Optimizing Network-Side Queries With Timestamp-Join In Wireless Sensor Networks

Giuseppe Amato *, Stefano Chessa *†, Claudio Vairo *†
*ISTI-CNR, Pisa Research Area, Via G.Moruzzi 1, 56124 Pisa, Italy
†Computer Science Department, University of Pisa, Largo B. Pontecorvo 3, 56127 Pisa, Italy

***Abstract*—This paper proposes a new method for optimizing in-network distributed queries that perform join of data produced simultaneously by different sensors in a wireless sensor network. We adopt a modified version of the standard join operator that relates tuples having the same timestamp, and an optimized version of it, which provides on-demand, pull-mode, data acquisition from sensors. The optimizer uses an algebraic approach based on transformation rules and ordering of operators to generate and chose a query plan that reduces the query execution cost in terms of consumed energy. We implemented these join operations in a query processor for mica-class sensors and we performed extensive tests to prove that our approach may reduce energy required to process a long running query, by order of magnitudes, with respect to non optimized query plans.**

I. Introduction

Recent proposals [4], [11], [15] suggest the use of database paradigms and query languages (generally SQL-like), to interact with Wireless Sensor Networks (WSN) [5]. In a traditional database system, queries are used to search for data contained in a persistent storage repository. In a WSN, the data base consists of the environmental data that can be measured/acquired by the transducers available on the sensor nodes. Queries instruct sensor nodes on the management, filtering, and processing of the data acquired from the environment. Environmental data is acquired by the transducers of the sensor nodes when needed, in accordance with the query that the network is processing. A new data is thus available every time a transducer is activated.

Differently than in conventional databases, data acquired by WSN are meaningful as long as they can be associated to their time and place of acquisition, and the aggregation and processing of data produced (almost) simultaneously or in specific places is of great practical interest [9]. In WSN this can be achieved by expressing queries that join data having the same timestamp or specific geographical coordinates. However conventional join operator is not optimized to this purpose. For this reason we reconsider the conventional join operator and analyse two variants that relate tuples having the same timestamp (called *timestamp-join*), and an optimized version of timestamp-join (called *sync-join*). More specifically we focus on the optimization of such join operators by defining rules aimed at producing query plans optimized from the point of view of energy efficiency of the query. We show that our optimization approach may reduce the energy consumed by the WSN by order of magnitude with respect to not optimized query plans. Following these results we have also implemented the join operators and the optimization rules in the MaD-WiSe system [2], that allows the interaction with a TinyOS-based [3] WSN by mean of SQL-like queries. The MaD-WiSe query parser and optimizer are implemented in Java and run on a standard PC.

II. A Model for Query Processing in WSN

Periodic monitoring of the environment is of great practical interest and it has been widely studied [10], [12]. With periodic monitoring the data are acquired by the sensor nodes with a given (constant) rate, thus also the data processing and aggregation operations (that depends on the availability of fresh data) are executed periodically. In WSN organized according to the database paradigm, periodic monitoring can be easily expressed in form of an SQL-Like query.

Normally a query execution strategy is expressed in form of a query plan, represented as a tree where the nodes are the operators of the query algebra and edges between nodes represent the stream of data tuples from one operator to another one. A single query is executed exploiting the cooperation of several sensor nodes. Thus the operators of a query plan are distributed across several sensor nodes. We call $local$ streams those that connect operators assigned and executed on the same sensor node. Otherwise, if two operators are executed on different sensor nodes, we call $remote$ stream the steam connecting the two operators. Finally we call $sensor$ streams those that transfer tuples of data acquired by the transducers to the associated operators. Sensor streams are used to sample environmental data and they can be activated periodically or on-demand. In *periodic update* mode the associated transducer is activated to acquire a tuple at a fixed rate. The interval between two consecutive transducers activations is called *sampling period*. In *on-demand update* mode the associated transducer is activated as a consequence of a read request on the stream that causes a tuple to be acquired. This mode can be used to obtain transducers readings only under specific conditions (see *sync-join* operator below).

According to the relational algebra [7], we have basic operators for performing *selection*, *projection*, and *union* of tuples that traverse streams. In addition, we provide a special definition for the *spatial aggregation* (used to aggregate tuples produced by different streams), and for the *temporal aggregation* (used to aggregate data that arrive sequentially in

a stream). We also provide two modified versions of the join operator, defined as follows.

The *timestamp-join* operator, $\bowtie (S_1, S_2)$, takes two input streams, S_1 and S_2, and returns one output stream where tuples of S_1 and S_2 with the same timestamp are joined to create the output tuple. Here, if several tuples arrive from one stream (say S_1) before a tuple arrives from the other (say S_2), at most one tuple for S_1 is used, while all the others are discarded. This means that the transducer activations that generate the discarded tuples were useless.

On the other hand, in the *sync-join* operator, $\bowtie_{sync} (S_1, S_2)$, S_2 must be an on-demand stream and it is activated only when a tuple arrives on S_1. This avoids useless transducer activations on the on-demand stream, thus saving energy. The sync-join operator combines both the push and the pull sensing techniques and has a master-slave behavior: stream S_1 is the master and the slave stream S_2 is read only when a tuple is received from the master. In this case the sensor node in which is instantiated the slave stream, has to be the sensor node that executes the join operator.

A query can be represented by several equivalent query plans. For example, Figure 1 shows three possible (and equivalent) query plans for the query in Table III. Boxes in figures represent sensor nodes with the operators they execute. Each of the proposed query plan correctly processes the query, but it has a different cost, in term of energy consumption, as reported in Table IV. The objective of this paper is to define optimization rules used by the query optimizer to generate different query plans and choose the one that reduces the query execution, according to the cost model described in the next section.

III. The cost model

The cost of a query plan is the power P needed to process the query plan in terms of energy E consumed per unit of time ($P = E/t$). We estimate the cost in terms of the energy required to send records across streams because the cost required by an operator to process a data is negligible with respect to the cost of sending data in a stream.

Let $E(S, s)$ be the energy required to send a single record of size s across the stream S and $f(S)$ be the frequency of records traversing the stream S. The cost of stream S (that is the energy required to transmit data along the stream), is $P(S) = f(S)E(S, s)$. The cost of a query plan, say QP, is the sum of the cost of its streams. Let $\mathbf{S}$ be the set of streams contained in the query plan QP. The cost of QP is $P(QP) = \sum_{S \in \mathbf{S}} P(S)$.

In order to evaluate the previous expressions, we need to know 1) the energy required to send and receive a record across each stream and 2) the frequency of records that traverse each stream. We will discuss these issues in the following.

Energy Required for Sending a Record - The energy required to send and receive a record s across a stream depends on the type of stream considered (sensor, local, remote).

Records of sensor streams contain the timestamp and the read value. The cost of a record that traverses this stream solely depends on the transducer used. Given a sensor stream SS associated with transducer TR, we have $E(SS, s) = energy\_per\_sample(TR)$. Table I shows the energy consumption for a single sample from various transducers of the sensor board MTS, all produced by Crossobow [1].

Transducer	Energy per Sample (mJ)
Thermistor	0.0000891
Accelerometer	0.03222
Magnetometer	0.2685

TABLE I
ENERGY REQUIRED FOR A SAMPLE FROM VARIOUS TRANSDUCERS OF MTS310CA BOARD.

For what concerns local streams the energy required to send a record in main memory is negligible with respect to the cost incurred by the other types of stream. Thus we can reliably consider a zero cost in this case. Given a local stream LS we have $E(LS, s) = 0$.

Transmission of a tuple along a remote stream requires that the nodes involved in the corresponding multi-hop path collaborate to forward the tuple toward the destination node. The transmission cost includes the cost payed by the sender, the cost payed by the receiver, and the cost payed by the internal nodes to forward the message. Let $E_t(s)$ be the energy required for transmitting a record of size s over the radio interface, and $E_r(s)$ be the energy required for receiving it. The energy required to send a record of size s over a remote stream RS along a n hops path can be approximated by $E(RS, s) = n(E_t(s) + E_r(s))$.

Given that in many real cases the number of hops between two nodes is proportional to the distance between the two nodes [8], we can express the energy needed to send a record of size s across a remote stream S, where source and destination nodes have distance d as $E(S, s) = d \cdot c \cdot (E_t(s) + E_s(s))$, where c is a tuning parameter that depends on the density and transmission range of the nodes in the network.

The energy required to send and receive a 50 bytes packet by the MICAz platform [1] is respectively 0.1494225 mJ and 0.161445 mJ.

Frequency of Records in Streams - The frequency of records across streams depends on the periodicity of the data acquisition of the sensor streams, and on the specific operators used to connect streams.

In a periodic sensor stream S_p with period p data are acquired and sent with frequency $f(S_p) = 1/p$. An on-demand sensor stream is used as input to a sync-join operator, $\bowtie_{sync} (S, S_{od})$, where S_{od}, is the on-demand sensor stream. In this case we have that $f(S_{od}) = f(S)$ since a record is requested from S_{od} every time a record arrives from S.

The frequency of local and remote streams depends on the operators that write in the streams and on the stream(s) where that operators read. The frequencies of the streams in output from the main operators ($\pi, \sigma, \bowtie$) are summarized in Table II.

Operator	$f(S_O)$
$S_O \leftarrow \pi\_(S_I)$	$f(S_I)$
$S_O \leftarrow \sigma_{pred}(S_I)$	$f(S_I) \cdot \mathtt{Pr}(pred = true)$
$S_O \leftarrow\!\bowtie (S_{I_1}, S_{I_2})$	$\min\{f(S_{I_1}), f(S_{I_2})\}$

TABLE II
FREQUENCY FOR LOCAL AND REMOTE STREAMS.

SELECT *
FROM 1.Magnetism, 2.Acceleration, 3.Temperature
WHERE p_1(1.Magnetism) **and** p_2(2.Acceleration) **and** p_3(3.Temperature)
EVERY 1000

TABLE III
QUERY USED FOR THE QUERY OPTIMIZATION EXAMPLE.

IV. QUERY OPTIMIZATION

Several transformation rules proposed in the literature to optimize traditional database query execution can be applied in our context. For instances rules to push-down selections and projections, and selectivity-based ordering of selections are very useful since they contribute to reduce the amount of data to be transferred upward in a query plan. This implicitly reduce the amount of data traversing remote streams, and, in turn, it reduces the amount of radio activity and of energy consumed.

Here we introduce new transformation rules that are particularly useful in our context since they make optimal use of the data model and of the operators that we have defined. Specifically we consider rules according to the following guidelines (we do not report a formal definition of the rules, given to the lack of space):

1) Sync-join and on-demand streams should be used whenever possible. In particular, if a periodic sensor stream is on the right side of a join, the join can be transformed into a sync-join and the sensor stream into an on-demand sensor stream (SJ rule). If there are some unary operators between the sensor stream and the join, the unary operators can be moved on top of the join and the transformation can still take place. Note that even if the unary operators are moved up (contrarily to the traditional push-down strategies), this is not a problem, given that the sensor stream is activated only if needed.

2) Given that a sync-join requires a sensor stream on the right side, trees representing query plans should be unbalanced to the left (Left Deep Join Trees, LDJT rule). In this way, the chance that a sensor stream (a leaf node) is found as the right argument of a join is increased.

3) Unary operators such as selections, projections, and temporal aggregates (which reduce the amount of data being forwarded) should be moved on the node where data is acquired (Push-Down, PD rule).

The optimizer, given a query plan, executes three sequential steps:

1) Tries to heuristically use the transformation rules that apply to it, until there are no transformation rules that can be applied.

2) Performs an operators re-ordering, according to selectivity of predicates, cost of acquisition, and topology criteria.

3) Evaluates the query plans obtained, according to the cost model defined above, and chooses the query plan that consumes less energy.

Query Optimization Example - Let us consider the query showed in Table III where p_1, p_2, and p_3 are some predicates on magnetism, acceleration and temperature readings, respectively, with probability $\mathtt{Pr}(p_1) = 0.01, \mathtt{Pr}(p_2) = 0.05, \mathtt{Pr}(p_3) = 0.1$, respectively.

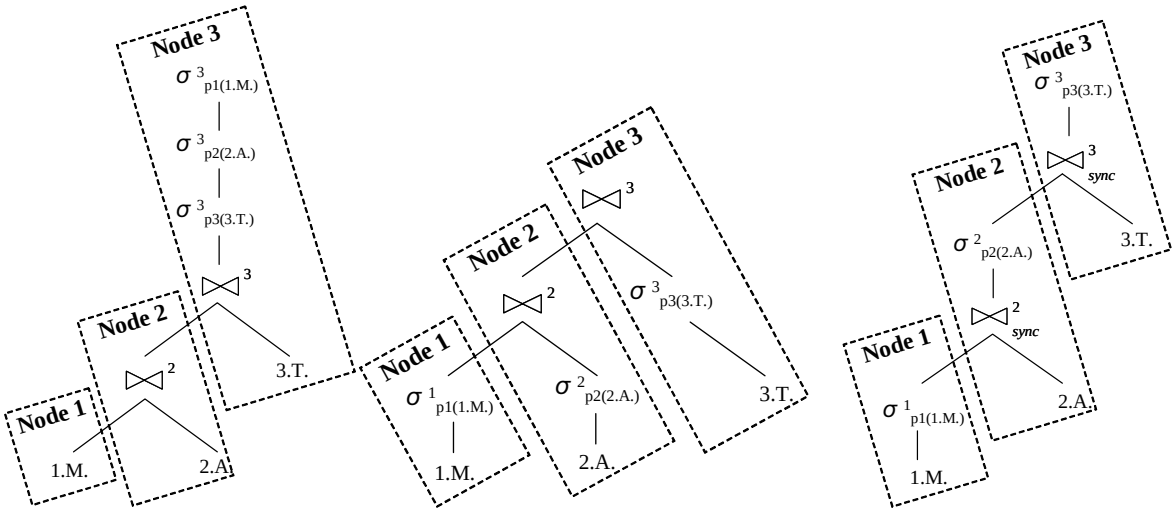
Figure 1 shows three possible equivalent query plans that can be used to process the above query. We assume that the sink is an external node connected to node 3. QP1, on the left, is obtained by applying the LDJT rule. It first acquires all specified data and then joins them before applying the three selections on the last node. This requires that all magnetism readings be sent to Node 2 and joined with the acceleration readings. The result of the join is sent to Node 3 where it is joined with the temperature reading and then the three selections are applied. QP2, in the middle, is obtained from QP1 by using the PD rule. In this query plan all data must be acquired. However, magnetism is sent to Node 2 only if it satisfies p_1. The join on Node 2 is thus executed only if both p_1 and p_2 are satisfied and in this case the result is sent to Node 3. The join in Node 3 is executed only if all three predicates are true, and in this case the result is sent to the sink. QP3, on the right, is obtained from QP2 by using the SJ rule. In this case, magnetism is always acquired. It is sent to Node 2 if it satisfies predicate p_1 and in this case the acceleration is also acquired and joined with the magnetism. If p_2 is satisfied, the result is sent to Node 3 and temperature is acquired. If p_3 is satisfied the result is eventually sent to the sink.

The costs of different query plans are reported in Table IV. The lower cost of QP2 with respect to QP1 is due to the reduced number of communications required. The lower cost of QP3 with respect to QP2 is due to the combined reduction of communications and acquisitions. In this simple example, the improvement of QP3 with respect to QP2 is limited. However, we will show that the use of sync-joins, as produced for QP3, with appropriate ordering of operators can provide significant performance improvements in more general cases.

Ordering of Operators - Several equivalent query plans that maintain the same overall structure can be obtained by changing the order of the operators in a tree. Here we consider three different ordering criteria. Operators can be ordered so that:

- more selective selections are pushed down in the tree (criterion S);
- less expensive transducers are pushed down in the tree (criterion P);
- short range communications are given priority (topological ordering, criterion T);

The first criterion give precedence to very selective predicates to filter immediately useless data, thus reducing communications and data acquisitions by means of sync-joins. The second



(a) Query Plan QP1 (b) Query Plan QP2 (c) Query Plan QP3

Fig. 1. Three possible execution plans for the same query. Boxes represent sensor nodes, the superscript beside each operator represents the sensor node in which the operator is executed.

		QP1:		QP2:		QP3:	
Action	**Energy(mJ)**	**Freq.**	**Power**	**Freq.**	**Power**	**Freq.**	**Power**
Acquire M.	0.2685	1	0.2685	1	0.2685	1	0.2685
Send M.	0.31087	1	0.31087	0.01	0.00310	0.01	0.0031
Acquire A.	0.03222	1	0.03222	1	0.03222	0.01	0.00032
Send M.A.	0.31087	1	0.31087	0.0005	0.00016	0.0005	0.00016
Acquire T.	0.00009	1	0.00009	1	0.00009	0.0005	4.46E-08
Send M.A.T.	0.31087	5.0E-5	1.55E-05	5.0E-5	1.55E-05	5.0E-5	1.55E-05
Total Cost:			0.92256		0.30408		0.2721

TABLE IV
COSTS OF THE THREE EXECUTIONS PLANS IN FIGURE 1.

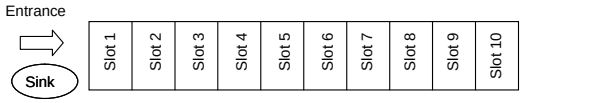
criterion gives precedence to low cost acquisitions. High cost acquisitions are thus executed with low probability since they are high in the tree, and the data collected at the lower levels of the tree must pass the selections first. The third criterion reduces the communication costs by choosing an ordering of the operators and their allocation to the nodes such that the multi-hop communication paths are shortened.

Figure 2 shows the different query plans obtained by applying these criteria. Differently from the previous section, multi-hop paths are taken into account here. In QP3 operators are ordered according to criterion S, criterion P is used in QP4, and criterion T is used in QP5. The optimizer must generate different orderings according to the various criteria and choose the one providing the best performance, according to the cost model described before. In our example QP3 has cost 0.2723mJ, QP4 0.06841mJ and QP5 0.05838mJ.

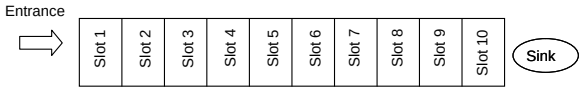
V. EXPERIMENTS

In this section we discuss the results of some empirical tests that we have performed to validate our approach.

We consider a parking lot monitoring scenario. For simplicity, we assume that the parking lot consists of a linear sequence of parking slots (see Figure 3). The entrance to the parking lot is at the beginning of the sequence of parking slots. We



(a) Scenario 1

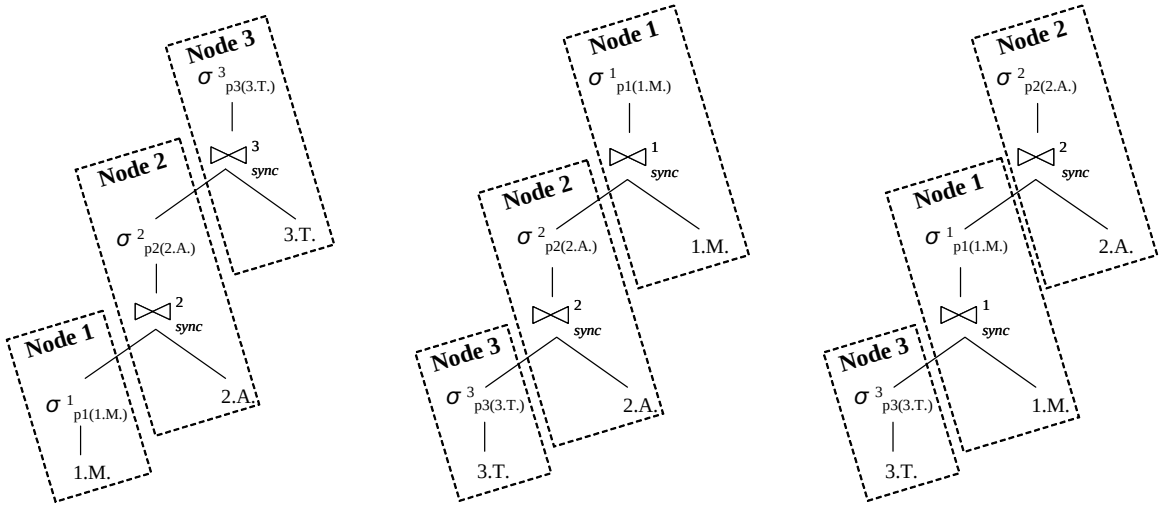


(b) Scenario 2

Fig. 3. The two scenarios considered for the experiments, in the case of 10 park lots with the Sink (the circle) placed at the entrance (Scenario 1), and at the opposite side of the entrance (Scenario 2) of the parking.

assume that under each parking slot we place a sensor with a magnetic and a light transducer. We assume that we can decide that a parking slot is occupied when both the light is below a certain threshold and the magnetic field is above a certain threshold. We want to receive an alarm from the network when the parking lot is full. This can be obtained by executing the query reported in Table V which controls every 10 seconds that all parking slots are occupied.

We assume that car drivers park in the first empty slot that they find. Therefore, the probability to find an empty



(a) Query Plan QP3 (b) Query Plan QP4 (c) Query Plan QP5

Fig. 2. Three possible execution plans for the same query using joins.

SELECT *
FROM all.Light, **all**.Magnetism
WHERE
all.Magnetism $> threshold_m$ **and**
all.Light $< threshold_l$
every 10000

TABLE V
A QUERY THAT CHECKS IF THE PARKING LOT IS FULL.

slot increases as we move away from the entrance. That is, selectivity of predicates on nodes placed in slots far from the entrance is higher than that of slots close to the entrance. For simplicity, we model the selectivity of a predicate on the i-th slot as $1 - ((numSlots - i + 1)/(numSlots + 1))$, where $1 \leq i \leq numSlots$. We also consider two different scenarios for the placement of the sink node: the sink is placed at the entrance, and the sink is placed at the opposite side from the entrance.

All possible orderings are labeled with the corresponding labels. For instance PST means that first operators are ordered according to power (P), then to selectivity (S), finally according to topology considerations (T).

We performed various tests varying the number of slots available in the parking lot from 1 to 100 according to the two described scenarios. Obtained results are reported in Figure 4 and 5, where we show separate plots for the query plan obtained by using only the Left Deep Join Tree rule (LDJT in sub-figures (a)), by using also the selection Push-Down rule (LDJT + PD in sub-figures (b)), and by using also the Sync-Join transformation rule (LDJT + PD + SJ in sub-figures (c)). Every plot reports the different orderings of the operators for each query plan. The costs are expressed in milli-watts (mW).

As expected, the only application of the LDJT rule (check QP1 in Figure 1 to infer the overall structure of the obtained query plan) returns the worst results. In fact in this case all transducers are always activated at every sensing period. Also all data are reported to the sink node that, in turn, executes the sequence of selections required by the **WHERE** clause of the query. Differences in the performance of the various orderings is due to the number of communications needed in the corresponding query plans.

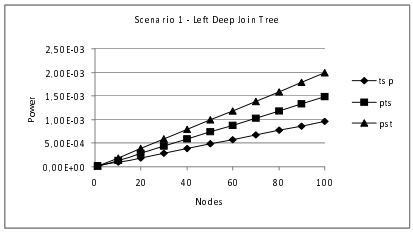
The application of the PD rule (check QP2 in Figure 1 to infer the overall structure of this query plan) in addition to the LDJT gives on average results one order of magnitude better. In this case, as before, transducers are all activated at every sensing period. However, transmissions can be reduced due to the application of the selection operators immediately after sensing. Difference between the various ordering is due basically to the capability of early filtering the packet sent in the network.

Finally the application of the SJ rule (the obtained query plan can be inferred from QP3 in Figure 1) offers the best performance providing a query execution cost more than three orders of magnitude smaller than the previous case. The SJ rule, in fact, allows also reducing the cost of acquisition in addition to the cost of transmission, thus drastically reducing the cost of executing the query.

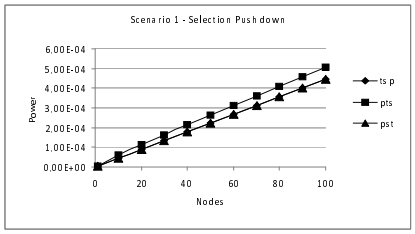
In summary, the sync-join strategy is very effective in reducing the cost of queries that relate data produced by many sensors, since it reduces both the costs of transmission and acquisition. In particular the sync-join can offer performance order of magnitude better than conventional join, provided that correct priority is given to ordering according to topology, selectivity and acquisition cost.

VI. RELATED WORK

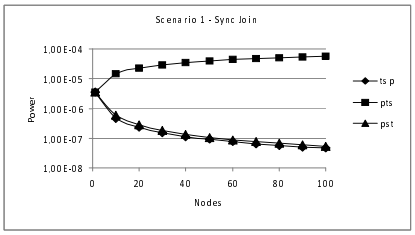
Several proposals in the literature have addressed the problem of query optimization in WSN. In particular [13] proposes a comprehensive approach to query optimization that takes into account the cost of acquiring metadata, cost of sensing and communications. In our work we do not take into consideration



(a) LDTJ

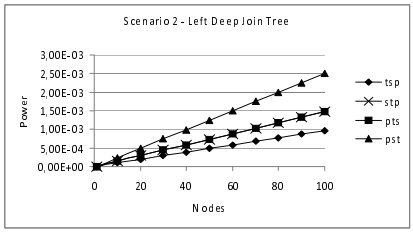


(b) LDTJ + PD

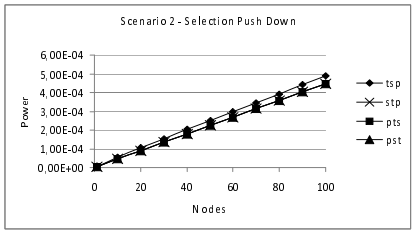


(c) LDTJ + PD + SJ

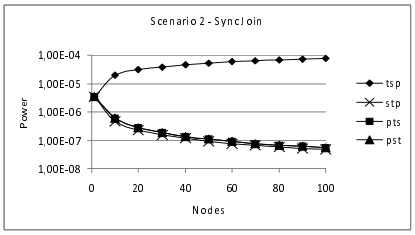
Fig. 4. Results obtained when the sink is placed at the entrance of the parking lot.



(a) LDJT



(b) LDJT + PD



(c) LDJT + PD + SJ

Fig. 5. Results obtained when the sink is placed at the opposite side of the entrance of the parking lot.

the cost and strategies for acquiring metadata, however we consider the cost of sensing and communication. In addition, given that in our framework a query is executed in a distributed fashion and that a query plan spans over several nodes, we also consider the possibility of conveniently allocate the operators (portions of query plans) on the nodes of the network. The problem of placement of the query operators in the nodes of the network was studied in [6] to minimize the transmission cost. The operations taken into acount are the correlation and aggregation of the data acquired by different nodes. The problem of placement of the join operator is also addressed in [14], where both centralized and distributed computations of the join are analyzed by using statistical methods. In our proposal operators are placed using transformation rules based on heuristics and the operators are linked together using operator ordering strategies that take into account network topology, sensing costs, and predicate selectivity.

VII. Conclusions

In this paper we presented a new method for optimizing in-network queries in WSN. In particular we adopted a modified version of the join operator (*sync-join*) that aggregates data produced almost simultaneously by different sensors. The sync-join operator exploits the pull-mode paradigm for data acquisition form the sensors, thus allowing avoiding useless data acquisitions. We implemented the proposed approach and we tested it both in a real sensor network and in a simulator. Our experiments show that the optimization strategy proposed allows reducing energy required to process queries by order of magnitude, with respect to non optimized query plans.

References

[1] Crossbow Technology Inc., http://www.xbow.com.

[2] MaD-WiSe: Management of Data in Wireless Sensor networks. http://mad-wise.isti.cnr.it.

[3] TinyOS. http://www.tinyos.net/.

[4] G. Amato, P. Baronti, and S. Chessa. MaD-WiSe: a distributed query processor for wireless sensor networks. In *ISTI-CNR Technical report ISTI-39/2006*, 2006.

[5] P. Baronti, P. Pillai, V. Chook, S. Chessa, A. Gotta, and Y. F. Hu. Wireless Sensor Networks: a survey on the state of the art and the 802.15.4 and ZigBee standards. *Computer Communications*, 30(7):1655–1695, 2007.

[6] B. J. Bonfils and P. Bonnet. Adaptive and decentralized operator placement for in-network query processing. *Telecommunication Systems*, 26(2-4):389–409, 2004.

[7] E. Codd. *A Relational Model of Data for Large Shared Data Banks*. ACM, 1970.

[8] S. De, A. Caruso, T. Chaira, and S. Chessa. Bounds on Hop Distance in Greedy Routing Approach in Wireless Ad Hoc Networks. *International Journal on Wireless and Mobile Computing*, 1(2):131–140, 2006.

[9] P. D. Felice, M. Ianni, and L. Pomante. A spatial extension of TinyDB for wireless sensor networks. In *IEEE Symposium on Computers and Communications (ISCC 2008)*, pages 1076–1082, Marrakech 2008.

[10] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: a scalable and robust communication paradigm for sensor networks. In *Proceedings of the 6th annual ACM/IEEE international conference on mobile computing and networking, Boston, MA, USA*, pages 56–67, 2000.

[11] K. Liu, L. Chen, Y. Liu, and M. Li. Robust and Efficient Aggregate query processing in wireless sensor networks. *Mobile Networks and Applications*, 13(1-2):212–227, 2008.

[12] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TinyDB: an acquisitional query processing system for sensor networks. *ACM Trans. Database Syst.*, 30(1):122–173, 2005.

[13] R. Rosemark, W.-C. Lee, and B. Urgaonkar. Optimizing energy-efficient query processing in wireless sensor networks. In *8th International Conference on Mobile Data Management (MDM 2007), Mannheim, Germany, May 7-11, 2007*, pages 24–29, 2007.

[14] M. Stern, E. Buchmann, and K. Bhm. Where in the sensor network should the join be computed, after all? In *In Proceedings of the First Ubiquitous Knowledge Discovery Workshop (UKD'08)*, September 2008.

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