Query optimization for wireless sensor network databases in the MadWise system ⋆ (Extended Abstract)

[1] Giuseppe Amato, [1,2] Paolo Baronti, and [1,2] Stefano Chessa

[1]ISTI-CNR Pisa, Italy, [2]Department of Computer Science, University of Pisa, Italy.

Abstract. We propose a comprehensive approach to distributed query processing in wireless sensor networks. In our proposal we reinterpret the classical approach to database system design according to the wireless sensor networks context, and we redefine the aspects related to the definition of a query language, data model, query algebra, and query optimization strategies. We show that our approach enables optimizations of the query plan which may reduce the costs, in terms of consumed energy, of orders of magnitude.

1 Introduction

Wireless Sensor Networks [4, 6] are composed of a set of (tiny) devices (hereafter called sensors), which are microsystems, each comprising a processor, a memory, a set of transducers, and a low-range, low-bandwidth radio transceiver. Sensors are powered by on board batteries thus their energy efficiency is critical in most applications. Applications of sensor networks include, among others, environment sampling, disaster areas monitoring,and health monitoring.

The MaD-WiSe system [8],[2] allows interaction with a wireless sensor network as a traditional database management system. In a traditional database system queries are used to search for data contained in a persistent storage repository. In a wireless sensor network, the data base consists of the environmental data that can be measured/acquired by the transducers available on the sensor nodes. Queries instruct nodes on the management, filtering, and processing of the data acquired from the environment. The wireless sensor network and the software running on the nodes are the means that allow data to be acquired when needed from the environment, exactly in the way that a traditional database software allows data to be accessed on disks. In a wireless sensor network data is not stored anywhere: environmental data is acquired by transducers of the nodes when needed, in accordance with the query that the network is being processing. A new data is available every time a transducer is activated.

The MaD-WiSe system consists of a set of modules that implement a distributed stream management system on a wireless sensor network. A part of the MaD-WiSe modules (the distributed query processor) run on the nodes of the wireless sensor network (network side) and the rest of the modules (query parser, and query optimizer) run

⋆ Work funded in part by the European Commission in the framework of the FP6 projects PERSONA (contract N. 045459) and INTERMEDIA (contract N.38419)

on a generic client node connected to the wireless sensor network. MaD-WiSe is implemented on the MICAz motes platform [1], relying on TinyOS [3]. The query parser and optimizer are implemented in Java and run on a standard PC. A sample MaD-WiSe query is the following:

SELECT roomB.Temp
FROM roomA, roomB
WHERE roomA.Temp $>$ roomB.Temp **and** roomA.Temp $>$ 50
EVERY 10000

2 About the data model and the query algebra

A natural way to imagine data processed by a wireless sensor network is that of a stream or sequence of tuples. We define three different types of streams to model the different tuple access modes: *sensor streams*, which represent streams of data acquired by the transducers, *remote streams*, which model streams of data sent by a source node to a destination node, and *local streams*, which represent streams of data generated by execution of local operations and sent as input of other local operations.

We have considered two different modes for updating the tuple of a sensor stream: i) the *periodic update mode* where the transducer is activated with a fixed period (the *sampling period*) to update the stream (this mode can be used to collect transducer readings every x seconds) and the ii) *on-demand update mode* where the transducer is activated as a consequence of a read request and causes the stream update (this mode can be used to obtain transducers readings only under specific conditions).

Remote streams support the transfer of intermediate query results between remote nodes. Note that remote streams require communication between nodes through their radio interfaces. Since radio activation is one of the primary causes of energy consumption, it should be carefully managed.

Local streams model data transfers between operators located on the same node in order to enable pipelined executions of the operations. We distinguish local from remote streams since they have different costs in terms of energy consumption. Local streams mainly consume memory resources, while their energy consumption is negligible.

Differently than the relational query algebra [7] our operators supporting the query language deal with streams of tuples according to the model defined above. These operators have a strictly pipelined behavior that avoids the use of temporary buffers for producing results.

In particular we have changed the definition of the *join* operator to adapt to the needs of data management in wireless sensor networks. This because in this context it is very useful to relate tuples generated at the same instant (or approximatively the same instant) by different transducers and/or nodes. For this reason we define a join operator, $\bowtie (S_{I_1}, S_{I_2})$, that relates tuples having the same timestamp **TS**. For every new tuple read on one of the input streams the join operator checks if the last tuple read from the other stream has the same timestamp. This operation is clearly non-blocking and its execution requires only single-position buffers. We also provide a further implementation of the join operator called *sync-join*, $\bowtie_{sync} (S_{I_1}, S_{I_2})$, where S_{I_2} is an on-demand stream. The sync-join requests the activation of S_{I_2} only when a tuple arrives on S_{I_1}.

3 The cost model

We measure the cost of a query execution plan by evaluating the power P needed to process the query 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 its power, is $P(S) = f(S)E(S, s)$. The cost of a query execution plan, say QEP, is the sum of the cost of its streams. Let $\mathbf{S}$ be the set of streams contained in the query execution plan QEP. The cost of QEP is $P(QEP) = \sum_{S \in \mathbf{S}} P(S)$.

Energy required for sending a record: The cost of sensor streams is dominated by the energy required to sample a value with the associated transducer. The cost of a local stream is that needed to store the records in the temporary buffer. The cost of a remote stream is dominated by the activity of the radio interfaces of the nodes in the multi-hop path from the source node to the destination node.

The energy required to store a record in a local stream 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 $E(LS, s) = 0$.

We suppose that the records of sensor streams have fixed size and structure, thus the cost solely depends on the transducer used. Given a sensor stream SS associated with transducer TR, we have $E(SS, s) = energy\_per\_sample(TR)$. Thermistor, Accelerometer, and Magnetometer transducers embedded in the Crossobow [1] sensor boards consume respectively 0.0000891, 0.03222, and 0.2685 mJ per sample.

In case of remote streams the situation is a bit more complex. 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.

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. Thus the energy required to send a record of size s over a remote stream RS along a n hops path can be approximated by $E(RS, s) = n(E_t(s) + E_r(s))$. In many real cases the number of hops between two nodes is proportional to the distance between the two nodes [5]. Therefore, 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 case of sensor streams, we distinguish between periodic and on-demand streams. In a periodic stream S_p with period p data are acquired and sent with frequency $f(S_p) = 1/p$. An on-demand stream is intended to be used as input to a sync-join operator as in $S_O \leftarrow\bowtie_{sync} (S, S_{od})$, where S_{od}, is the on-demand sensor stream. We have that

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

Table 1. Frequency for local and remote streams connecting various operators

$f(S_{od}) = f(S)$ given that 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 various possibilities are summarized in Table 1.

4 Query optimization

In this paper we consider an algebraic optimization approach, that is based on transformation rules transforming a query plan into a semantically equivalent one with a lower cost. The final query plan is obtained by applying successive transformations to an initial query plan built from the MW-SQL query. We will also discuss some issues related to the ordering of the operators, which can be affected by the transformation rules.

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 discuss some peculiar 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 the following guidelines:

1. Sync-join and on-demand streams should be used whenever possible.
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). 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 as close as possible to the node where data is acquired.

Transforming a join into a sync-join: According to our previous observations we define rules that transform a join into a sync-join. The idea is that 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. If there are some unary operators between the sensor stream and the join, the unary operators can be moved after the join (to process the output of the join) and the transformation can still take place. Note that

even if the unary operators are moved up, this is not a problem, given that the sensor stream is activated only if needed.

Formally the transformation rule is the following:

$$\frac{\bowtie^k (B, \xi^*(\breve{S}^h)) \qquad \text{where } \xi^* \neq \emptyset \vee k \neq h}{\widetilde{\xi^h}(\bowtie^h_{sync} (B, \breve{S}^h))} \tag{1}$$

where $\breve{S}^h$ is a sensor stream on node h, $\widetilde{\xi^h}$ is obtained from the sequence ξ^*, where each π_X is transformed to $\pi_{X \cup Attr(B)}$, and all elements are localized on Node h.

Previous rule can be easily modified to consider the case where the sensor stream is on the right side of the join.

Obtain Left Deep Join Trees: The following rule rearranges the joins in a query plan to obtain a left deep join tree, and facilitate transformation of joins into sync-joins by means of the previous rules.

$$\frac{\bowtie (A^h, \xi^*(\bowtie (B^k, C^j)))}{\widetilde{\xi^{j*}}(\bowtie^j (\bowtie^k (A^h, B^k), C^j))} \tag{2}$$

where $\widetilde{\xi^{j*}}$ is obtained from ξ^* by transforming all π_X into $\pi_{X \cup Attr(A)}$, and all elements are located on Node j.

This rule can be profitably used with standard rules to push down selections and to order joins and selections so that the most selective selections are performed first, reducing the amount of data traversing the tree.

Moving unary operators close to the source of data: Unary operators are moved close to the source of the data, as suggested by the third observation, by using the following rules in addition to traditional push-down transformation rules.

$$\frac{\pi^h_X(A^k) \quad \text{where} \quad h \neq k}{\pi^k_X(A^k)} \tag{3}$$

$$\frac{\sigma^h_c(A^k) \quad \text{where} \quad h \neq k}{\sigma^k_c(A^k)} \tag{4}$$

Query optimization example: Let us suppose that we submit the following MW-SQL query:

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

where p_1, p_2, and p_3 are some predicates on magnetism, acceleration and temperature readings, respectively, with probability $\texttt{Pr}(p_1) = 0.01, \texttt{Pr}(p_2) = 0.05, \texttt{Pr}(p_3) = 0.1$, respectively. Figure 1 shows three possible equivalent query plans that can be used to process the above query. QP1 is obtained by applying the left deep join trees rule. QP2 is obtained from QP1 by using the selections push-down rule and their allocation

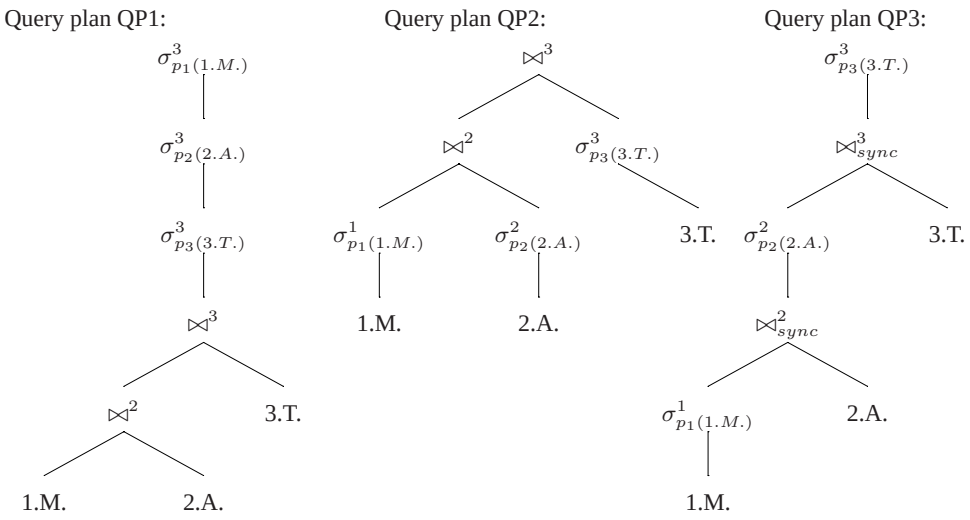

Fig. 1. Three possible execution plans for the same query.

		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 2. Costs of the three executions plans in Figure 1.

on the node where data are generated. QP3 is obtained from QP2 by using rules for transforming joins into sync-joins.

As reported in Table 2, the cost of QP2 is approximatively $1/3$ of QP1. Cost of QP3 is a slightly better than QP2. This means that the expected lifetime of a network running QP2 or QP3 is about 3 times longer than the lifetime expected when running QP1. The lower cost of QP2 with respect to QP1 is due to the reduced number of communications that it requires. The lower cost of QP3 with respect to QP2 is due to the combined reduction of communications and acquisitions.

The performance improvement of QP3 with respect to QP2 is very 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.

Ordering of operators: Here we discuss three different ordering criteria. Operators can be ordered so that i) more selective selections are pushed down in the tree; ii) less expansive transducers are pushed down in the tree; iii) short range communications are given priority (topological ordering).

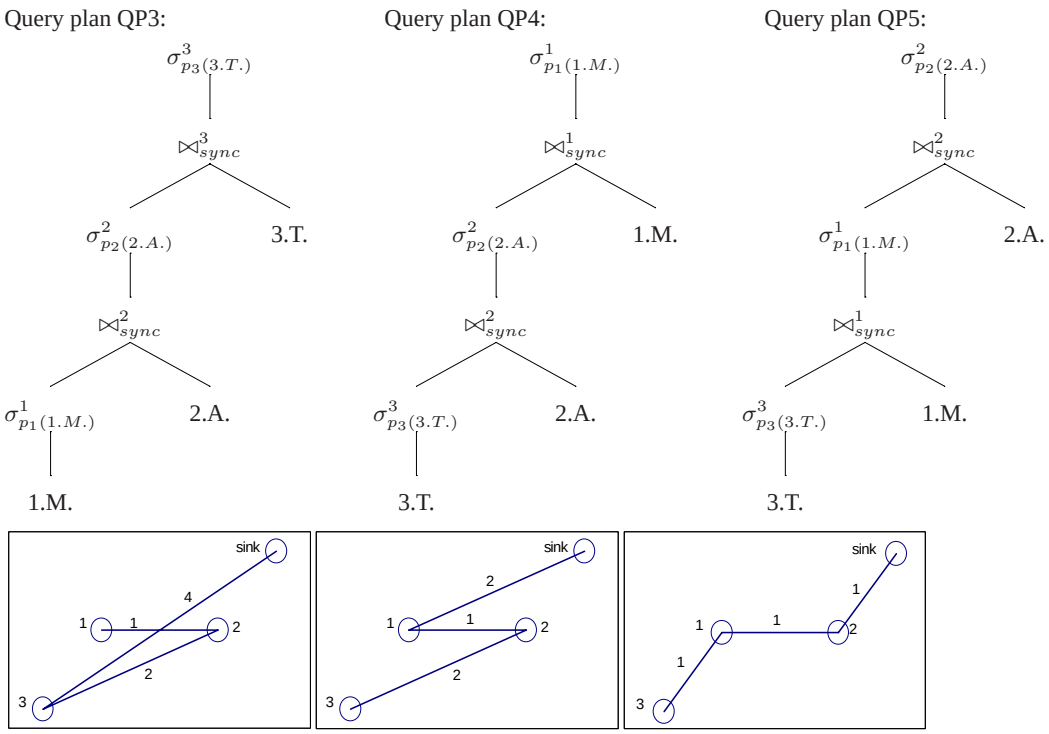

Fig. 2. Three possible execution plans for the same query using joins.

Examples of the application of these criteria are given in Figure 2. Differently from the previous section, multihop paths are taken into account here. In QP3 operators are ordered according to criterion i), criterion ii) is used in QP4, and criterion iii) is used in QP5. Their costs are given in Table 3. The cost of QP4 (0.068 mJ) is one order of magnitude smaller than the cost of QP3 (0.27 mJ). The cost of QP5 (0.058 mJ) is slightly smaller than the cost of QP4. Therefore, the expected lifetime of a network running QP4 or QP5 is about 5 times longer than running QP3.

However, this is not a proof that ordering according to the topology of the network is always the best solution. The results can vary depending on the selections selectivity and on the acquisitions costs. In general, there is not a best ordering strategy. The optimizer must generate different orderings according to the various criteria and choose the one providing the best performance. As shown in our example, this may lead to performance improvements of orders of magnitude.

5 Conclusions

In this paper we have presented a comprehensive and consistent approach to query processing in wireless sensor networks. In particular we have defined, analyzed, and discussed the aspects related to data modeling, query algebra, and query optimization.

QP3:				**QP4:**			
Action	**Energy(mJ)**	**Freq.**	**Power**	**Action**	**Energy(mJ)**	**Freq.**	**Power**
Acquire M.	0.2685	1	0.2685	Acquire T.	0.00009	1	0.00009
Send M.	0.31087	0.01	0.00311	Send T.	0.62174	0.1	0.06217
Acquire A.	0.03222	0.01	0.00032	Acquire A.	0.03222	0.1	0.00322
Send M., A.	0.62174	0.0005	0.00031	Send T., A.	0.31087	0.005	0.00155
Acquire T.	0.00009	0.0005	4.46E-08	Acquire M.	0.2685	0.005	0.00134
Send M., A., T.	1.24347	0.00005	6.21E-05	Send T., A., M.	0.62174	0.00005	3.11E-05
Total Cost:			0.2723	**Total Cost:**			0.06841

QP5:			
Action	**Energy(mJ)**	**Freq.**	**Power**
Acquire T.	0.00009	1	0.00009
Send T.	0.31087	0.1	0.03109
Acquire M.	0.2685	0.1	0.02685
Send T., M.	0.31087	0.001	0.00031
Acquire A.	0.03222	0.001	0.00003
Send T., M., A.	0.31087	0.00005	1.55E-05
Total Cost:			0.05838

Table 3. Cost of the query plans QP3, QP4, and QP5.

Our approach offers many opportunities for query optimization according to the network topology, data statistics, and types of transducers. We show that accurate query optimization may provide a reduction of the query execution cost of some orders of magnitude. Our approach separates the aspects of communication, data acquisition, data representation, and data processing, and it gives the opportunity to experiment new strategies related to each of these aspects without affecting the entire system design.

References

1. Crossbow Technology Inc., http://www.xbow.com.
2. MaD-WiSe: Management of Data in Wireless Sensor networks. http://mad-wise.isit.cnr.it.
3. TinyOS. http://www.tinyos.net/.
4. I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. A survey on sensor networks. *IEEE Communication Magazine*, 40(8):102–114, 2002.
5. C. Antonio, C. Tamalika, and C. Stefano. Bounds on Hop Distance in Greedy Routing Approach in Wireless Ad Hoc Networks. *International Journal on Wireless and Mobile Computing*. To appear.
6. 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. Techical Report ISTI-2006-TR-18, ISTI-CNR, 2006. http://dienst.isti.cnr.it/.
7. E. F. Codd. A relational model of data for large shared data banks. *Commun. ACM*, 13(6):377–387, 1970.
8. ISTI-CNR, Via G. Moruzzi, 1, 56124, Pisa, IT. *SensorViz/MaD-WiSe*, version 1.3 edition, July 2006. http://mad-wise.isti.cnr.it/manual_13.pdf.