Optimizing Energy-Efficient Query Processing in Sensor Networks

Ross Rosemark Bhuvan Urgaonkar Wang-Chien Lee Daniel Lee
 Department of Computer Science and Engineering
 The Pennsylvania State University
 University Park, PA 16802, USA
 {rosemark, bhuvan, wlee, djlee}@cse.psu.edu

Abstract—Query processing in sensor networks has recently received significant attention from the research community. Processing queries in an energy constrained manner is a crucial challenge as nodes are typically energy-deficient. In this paper, we present a query optimizer for sensor networks. Central to the working of our query optimizer is a cost-based analysis to estimate the overall energy consumption associated with a given query plan over the lifetime of a query. By utilizing this estimation technique, our query optimizer compares the desirability of several candidate plans for a given query and chooses a plan that is likely to utilize the least energy for execution. Simulation results show that the query plan chosen by our query optimizer consumes significantly less energy than our simulation of the state-of-the-art query processing engine for sensor networks.

I. INTRODUCTION

Query processing systems designed for sensor networks must incorporate energy awareness in order to extend the lifetime of the nodes and the network [1, 2, 3, 4, 5, 6, 7, 8]. In sensor networks equipped with query processing engines, e.g., TinyDB [1] and Cougar [2], a user injects (in an extended SQL) a query at the access point (AP). A query issued in such environment typically specifies: 1) a set of predicates, 2) *epoch* (interval between two successive executions of the query at the nodes involved), and 3) *duration* (lifetime of the query). A sensor node has one or more sensing devices or sensors.¹ For simplicity, each predicate in a query is associated with a distinct sensor. To process a query, a node samples its sensors and checks if resulting readings satisfy the relevant predicates.

In terms of query processing, the following procedures consume energy: 1) sampling the sensors at the nodes and 2) routing collected results from nodes to the AP.

To facilitate query optimization, the AP may choose to collect *metadata* from the nodes participating in the query. Briefly, metadata provides statistics describing the distribution of the values generated by the sensors at a given node. This metadata can be utilized to improve the energy consumed by sampling the sensors and routing the results to the AP.

The order in which a node samples its sensors, conventionally referred to as a *query plan*, can be a crucial factor affecting the energy consumed by the sensor network. Therefore, such orderings for the various nodes involved in a query are an essential part of the query plan. Metadata can be utilized by the AP to determine energy-efficient query plans for the nodes participating in the query. It is important to note that the cost of determining the optimal query plan for a node depends on the complexity of the query. While for simpler queries, a node may itself be able to derive the optimal query plan by spending a small amount of energy/memory, for complex queries, it might be desirable to delegate this task to the energy/memory-rich AP. To give an idea, for a query with X predicates there exists $X!$ query plans. This means that for even a small query that contains 10 predicates there exists 3628800 query plans. Due to memory/energy constraints of nodes, in this paper, we assume that the query plans are determined by the AP (same as TinyDB).²

Choosing an efficient query plan for a given query has already been studied as the problem of *query optimization* in database research. In relational databases, a query optimizer evaluates a set of query plans, choosing a query plan that potentially utilizes the minimum number of disk accesses [9]. For query processing in sensor networks, a query optimizer can be deployed at the AP to choose a query plan that utilizes minimal energy

¹To avoid confusion, we will use the term *node* for a sensor node and *sensor* for one of the possibly many sensing devices on a node (e.g., sensors for temperature and precipitation).

²As already pointed out, if the cost of deriving an efficient query plan is not significant, this task can be easily delegated to the individual nodes without any substantial changes in the technique that we develop.

(e.g. TinyDB). However, in the existing approaches a query plan defines *only how to extract data from an individual node*. Specifically, a query plan defines only the order that a nodes samples its sensors and checks if the resulting tuples satisfy the relevant predicate. In this paper, we broaden the scope of a query plan in the following way: *a query plan should not just define how to extract data from individual nodes but should define how to extract data from the entire set of participating nodes*. As shown in existing research [10, 11, 12, 13, 14] there exist multiple routing protocols that can be utilized to route the data generated by the nodes participating in the query to the AP. Choosing which routing protocol is most efficient for a given query is a crucial problem. Therefore, a query plan, according to our broader definition, includes both the order of sampling the sensors at the nodes *and* the routing protocol utilized to send the results back to the AP. Our goal then is to derive a query plan that minimizes the total energy associated with these operations.

Like TinyDB, we propose a query optimizer at the AP that iterates through a set of query plans, choosing the query plan that it estimates utilizes the least energy. Moreover, unlike TinyDB a query plan in our approach is an aggregation of all of the following procedures related to query processing:

- **Metadata Collection:** what metadata, if any, is needed and how to collect it.
- **Query Execution:** in what order should the nodes extract data from their sensors.
- **Result Collection:** how should the collected data be delivered from the nodes to the AP.
- **Building the Infrastructure:** how should the AP notify the nodes of the chosen query execution order and the routing infrastructure derived for result collection.

To determine how to extract data from the sensor network the query optimizer must have global metadata that defines the data profiles of the nodes participating in the query. However, in sensor networks metadata is maintained by each node rather than the query optimizer. As a result, energy must be consumed in order to send metadata from nodes to the query optimizer. If metadata is not collected, the query optimizer will not be able to accurately estimate the energy cost of query plans. As a result the query plan chosen may not be one of the best choices. For *heavy queries*, that consume significant energy performing query execution and result collection the overhead on metadata collection is a worthy investment. On the other hand, for *light queries*, that consume little energy performing these procedures,

metadata collection can be costly. The query optimizer we propose utilizes statistics to classify a given query as heavy or light. Based on this classification our query optimizer will decide whether to collect metadata.

This paper makes two significant contributions. First, the main contribution is the design of our query optimizer, which includes defining the cost model utilized to estimate the energy cost of a query plan. To the best of our knowledge, we are the first to propose a query plan that defines not only how to extract data from a node but also how to extract data from the sensor network. A salient feature of our approach is the estimation and comparison of the metadata collection cost for a query versus the remaining costs. This allows our system to collect metadata *only* for queries when it would help improve the overall energy cost. To the best of our knowledge, this is the first study that exploits the trade-off between the cost of metadata collection and the improved accuracy of estimating energy costs provided by metadata. Second, we conduct simulations to show that our approach utilizes significantly less energy than a local query processing engine which only evaluates query plans that extract data from a sensor node. In our simulation, our approach utilizes 46% less energy, on average.

The rest of this paper is organized as follows. In Section 2, we give an overview of existing query processing systems. We then introduce our query optimization framework in Sections 3 and 4. Next, we conduct simulations in Section 5 to determine the efficiency of our approach. Finally, in Section 6 we conclude our work and present areas for future research.

II. RELATED WORK

Two well known query processing engines for sensor networks are TinyDB and Cougar [1, 2, 3, 4]. In these systems, a user injects a query at the AP. Upon receiving the query, the AP collects metadata in an aggregated form from all nodes participating in the query. Based on the collected metadata, the AP generates a single query plan that defines the order a node samples its sensors. Next, the AP utilizes a MST to disseminate the chosen query plan to all nodes participating in the query. Nodes receiving the query execute the query plan and send the resulting data to the AP via the MST.

Chowdhary et. al [15] argue that a sensor network can logically be viewed as a relational database in which each node is a table. Utilizing this viewpoint, they proposed a set of algorithms that implement the SQL "join" operator in sensor networks. The goal of this work was to determine how to reduce the memory

and communication required to join the data generated by two distinct sets of nodes. In this paper, the sets of nodes that are joined is specified in the SQL query. The cartesian product followed by the set of predicates is utilized to perform the join. Our work is different because we consider joining multiple node(s), which, requires that a query plan defines the order that the nodes are joined.

Existing research has proposed several parameterized query optimization systems which postpone part of the query optimization process until runtime [16, 17, 18, 19, 20]. In sensor networks, each node is equipped with a query optimizer which chooses a set of query plans and the conditions utilized to select the appropriate query plan at runtime. By postponing parts of the query optimization process until runtime this approach is capable of gracefully re-optimizing to account for expected changes in the selectivity of the query plan's predicates. The problem with this approach, is that it does not adapt well to unexpected changes because the set of query plans chosen by the query optimizer does not contain all possible query plans. A more aggressive strategy proposed in Eddies [21, 22, 23] combines query optimization and query execution. In this approach, a query plan is defined at runtime on a tuple to tuple basis. Unlike traditional relational query optimizer, which utilize collected metadata, this approach utilizes fluid dynamics and a lottery based ticketing scheme in order to determine at runtime the selectivity of the queries predicates.

Dynamically re-adapting a query plan is beyond the scope of this paper. Also, simple extensions cannot be provided to existing parameterized query optimization systems to support the query plans defined in this paper. The reason is that global knowledge is required by the query optimizer to determine how extract data from the sensor network. Global knowledge is required because the efficiency of a routing protocol is dependent on the data characteristics of multiple nodes. For aggressive strategies such as Eddies the cost of collecting global information could outweigh the benefit achieved by dynamically re-adapting the query plan. In the future, we plan on extending our approach to dynamically re-adapt the query plan at runtime.

III. QUERY OPTIMIZER DESIGN

This section describes our query optimizer's architecture. In this paper, we focus on optimization of a single query (Q1) which is comprised of Q predicates $PS_{Q1} = \{P_1, P_2, \dots, P_Q\}$ and is participated by NN nodes, $SN_{Q1} = \{N_1, N_2, \dots, N_{NN}\}$.

In this paper we assume that each node is equipped with Global Positioning Satellite (GPS) [24] and the AP innately knows the Longitude and Latitude of each node in the sensor network. Next, for simplicity we assume that each node is static with respect to movement, nodes do not fail, and the AP is attached to a fixed power source. In addition, we assume that each node has a set of sensors, $Sen = \{S_1, S_2, \dots, S_n\}$, and for each sensor $S_i \in Sen$, a node maintains a histogram that represents the probability that S_i generates a given value. For instance, Figure 1 illustrates a histogram depicting node i 's Temperature sensor. In this paper we denote the bounds of values sensed by $S_i \in Sen$ as $S_i\{L\}$ and $S_i\{H\}$, where $S_i\{L\}$ and $S_i\{H\}$ are the lower and upper bound of values that sensor S_i can generate, respectively. For instance, referring to Figure 1, $S_i\{L\}$ is 0 and $S_i\{H\}$ is 10. In this paper we assume the AP innately knows $S_i\{L\}$ and $S_i\{H\}$, for each sensor $S_i \in Sen$.

Fig. 1. Example Histogram

A. Overview

In our approach, the AP generates a set of query plans, referred to as the Query Space (QS). Each query plan defines how to extract data from the sensor network in addition to a node. For clarity, we refer to the QS of Q_1 as $QS_{Q1} = \{QP_1, QP_2, \dots, QP_n\}$, where QP_i denotes a distinct query plan.

There are two stages in our query optimizer. The first, referred to as the *classification stage*, estimates the cost of each query plan to determine what metadata, if any, should be collected. In this stage, the AP initially evaluates each query plan within the QS. The statistics utilized to calculate the cost of a query plan are discussed in Section 5. Based on this evaluation, the AP chooses $QP_x \in QS_{Q1}$ that utilizes minimal energy. Next, if the metadata collection component of QP_x specifies that no metadata is collected, QP_x is distributed to each node $N_i \in SN_{Q1}$, which execute the query. On the other hand if the metadata collection component of QP_x dictates that metadata is collected the query optimizer performs the second stage of the algorithm, referred to as the *optimization stage*. The goal of this stage is to determine a good query plan. In this stage, the AP: 1) collects metadata, 2) utilizes the collected metadata to evaluate each query plan in QS_{Q1} , 3) chooses the

query plan $QP_y \in QS_{Q1}$ that utilizes the least energy 4) disseminates QP_y to each node $N_i \in SNQ_{Q1}$, which execute the query.

In a large network, the QS can consist of numerous query plans. As with traditional relational query optimizers, generating and evaluating each query plan in the QS is un-necessary [25, 26]. In this paper, we limit the QS to contain only query plans with the following characteristics: 1) *each node performs the queries predicates/projections and compresses the resulting data before performing communication*, 2) *each node routes its data to a single node*. Briefly, before transmitting data a node utilizes the Run Length compression algorithm [27], wherein consecutive identical data values are replaced by a simple code with the data value and length of the run. Our query optimizer evaluates all query plans that adhere to these conditions. To ensure that cycles do not exist in a query plan, our query optimizer tests each query plan utilizing the Topological ordering algorithm [28]. Query plans that contain a cycle are not evaluated by our query optimizer.

IV. COST ESTIMATION

In this section, we define the statistics our query optimizer utilizes to estimate the energy cost of a query plan.

A. Overview

In our approach, costs associated with query plans are estimated in two separate stages. As discussed, the first is referred to as the classification stage and the second is referred to as the optimization stage.

Classification Stage: In this stage we determine if the query under consideration is a light or a heavy query, as defined in Section III-A. This requires an estimation of the cost of the different query plans. However, estimating the cost of a query plan is a tricky problem because its accuracy requires the system to already have some knowledge of the metadata itself. We utilize a simple estimation technique in this work. In the absence of any knowledge of the metadata, we work with the latest information we have about the form of the metadata. An example of this could be the distribution of metadata obtained when it was gathered last. Another approach could be to assume a fixed distribution for the metadata - the closer this assumed distribution is to the real distribution, the more accurate our estimation of metadata collection cost would be. In this paper, we assume the metadata to be uniformly distributed over the feasible range of data values.

Optimization Stage: In this stage we utilize the collected metadata to determine a good query plan, as defined in Section III-A.

B. Assumptions

To estimate the cost of a query plan, the AP must have some knowledge of the selectivity of each predicate $P_i \in PS_{Q1}$. Therefore, in our approach the AP initially assumes for each node N_i a *default histogram* that represents the probability that N_i 's sensor S_i generates a given value. In our approach, since the AP has no knowledge of the histogram node $N_i \in SNQ_{Q1}$ maintains for sensor $S_i \in Sen$ it assumes that sensor S_i , generates any value with equal probability. In other words, the AP assumes each node's histograms are uniformly distributed.

In this section we make the following assumptions:

- Node 0 is the AP.
- The AP's metadata represents at any given time the selectivity of the set of buckets $APM = \{B_1, B_2, \dots, B_n\}$
- The selectivity of bucket B_i is denoted as $Sel\{B_i\}$.
- The size (in bits) of QP_i is denoted as $S\{QP_i\}$.
- The *duration* and *epoch* of $Q1$ is defined as D_{Q1} and E_{Q1} , respectively.
- Query $Q1$ accepts tuples in the range $T_{Q1,L}$ to $T_{Q1,H}$.
- The rate metadata is collected is denoted as MCR.
- The communication radius of a nodes radio is denoted as α .
- A sensor requires θ Joules to sample.
- The bandwidth of a node's radio is denoted as δ .
- The number of Joules to send/receive a bit is defined as β and γ , respectively.
- The number of hops between two nodes N_i and N_y , is denoted as $H\{N_i, N_y\}$.
- The communication environment is contention- and error-free; hence, sensors do not have to retransmit any data.

Since we assume the AP innately knows the $\langle \text{longitude}, \text{latitude} \rangle$ of each node as well as the geographical radius of a nodes radio, Dijkstras shortest path algorithm [29, 30] can be utilized to estimate the number of hops that exist between two nodes.

C. Classification Stage

In this section we discuss how to estimate the energy associated with a given query plan $QP_i \in QS_{Q1}$ in the classification stage.

1) *Metadata Collection:* In this section we estimate the energy associated with metadata collection. In our algorithm the AP sends a message to each node $N_i \in SNQ_{Q1}$ instructing it to send the selectivity of each

predicate $P_i \in PS_{Q1}$ to the AP. The total number of hops this message traverses is denoted as:

$$HCM\{QP_i\} = \sum_{y=1}^z H\{N_0, CMN_{QP_i}[y]\}$$

Second, each node $N_y \in CMN_{QP_i}$ sends its metadata to the AP. Again, $HCM\{QP_i\}$ denotes the number of hops to collect metadata. Assume that the average size (in bits) of the metadata $M\{N_y\}$ collected from each node $N_y \in CMN_{QP_i}$ is defined as $S(M\{N_y\})$. Under this assumption the total energy required to collect metadata is denoted as:

$$C\{MC, QP_i\} = \left(\frac{S\{QP_i\} + S(M\{N_y\})}{\delta} * \gamma \right) * MCR * HCM\{QP_i\}^2$$

2) *Query Execution*: In this section we define the statistics utilized to estimate the energy associated with query execution. In our approach, the AP first temporarily updates its metadata to represent the metadata collected by the query plan. However, as discussed the classification stage is to determine if metadata should be collected, thus in this stage the AP has yet to collect metadata for the query plan. In our approach, to simulate metadata collection the AP utilizes its existing metadata to estimate the metadata collected by the query plan. Assume that if QP_i is chosen node $N_y \in CMN_{QP_i}$ sends to the AP metadata that represents the selectivity of the set of buckets $M\{N_y\} = \{B_1, B_2, \dots, B_m\}$. To determine the selectivity of each bucket $B_i \in M\{N_y\}$ the AP iterates through its current metadata to determine two buckets: 1) the bucket, $B_L \in APM$ in which $\min(B_i - B_L)$ and $B_L \leq B_i$, 2) the bucket $B_H \in APM$ in which $\min(B_H - B_i)$ and $B_H \geq B_i$. Using this notation the AP utilizes the following equation to estimate the selectivity of bucket B_i :

$$Sel\{B_i\} = \begin{cases} \text{if } B_L = P_i & Sel\{B_L\} \\ \text{if } B_H = P_i & Sel\{B_H\} \\ \text{if } B_H \neq P_i \cap B_L \neq P_i & Sel\{B_L\} + \frac{P_i}{B_H} * \\ & (Sel\{B_H\} - Sel\{B_L\}) * \left(100\% - \frac{B_L}{P_i}\right) \end{cases}$$

Next, the AP temporarily updates its metadata to represent the selectivity $Sel\{B_i\}$ of each bucket $B_i \in M\{N_y\}$. Note that after calculating the cost of a query plan the AP erases this temporary metadata. Utilizing this temporary metadata the AP then determines the selectivity of each predicate $P_i \in PS_{Q1}$ at each node $N_i \in SNQ_{Q1}$. In our approach, we assume that if metadata is not accurate the AP will not choose a good

query plan. Therefore, if the AP's metadata does not accurately represent the selectivity of predicate P_i we pessimistically assume that P_i has a lower selectivity. To determine the selectivity of predicate P_i the AP iterates through its current metadata to determine two buckets: 1) the bucket, $B_L \in APM$ in which $\min(B_i - B_L)$ and $B_L \leq B_i$, 2) the bucket $B_H \in APM$ in which $\min(B_H - B_i)$ and $B_H \geq B_i$. Using this notation the following equation estimates the selectivity of predicate P_i :

$$WSEL\{P_i, N_y\} = \begin{cases} \text{if } B_L = P_i & Sel\{B_L\} \\ \text{if } B_H = P_i & Sel\{B_H\} \\ \text{if } B_H \neq P_i \cap B_L \neq P_i & Sel\{B_L\} + \\ & (Sel\{B_H\} - Sel\{B_L\}) * \\ & \left(\frac{P_i + \frac{B_H - P_i}{2}}{B_H} \right) * \left(100\% - \frac{B_L}{P_i}\right) \end{cases}$$

Assuming that at node N_i , QP_i states that predicate P_i should be executed before predicate P_{i+1} the energy associated with query execution at node N_i is defined as follows:

$$WQE\{N_i\} = \left[\left(\sum_{y=1}^Q WSEL\{P_y, N_i\} * \prod_{d=1}^{y-1} WSEL\{P_d, N_i\} \right) * \theta \right]$$

Thus the energy associated with query execution is defined as follows:

$$C(QE, QP_i) = (D_{Q1}/E_{Q1}) * \sum_{y=1}^{NN} WQE\{SNQ_y\}$$

3) *Result Collection*: In this section we define the statistics utilized to estimate the energy associated with result collection.

First, we calculate the number of data items each node $N_i \in SNQ_{Q1}$ generates for Q_{Q1} . We assume that N_i generates at maximum 1 data item every *epoch*. Under this assumption the following equation denotes the number of data items generated by node N_i :

$$MDI\{N_i\} = \prod_{y=1}^Q WSEL\{P_y, N_i\}$$

In our approach QP_i denotes the set of R nodes $RN_{Q1} = \{N_1, N_2, \dots, N_R\}$ that participate in result collection. Note that $SNQ_{Q1} \subset RN_{Q1}$. Since each node in set $SM_{Q1} = \{RN_{Q1} - SNQ_{Q1}\}$ does not participate in query execution the $MDI\{N_x\}$ for a given node $N_x \in SM_{Q1}$ is 0.

Second, we calculate the number of data items each node receives from its children. Assume that $N_i \in RN_{Q1}$ has C children $C_{N_i} = \{N_1, N_2, \dots, N_C\}$ which

it receives $ND\{N_1\}, ND\{N_2\}, \dots, ND\{N_C\}$ data items from, respectively. Under this assumption the following equation denotes the number of data items N_i receives from its children:

$$TDI\{N_i\} = \sum_{y=1}^c ND\{N_y\}$$

Hence the following equation defines the number of data items at node N_i during a given epoch E :

$$NDI\{N_i\} = (MDI\{N_i\} + TDI\{N_i\})$$

Third, node $N_i \in RN_{Q1}$ compresses its $NDI\{N_i\}$ data items into a new set of data items of size $CNDI\{N_i\}$. Again utilizing the AP's metadata, assume that the probability that node N_i has a given data item DI_o at epoch E is $PB\{N_i, DI_o, E\}$. Utilizing the harmonic mean the average probability that a node $N_y \in C_{N_i}$ or N_i has a given data item DI_o is denoted as:

$$PDI\{DI_o, E\} = \frac{C+1}{\frac{1}{PB\{N_i, DI_o, E\}} + \sum_{w=1}^C \frac{1}{PB\{C_{N_i}[w], DI_o, E\}}}$$

Next, based on the binomial distribution it can be derived that the probability that K nodes have data item DI_o at epoch E is defined as:

$$PBNN\{DI_o, K, E\} = \frac{C+1}{(K!)((C+1-K)!)} * (PB\{DI_o, E\})^K * (1 - PB\{DI_o, E\})^{C+1-K}$$

Thus, the following equation represents the number of nodes that have data item DI_o .

$$CN\{DI_o\} = \sum_{j=2}^{c+1} PBNN\{DI_o, j, E\} * j$$

Based on this the number of duplicate data items is denoted as:

$$DDN\{N_i\} = \sum_{j=1}^{T_H - T_L} CN\{DI_j\}$$

The total number of bits needed to represent the length of all runs (utilized by the Run Length encoding algorithm) is denoted as:

$$RLS\{N_i\} = \sum_{j=1}^{T_H - T_L} \text{if } CN\{DI\} > 1 \quad 32bits$$

Thus the total number of unique data items at N_i is denoted as: $NDU\{N_i\} = NDI\{N_i\} - (DDN\{N_i\})$. The total size (in bits) of the data N_i transmits to its parent is defined as: $CNDI\{N_i\} = NDU\{N_i\} * 32 + RLS\{N_i\}$. Next, assuming that K nodes $N_1, N_2, \dots, N_k$ in RN_{Q1} are leaf nodes in the routing infrastructure, the energy required to send all data packets to the AP is defined as:

$$C(SDP) = \sum_{y=1}^R \frac{CNDI\{N_y\}}{\delta} * \beta$$

The energy required to receive all data packets is defined as:

$$C(RDP) = \sum_{y=k+1}^R \frac{CNDI\{N_y\}}{\delta} * \gamma$$

Thus the energy required to perform result collection is denoted as:

$$C(RC, QP_i) = \frac{D_{Q1}}{E_{Q1}} * (C(SDP) + C(RDP))$$

4) *Building the Infrastructure*: In this section we define the statistics we utilize to estimate the energy associated with building the infrastructure. In our approach the AP sends the infrastructure to each node $N_i \in RN_{Q1}$. Thus the total number of hops that the infrastructure message traverses is defined as:

$$HCI = \sum_{y=1}^R H\{N_0, RN[y]\}$$

The total energy utilized to disseminate the infrastructure is defined as follows:

$$C(BI, QP_i) = HCI * (C(SDP) + C(RDP))$$

5) *Total Energy*: The total energy of a QP_i is defined as:

$$C(QP_i) = C(MC, QP_i) + C(QE, QP_i) + C(RC, QP_i) + C(BI, QP_i)$$

Utilizing this equation the query optimizer calculates the energy associated with each query plan $QP_y \in QS_{Q1}$ choosing the query plan $QP_c \in QS_{Q1}$ that utilizes the least energy. The query optimizer then collects the metadata specified by QP_c . If not metadata is collected by the chosen query plan QP_c , QP_c is distributed to each node $N_i \in RN_{Q1}$.

D. Optimization Stage

In this section we discuss how to estimate the energy associated with a given query plan $QP_i \in QS_{Q1}$ in the optimization stage. Since in the optimization stage, metadata has been collected, we only calculate the energy associated with: 1) query execution, 2) result collection, and 3) building the infrastructure.

The statistics utilized to estimate the cost of a query plan in the optimization stage are equivalent to the statistics defined in the classification stage. Note that our cost estimates are now based on the collected metadata. Hence, $C(QE, QP_i)$ defines the energy associated with query execution, $C(RC, QP_i)$ defines the energy associated with result collection and $C(BI, QP_i)$ defines the energy associated with building the infrastructure. Utilizing, these equations the total cost of a query plan QP_i is denoted as:

$$OC(QP_i) = C(QE, QP_i) + C(RC, QP_i) + C(BI, QP_i)$$

Utilizing this equation the query optimizer calculates the energy associated with each query plan $QP_y \in QS_{Q1}$ choosing the query plan $QP_c \in QS_{Q1}$ that utilizes the least energy. Query plan QP_c is then distributed to each node $N_i \in RN_{Q1}$.

V. PERFORMANCE EVALUATION

To evaluate our approach, experiments utilizing NS-2 [31] network simulator were conducted. First, experimental results show that our query optimizer accurately classifies a query as heavy or light. Second, our experiments show that, for a heavy query, our query optimizer accurately estimates the cost of a query plan. Finally, experimental results show that compared to our simulation of a local query processing engine, our approach utilizes significantly less energy.

A. Experimental Settings

In all experiments, the sensor network consisted of 50 static nodes distributed in a fixed area (600m * 600m). A node's position was dictated by 1) uniformly chosen, 2) the coordinates (scaled accordingly) of a subset of nodes given by The Joint Institute for the Study of the Atmosphere and Ocean (JISAO) [32]. In our experiments the local metadata of a node was: 1) generated uniformly at random such that a bucket in each histogram is between 0–100, 2) obtained from JISAO. In terms of metadata, the buckets in each histogram represents a probability between 0–100%, and all buckets in a histogram add up to 100%. Unless specified, in our

Fig. 2. Factors that Affect the Classification Algorithm

experiments each node utilizes 1500 uJ to sample a sensor [1] and metadata is collected once per week. This choice of parameters covers a wide range of settings that are realistic to nodes [33].

B. Classification of a Query

This section conducts experiments to determine if our query optimizer correctly classifies a query. In terms of classification, the following factors impact whether a query is heavy or light: 1) the number of times the query is executed (execution rate), 2) cost to execute the query (execution cost), 3) the rate metadata is collected (metadata rate). For each of these factors, we conducted an experiment to determine its impact on our query optimizer. In this experiment the metadata utilized was generated uniformly at random.

In each experiment we compared the estimates of the energy consumed by the best query plan that dictates metadata is collected (QPMC) vs. the best query plan that dictates metadata is not collected (QPNMC). For each query plan we illustrate: 1) the actual energy consumed (actual energy), 2) the energy our query optimizer estimated the query plan to consume (estimated energy). As discussed, if the estimated energy of QPNMC is greater than the estimated energy of QPMC, our optimizer classifies the query as heavy, otherwise it classifies the query as light. To determine if our query optimizer correctly classified a query, we compare the actual energy of QPMC and QPNMC. If our query optimizer correctly classified a query as heavy, the actual energy of QPMC will be less than the actual energy of QPNMC or vice versa. The query considered in these experiments is defined as follows:

[Query 1]

Select Temperature

From Sensors

Where Precipitation < 150 and Temperature < 25 and Pressure < 107.5 and Longitude < 450 and Longitude > 200 and Latitude < 600 and Latitude > 300

epoch E minutes *duration* 1 Month

First, we analyzed energy consumption as the execution rate of the query is increased from 39 times per month ($E = 21$ hours) to 10080 times per month ($E = 4$ minutes). As illustrated in Figure 2(a), if the rate is small (i.e. ≤ 157), our query optimizer correctly classified the query as light. If the rate is large (i.e. ≥ 2520), our query optimizer correctly classified the query as heavy. Also as illustrated, if the rate is between 315 and 1260 times per month, our query optimizer may incorrectly classify the query. However, our query optimizer makes the wrong decision due to the relative closeness in energy consumption between the two query plans. In other words, this error is only prevalent if the difference (in energy) between QPMC and QPNMC is relatively small, thus it does not significantly impact the

performance of our query optimizer. Second, to analyze the affect of the execution cost, we fixed the queries *epoch* such that $E = 64$ and increased the energy required to sample a sensor from 187.5 uJ to 6000 uJ. As illustrated in Figure 2(b) if little energy is required to sample a sensor (i.e. ≤ 187.5 uJ), our query optimizer correctly classified the query as light and if significant energy is required to sample a sensor (i.e. ≥ 1500 uJ) our query optimizer correctly classified the query as heavy. Also as illustrated, if the sampling cost is between 375 uJ and 750 uJ our query optimizer may incorrectly classify the query. However, again this is due to the relative closeness in energy between the two query plans, thus it does not significantly impact the performance of our query optimizer. Third, we fixed the queries *epoch* such that $E = 64$, and analyzed energy consumption as the metadata rate is increased from 1 time per month to 64 times per month. As illustrated in Figure 2(c), if metadata is collected at a high rate (i.e. ≥ 32), our query optimizer correctly classified the query as light and if metadata is collected at a low rate (i.e. ≤ 4) our query optimizer correctly classifies the query as heavy. Also as illustrated, if the metadata collection rate is between 8 and 16 times a month our query optimizer may incorrectly classify the query. However, again this is due to the relative closeness in energy between the two query plans, thus it does not significantly impact the performance of our query optimizer.

C. Accuracy of Query Plan

In this section, we conducted experiments to determine if the query plan chosen by our query optimizer is efficient in terms of energy. To achieve this, we modified our simulator to execute each query plan in the QS. For each query plan we illustrated the: 1) estimated energy, 2) actual energy. In this experiment we evaluated Query 1 where $E = 21$ hours (light query) and $E = 4$ minutes (heavy query). The metadata considered in this experiment was generated uniformly at random. First, as illustrated by Figure 3(a), for the light query our estimates are not accurate. In light queries metadata collection is costly, thus it is not collected by our query optimizer. As discussed, without metadata our cost estimates are inaccurate, but this inaccuracy is acceptable since the energy saved from not collecting metadata supersedes choosing a good query plan. For instance, in Figure 3(a) query plans 1–36 and 36–72 do and do not collect metadata, respectively. As this Figure illustrates, for a light query, not collecting metadata is a good strategy because the energy associated with query execution and result collection are minimal relative to the cost associated with metadata collection. Second, as

(a) Light Query

(b) Heavy Query

Fig. 3. Energy Consumed by Each Query Plan

illustrated in Figure 3(b), our cost estimates are accurate for the heavy query, thus we chose a good query plan. These estimates are accurate because they are based on the metadata collected from the nodes participating in the query. As illustrated, for a heavy query, choosing a good query plan is important because the difference (in energy) between the optimal query plan and the most inefficient query plan is 43.4%.

Fig. 4. Performance For Various Queries

D. Comparison

The goal of this last experiment is to determine if our approach is more efficient than our simulation of a local query processing engine. In this experiment we simulated 25 randomly generated queries, comparing for each query our approach and our simulation of a local query processing engine. In each query the following factors were chosen randomly:

- Topology: 5 topologies each consisting of 50 nodes.

- Metadata: generated uniformly at random or defined by JISAO.
- Predicates: 1 to 5 randomly chosen predicates.
- *Duration*: 1 day to 3 months.
- *Epoch*: 4 minutes to 3 months.
- Metadata Collection Rate: 1 to 64 times a month.

As illustrated in Figure 4, our approach consumes less energy for 100% of the queries consuming on average 46% less energy than our simulation of a local query processing engine. Our approach is more efficient because a query plan defines how to extract data from the sensor network in addition to the nodes.

VI. CONCLUSION

There exist multiple routing protocols that can be utilized to route the data generated by the nodes participating in the query to the AP [10, 11, 12, 13, 14]. Choosing which routing protocol is most efficient for a given query is a crucial problem. In this paper we propose a query optimizer at the AP that iterates through a set of query plans, choosing the query plan that it estimates utilizes the least energy. Unlike state-of-the-art query processing engines a query plan in our approach defines how to extract data from a sensor node *and* the routing protocol utilized to send the results back to the AP. The main contribution of this paper is the design of our cost model which is utilized to estimate the energy cost of a query plan. A salient feature in our approach is the estimation and comparison of the metadata collection cost for a query versus the remaining costs. This allows our system to collect metadata only for queries when it would help improve the overall energy cost. Through simulation we showed that our approach chooses a good query plan and our approach utilizes significantly less

energy than our simulation of a local query processing engine.

This research assumed only a single query is posed to the AP. In the future we plan on releasing this constraint. Also in this research we assumed metadata was static and nodes do not fail. This ensured that the chosen query plan is efficient for the lifetime of the query. However, since nodes can fail and data generated by sensors is streaming we plan in the future to extend our approach to dynamically re-adapt query plans.

VII. REFERENCES

- [1] S. R. Madden, M. J. Franklin, J. M. Hellerstein, , and W. Hong, "The design of an acquisitional query processor for sensor networks," in *2003 ACM SIGMOD international conference on on Management of data*, June 2003, pp. 491 – 502.
- [2] Y. Yao and J. Gehrke, "Query processing for sensor networks," in *Proceedings of the First Biennial Conference on Innovative Data Systems Research*, Asilomar, CA, 2003, pp. 21–32.
- [3] S. Madden, R. Szewczyk, M. Franklin, and D. Culler, "Supporting aggregate queries over ad-hoc wireless sensor networks," in *4th IEEE Workshop on Mobile Computing Systems and Applications*, Callicoon, New York, June 2002.
- [4] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "TAG: a tiny aggregation service for ad-hoc sensor networks," *ACM SIGOPS Operating Systems Review*, vol. 36, no. SI, pp. 131–146, 2002.
- [5] R. Rosemark and W.-C. Lee, "Decentralizing query processing in sensor networks," in *Mobiquitous*, 2005, pp. 270–280.
- [6] X. Li, Y. J. Kim, R. Govindan, and W. Hong, "Multi-dimensional range queries in sensor networks," in *Proceedings of the first international conference on Embedded networked sensor systems*, Los Angeles, California, 2003, pp. 63–75.
- [7] D. Goldin, M. Song, A. Kutlu, H. Gao, and H. Dave, "Georouting and delta-gathering: Efficient data propagation techniques for geosensor networks," 2003.
- [8] Y. Xu and W.-C. Lee, "Window query processing in highly dynamic sensor networks: Issues and solutions," in *In Proceedings of NSF Workshop GeoSensor Networks (GSN03)*, Portland, Maine, Oct. 2003.
- [9] M. M. Astrahan, M. W. Blasgen, D. D. Chamberlin, K. P. Eswaran, J. Gray, P. P. Griffiths, W. F. K. III, R. A. Lorie, P. R. McJones, J. W. Mehl, G. R. Putzolu, I. L. Traiger, B. W. Wade, and V. Watson, "System r: Relational approach to database management," *ACM Trans. Database Syst.*, vol. 1, no. 2.
- [10] B. Krishnamachari, D. Estrin, and S. Wicker, "Modelling data-centric routing in wireless sensor networks," in *In IEEE INFOCOM*.
- [11] D. Braginsky and D. Estrin, "Rumor routing algorithm for sensor networks," in *Proceedings of the First ACM International Workshop on Wireless Sensor Networks and Applications*, Atlanta, Georgia, September 2002, pp. 22–31.
- [12] I. Stojmenovic, "Voronoi diagram and convex hull based geocasting and routing in wireless networks," 1999. [Online]. Available: citeseer.nj.nec.com/stojmenovic99voronoi.html
- [13] V. D. Park and M. S. Corson, "A highly adaptive distributed routing algorithm for mobile wireless networks," in *INFOCOM* (3), 1997, pp. 1405–1413.
- [14] C. Intanagonwiwat, R. Govindan, and D. Estrin, "Directed diffusion: a scalable and robust communication paradigm for sensor networks," in *Proceedings of the annual international conference on Mobile computing and networking*, Boston, Massachusetts, 2000, pp. 56–67.
- [15] V. Chowdhary and H. Gupta, "Communication-efficient implementation of join in sensor networks." in *DASFAA*, 2005, pp. 447–460.
- [16] A. Deshpande, C. Guestrin, W. Hong, and S. Madden, "Exploiting correlated attributes in acquisitional query processing," in *ICDE '05: Proceedings of the 21st International Conference on Data Engineering (ICDE'05)*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 143–154.
- [17] Y. E. Ioannidis, R. T. Ng, K. Shim, and T. K. Sellis, "Parametric query optimization," *VLDB Journal: Very Large Data Bases*, vol. 6, no. 2, pp. 132–151, 1997. [Online]. Available: citeseer.ist.psu.edu/article/ioannidis92parametric.html
- [18] S. Ganguly, "Design and analysis of parametric query optimization algorithms," in *VLDB'98, Proceedings of 24rd International Conference on Very Large Data Bases, August 24-27, 1998, New York City, New York, USA*, A. Gupta, O. Shmueli, and J. Widom, Eds. Morgan Kaufmann, 1998, pp. 228–238.
- [19] R. L. Cole and G. Graefe, "Optimization of dynamic query evaluation plans," 1994, pp. 150–160. [Online]. Available: citeseer.ist.psu.edu/cole94optimization.html
- [20] G. Graefe and K. Ward, "Dynamic query evaluation plans," in *SIGMOD '89: Proceedings of the 1989 ACM SIGMOD international conference on Management of data*. New York, NY, USA: ACM Press, 1989, pp. 358–366.
- [21] R. Avnur and J. M. Hellerstein, "Eddies: continuously adaptive query processing," 2000, pp. 261–272.
- [22] S. Madden, M. Shah, J. M. Hellerstein, and V. Raman, "Continuously adaptive continuous queries over streams," in *ACM SIGMOD Intl. Conf. on Management of Data*, Madison, WI, 2002, pp. 49–60.
- [23] A. Deshpande and J. Hellerstein, "Lifting the burden of history from adaptive query processing," 2004. [Online]. Available: citeseer.ist.psu.edu/article/deshpande04lifting.html
- [24] N. Patwari, A. O. H. III, M. Perkins, N. Correal, and R. O'Dea, "Relative location estimation in wireless sensor networks," *IEEE Trans. on Sig. Proc. Special Issue on Signal Processing in Networking*, vol. 51, no. 9, pp. 2137–2148, 2002.
- [25] S. T. Van, "Reducing the search space for conceptual." [Online]. Available: citeseer.ist.psu.edu/704638.html
- [26] Y. E. Ioannidis, "Query optimization," *ACM Computing Surveys*, vol. 28, no. 1, pp. 121–123, 1996. [Online]. Available: citeseer.ist.psu.edu/ioannidis96query.html
- [27] "Run length encoding." [Online]. Available: <http://en.wikipedia.org/wiki/Runlength.encoding>
- [28] "Topological ordering." [Online]. Available: <http://www.ics.uci.edu/~eppstein/161/960208.html>
- [29] D. B. Johnson, "A note on dijkstra's shortest path algorithm," *J. ACM*, vol. 20, no. 3, pp. 385–388, 1973.
- [30] F. Schulz, D. Wagner, and K. Weihe, "Dijkstra's algorithm online: an empirical case study from public railroad transport."
- [31] S. Bajaj, L. Breslau, D. Estrin, K. Fall, S. Floyd, P. Halder, M. Handley, A. Helmy, J. Heidemann, P. Huang, S. Kumar, S. McCanne, R. Rejai, P. Sharma, K. Varadhan, Y. Xu, H. Yu, and D. Zappala, "Improving simulation for network research," University of Southern California, Tech. Rep. 99-702b, 2003.
- [32] "Joint institute for the study of the atmosphere and ocean," http://tao.atmos.washington.edu/data_sets/.
- [33] "Crossbow technologies." [Online]. Available: <http://www.xbow.com/>