemel, Jannotti, J., Zgolinski, A.: Locality aware networked join evaluation. In: Proceedings of NetDB'05. (2005)
15. Abadi, D., Madden, S., Lindner, W.: Reed: Robust, efficient filtering and event detection in sensor networks. In: Proceedings of VLDB. (2005)
16. Bonfils, B.J., Bonnet, P.: Adaptive and decentralized operator placement for in-network query processing. In: Proceedings of IPSN. (2003)