

Similarity-Aware Query Processing in Sensor Networks

Ping Xia Panos K. Chrysanthis Alexandros Labrinidis
Department of Computer Science, University of Pittsburgh, Pittsburgh, PA 15260
{pxia, panos, labrinid}@cs.pitt.edu

Abstract

We assume a sensor network with data-centric storage, where sensor data is stored within the sensor network and ad hoc queries are disseminated to and processed in the network. In such an environment, there are often similarities among queries submitted to the sensor network, e.g., overlapping in query ranges for range queries. Using current solutions, similar queries may have to go through the same expensive query processing steps, i.e., recomputed from scratch every time. As a result, energy, the most scarce resource in sensor networks, is wasted. In this paper, we propose a similarity-aware query processing scheme (SAQP) that materializes previous query results within the sensor network and utilizes these materialized query results to answer future similar queries. Through simulation, we demonstrate that our SAQP scheme reduces energy consumption on queries with negligible increase in response time, and without compromising the quality of data.

1 Introduction

In typical sensor network applications, sensors are deployed to monitor changes in the environment. For example, in disaster management, sensors provide data for the detection of events such as fire, gas leak, earthquake, *etc.*). When a disaster happens, first responders are sent out to the disaster area to evaluate the situation. As such, they are mobile agents who communicate with nearby sensor nodes using their hand-held devices to get valuable sensor readings for their assessment and planning. During the response, one agent may issue a query “*list all hot regions that have temperature above 200F and light levels between 20 and 30*” and another agent might issue a similar query “*list all hot regions that have temperature above 250F and light level between 25 and 35*”.

In this scenario, the queries are typically ad hoc queries that could be issued through any sensor node within the communication range of the mobile agents. One way to handle such queries is to send all sensor data and queries to a base station. However, in most cases the base station, if one exists, is located at a place that is far away from regions where sensors are deployed. Given that energy is the most valuable and scarce resource in sensor networks, the base-

station approach is extremely energy-inefficient for two reasons. First of all, typically ad hoc queries only request small portions of sensor data generated over a short period, thus transferring sensor data that would never be requested to a faraway base station is a waste of energy. Secondly, all queries need to be forwarded to the base station and query results need to be sent back from the base station, which would be energy consuming because of the long distance between the base station and sensor nodes. Therefore, given that sensor nodes are closer to each other than to the base station, even if one exists, the most suitable schemes to answer such queries are data centric storage (DCS) schemes [1, 2, 6, 8, 10] under which sensor data (in the form of events) are mapped to specific sensor nodes, where data are stored and retrieved from.

However, not all data are created equal; skewed access patterns are part of nature. In a disaster management application, it is expected that users will request more frequently certain data items, making them more “hot” (because, for example, they are from the building that is on fire or because their values are outside the normal operating range). This uneven interest from users results in similarities among the queries submitted by users. Current DCS schemes cannot exploit similarities that exist among queries (and would have enabled them to save energy during query processing). Unaware of the similarities, the underlying network needs to process similar queries repeatedly, which involves routing the query to all relevant sensor nodes where events are stored, and returning the query results back from those nodes to the query issuer.

This observation motivates us to design a *similarity-aware query processing (SAQP)* scheme, which can answer queries in an energy-efficient way by exploiting the similarities among different queries issued to DCS sensor networks. The basic idea of our proposal is to replicate results (events) for previously issued queries as materialized views in the network and utilize the materialized views to answer similar queries. Our proposed scheme is a general query processing scheme that works for any type of queries, *e.g.*, range, aggregation and/or group-by. Under SAQP, to process a query, a query need not be forwarded to nodes where the events are originally stored. Instead, other nodes with

replicated events will be selected if they are more suitable to answer the query.

In summary, our contribution in this paper is four-fold:

1. We propose materialized views in sensor networks which dynamically replicate query results and enable the sharing of query processing among similar queries.
2. We propose a query processing algorithm that utilizes the materialized views when processing similar queries.
3. We formulate the candidate selection problem, *i.e.*, to find an optimal subset of candidate sensor nodes to answer a query, as an optimization problem. We then design a greedy algorithm to solve the problem.
4. We conduct an extensive simulation study to evaluate the performance of our proposal, comparing it to state-of-the-art query processing schemes for sensor networks. Our simulation results demonstrate that SAQP reduces energy consumption on queries with negligible increase in response time, and without compromising the quality of data.

We explore related work in the next section.

2 Related Work

Many data dissemination and query processing architectures in sensor networks have been proposed, *e.g.*, [2, 4, 6, 7, 8, 9, 10, 11]. These proposals can be grouped into two categories based on the type of query supported, *i.e.*, continuous or ad-hoc queries, and on the location where the data are stored, *i.e.*, within or outside the sensor network.

The first category assumes that there is a base station in the sensor network and sensor data are sent back to the base station for further processing. Under this category, the storage capability of sensor nodes have been completely ignored. Since communication is more energy-consuming compared to local computation, blindly sending all data back to the base station is not a good approach in general. In-network aggregation [3, 4, 7, 9, 11] could reduce energy consumption by aggregating data en route. However, it works typically for continuous aggregation queries such as SUM, MIN and MAX while it is not quite suitable for ad hoc queries. Besides, a setup phase is required such that queries are flooded from the base station to the network and then a routing tree rooted at the base station is built. If the duration of the query is not long enough, the setup phase will dominate the cost.

The other category is data centric storage (DCS) [1, 2, 6, 8, 10]. Instead of sending data back to and storing them at a base station, sensor data are stored at some *consolidator* sensor nodes, in the form of high-level events. Ad-hoc

queries are then submitted to the network and are responded to by sensor nodes where events satisfying the query are stored. Among these approaches, GHT [8] uses a hash function to hash each type of events to a geographic location. For each event type, the sensor node that is closest to the corresponding hashed location becomes the consolidator sensor node of that event type. GHT employs a geographical routing protocol, GPSR [5], to route events and queries to the consolidator sensor node. DIFS [2] uses a geographically bounded hash function to determine the locations where events are stored and builds a distributed hierarchical index based on quad trees to direct range queries. DIM [6] uses k-d trees to determine the consolidator sensor node. ZS [1] addresses storage load balance issues in DIM.

These DCS schemes are oblivious to the similarities among queries; processing each individual query will go through all the necessary steps repeatedly, which might be costly and redundant. Our SAQP scheme differs from current DCS schemes by considering the similarities among queries. SAQP generates materialized views, *i.e.*, replicated query results, for each query it has processed and utilizes these materialized views to process new queries that have similarities with previous queries. To the best of our knowledge, no previous work has been done to address the problem of using materialized results in sensor networks

3 Similarity Aware Query Processing

Traditionally, a database server may generate materialized views, which are materialized versions of previous query results. Such views are used to answer future queries that are similar to past queries. The purpose of materialization is to reduce query processing cost in terms of CPU time. Our goal of caching query results is also to reduce query processing cost, however, in terms of energy consumption. Because of this similarity, we call the cached query results materialized views (or *M-views*, for short).

In this section, we present the SAQP scheme in detail. For simplicity, we use range queries to demonstrate our idea. We first describe the system model. After explaining the general query processing scheme under SAQP, we show how queries are split in the situation when two queries overlap only partially. After formally defining the candidate selection problem, we propose a greedy algorithm to solve the problem. At the end of this section we discuss quality of data (QoD).

3.1 System Model

In DCS schemes, sensor data is stored in the form of high-level events, which are abstractions of raw sensor readings. Each event type consists of a set of attributes, corresponding to phenomena (*e.g.*, temperature and humidity) detected by

sensors, timing parameters (e.g., duration of events, etc.) or spatial dimensions (e.g., location of events, etc.) [2]. Events are more semantically attractive than raw sensor readings, since they reveal that interesting situations are developing. To give an example, both object detection and hot region could be defined as events.

We follow the same data abstraction used in DCS schemes. In our system, events are stored locally at the node where they are detected (we call this node original storage node, or *O-node*, for short) and indexes to these events are stored at an index node (or *I-node*, for short). As opposed to GHT, we are using a hash function to determine the index node rather than the consolidator node that stores all events. The index of an event is based on the attributes of the event that would be used in range queries, thus it is expected to be small compared to the size of the event. The I-node also keeps a directory of M-views. Each M-view directory entry is associated with a query in the past, which includes the Q-node ID from which the query is submitted and the range of that query. Hereafter, we refer to the nodes associated with M-views as M-view nodes, or *M-nodes*, for short.

We now describe the structure of the event, index, and M-view directory entries using the hot region event example.

An *event* (Table 1) consists of three parts: scalar attributes, timestamp, and other event details. For the hot region event, we assume that temperature is the only scalar attribute. Event details may include information such as heat gradient, the size of the hot region, coordinates of the boundary points of the hot region, etc. The timestamp indicates when the event was detected. New events do not replace all events and old events are dropped based on an aging scheme or when they are no longer needed.

An *index entry* (Table 2) includes an O-nodeId, scalar attributes and a timestamp, where O-nodeId indicates the O-node at which the event is stored, scalar attributes are interesting attributes of the event that will be used in queries, and the timestamp indicates when the event was generated.

An *M-view directory entry* (Table 3) also contains three parts: an M-nodeId, a range and a timestamp, where the M-nodeId specifies the M-node at which the M-view is stored, the range indicates ranges (based on the scalar attributes) of the events in the M-view and the timestamp indicates when the entry was created.

O-nodeId	temperature	timestamp	other event details
----------	-------------	-----------	---------------------

Table 1: The structure of a hot region event

O-nodeId	temperature	timestamp
----------	-------------	-----------

Table 2: The structure of an index

M-nodeId	range (e.g., 100-200)	timestamp
----------	-----------------------	-----------

Table 3: The structure of an M-view directory entry

By associating a timestamp to M-View entries, it is not necessary to invalidate an M-View entry if a new event would fall into the range covered by that M-View entry; without the timestamp, this is needed since the new event is not materialized. An M-view created at time t for range $[a, b]$ only contains complete set of events within range $[a, b]$ up to time t . This time dimension is natively treated as another range query predicate which allows for maximum utilization of the M-Views scheme.

Although in this paper we focus on a single index node for simplicity, in order to make the system fault-tolerant, the I-node could be replaced by a set of nodes (a cluster). For example, such a cluster can be formed using the Perimeter Refresh Protocol (PRP) (proposed in GHT [8]). Only one node (cluster head) in the cluster is responsible to process queries. However, using the same protocol, a refresh packet is circulated among the cluster, which ensures a consistent view among all nodes in the cluster and in the case that the cluster head fails, another node could be elected to be the new cluster head. Further, using a set of I-nodes instead of one node would also be helpful to reduce the storage requirement imposed on a single node. Under such a setup, each I-node would be responsible only for data in a subset of the whole data range thus the load would be balanced by these I-nodes.

3.2 Query Processing Scheme

In SAQP, the general procedure of query processing is as follows. When an I-node receives a range query from the Q-node, it examines each entry in its M-view directory to see whether its range overlaps with the range of the new query. If such an entry is found, the M-node associated with the entry is considered as a candidate to answer the query. If there are many such entries found, the M-nodes corresponding to these entries are all considered as *candidates*. In addition, the I-node examines the indexes to O-nodes. Those O-nodes storing at least one event that satisfies the query are considered as candidates as well. The I-node then forms a responder set by selecting a set of nodes from the candidate set based on some criteria (see Sec 3.4, 3.5). Once the responder set is determined, the I-node generates an M-view directory entry for this query and forwards the query to all responders. The responders in turn send the qualified events directly to the Q-node. The Q-node then materializes the query results as an M-view.

3.3 Split a Query

Ideally, when the I-node receives a query, we expect to find an M-view entry such that its range is a superset of the query range and thus the corresponding M-node alone can answer the query completely. However, in most cases the range of

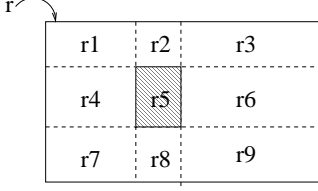


Figure 1: Splitting a 2-dimensional range r with range $r5$

an M-view entry might only partially overlap with the query range or only be a subset of the query range. In these situations, the system needs to find more than one M-nodes to answer the query. If two M-nodes are selected as responders, it is possible that the M-view range associated with the two M-nodes overlaps with each other, thus the I-node should send a modified query to the two M-nodes to avoid duplicates (in other words, split the original query). For example, suppose we have a query on range $[10, 20]$ and two M-view entries with ranges $[5, 14]$ and $[12, 21]$. If the two M-nodes are selected as responders, because range $[5, 14]$ overlaps with range $[12, 21]$, both of them will send events within range $[12, 14]$ to the Q-node, which results in duplicates and wastes energy, if it is not dealt with.

For simplicity, let's consider an one dimensional range query first. Given the query range $[a, b]$ and the range of an M-view entry $[x, y]$, there are three cases:

- $[a, b]$ contains $[x, y]$,
- $[x, y]$ contains $[a, b]$, or
- $[a, b]$ intersects with $[x, y]$.

In the first case, the range $[a, b]$ could be split as $[a, x]$, $[x, y]$ and $[b, y]$. The query is split into two queries: Q1 on range $[x, y]$ and Q2 on range $[a, x] \cup [b, y]$. In the second case, there is no need to split the query. Finally, in the last case, either $x < a \leq y \leq b$ or $a \leq x \leq b < y$. Q2 will be simplified to one term. After the query split, Q1 will be sent to the M-node associated with the M-view entry and Q2 is used to evaluate other candidates further.

In the case of multi-dimensional range queries, the splitting strategy is the same. The query range is split into disjunctions of conjunctions of ranges. However, as the dimensionality increases, the number of disjunction terms may increase dramatically. For example, Figure 1 shows the result of splitting a two dimensional query range using the above method where region r represents the query range and region $r5$ represents the range of the M-view entry. After splitting, Q1 contains $r5$ and Q2 consists of eight small ranges. Even if we combine the regions on the side ($r2, r4, r6$ and $r8$) with the regions at the corner ($r1, r3, r7, r9$), there are still at least four subregions generated after one split. This clearly does not scale to the number of splits.

One observation is that after a query split, the information that needs to be forwarded to other responders is the

query range not covered by the range used for split. Thus, we can represent a split region by two ranges, the range used for the split and the query range. For instance, to represent the above split region, we only need to specify two ranges r and $r5$, *i.e.*, it could be represented with $r-r5$. As a result, the size of the query forwarding message would only increase linearly with the number of splits.

3.4 Candidate Selection

The candidates to answer a query consist of M-nodes whose M-view range overlaps with the query range and O-nodes with indexes that fall within the query range. The candidate selection problem is to select among these candidates, a set of sensor nodes that can answer the query completely, with minimum energy cost. These selected sensor nodes form the *responder set*, which satisfies the following two requirements. First, the query can be answered completely by nodes in the responder set. Second, it should consume less energy for processing the query, *i.e.*, forwarding and answering the query, compared with the basic approach of forwarding query to and getting results from only O-nodes in the candidate set.

Let the set of candidates be $\{N1, N2, \dots, Nn\}$. Each candidate is associated with a cost, which is the sum of the energy cost of forwarding the query to the node and the energy cost of returning the results back to the Q-node. The associated cost could be represented as $\{C1, C2, \dots, Cn\}$. Each candidate is also associated with a disjunction of ranges, represented as r . For example, for candidate $N1$, $r1 = r11 \cup r12 \cup \dots \cup r1n$, where $r11, \dots, r1n$ correspond to the ranges of M-views or original events stored at $N1$. Note that for the range of an original event, the lower bound equals to the upper bound. The associated disjunction of ranges is thus represented as $\{r1, r2, \dots, rn\}$.

The candidate selection problem becomes an optimization problem in which we want to minimize

$$Total_cost = x1 * C1 + x2 * C2 + \dots + xn * Cn \quad (1)$$

where the coefficients, *i.e.*, $x1, x2, \dots, xn$, represent the percentage of the range in a M-View that will be used to answer a query and are subject to following condition:

$$x1 * r1 \cup x2 * r2 \cup \dots \cup xn * rn \geq r \quad (2)$$

3.5 Candidate Selection Algorithm

We propose a greedy algorithm to solve the candidate selection problem (Algorithm 1). The algorithm works as follows. First we find all events that satisfy the query based on the indexes of events (initialization of set Events in Algorithm 1). An O-node will be considered as a candidate

Algorithm 1 Greedy Candidate Selection Algorithm

Inputs:

Responder = { }
Events = {x | x is an event that satisfies the query. }
CandidateOnodes = {x | $\exists y \in Events \wedge x$ is the O-node that stores y}
CandidateMnodes = {x | x is an M-node and there are at least one M-views stored at x overlaps with the range of the query. }

Local Variables: cost, costWithoutMview, costWithMview, nid, CandidateOnodes', Events'

Procedure:

```
1: cost = 0
2: while CandidateMnodes  $\neq \emptyset$  do
3:   costWithoutMview = cost of forwarding query to all nodes in CandidateOnodes and returning events in set Events from those nodes to Q-node
4:   nid = the closest node (to Q-node) in CandidateMnodes
5:   CandidateMnodes = CandidateMnodes - {nid}
6:   Events' = {x | x  $\in$  Events  $\wedge$  x is materialized at node nid }
7:   Events = Events - Events'
8:   CandidateOnodes' = {x |  $\exists y \in Events' \wedge x$  is the O-node that stores y}
9:   CandidateOnodes = CandidateOnodes - CandidateOnodes'
10:  Split the query to query1 and query2 according to nid's M-view
11:  costWithMview = cost of forwarding query1 to node nid + cost of forwarding query2 to all nodes in CandidateOnodes + cost of returning events in set Events' from node nid to Q-node + cost of returning events in set Events from nodes in CandidateOnodes to Q-node
12:  if costWithMview < costWithoutMview then
13:    Responder = Responder  $\cup$  {nid}
14:    cost = cost + cost of forwarding query1 to node nid + cost of returning events in set Events' from node nid to Q-node
15:    replace query with query2 for further evaluation
16:  else
17:    Event = Event  $\cup$  Event'
18:    CandidateOnodes = CandidateOnodes  $\cup$  CandidateOnodes'
19:    continue use the query for further evaluation
20:  end if
21: end while
22: Responder = Responder  $\cup$  CandidateOnodes
23: cost = cost + cost of forwarding query to all nodes in CandidateOnodes and returning events in set Events from those nodes to Q-node
```

if at least one event in set Events is reported by this O-node. All such O-nodes form the first candidate set (initialization of set CandidateOnodes). Also we find all M-views that overlap with the query range and order the corresponding M-nodes based on their distance to the Q-node, which forms the second candidate set (initialization of set CandidateMnodes). M-nodes closer to the Q-node have higher priority. We then remove the M-node with the highest priority from set CandidateMnodes (Lines 4-5). By doing this, we intend to select this M-node to be a responder. Some of the O-nodes can then be removed from set CandidateOnodes since the M-views stored at the selected M-node might contain events stored on them (Lines 6-8). We then compare the cost of using this M-node as a responder to the cost of not using it. If using this M-node as a responder reduces the cost, the M-node will be selected as a responder (Lines 12-14), otherwise we take back the changes to set CandidateOnodes (Lines 16-17). The procedure continues until the

CandidateMnodes set is empty. At the end, if set CandidateOnodes is not empty, all O-nodes in it are added into the responder set (Line 21).

The algorithm might not always be able to provide an optimal solution, however, the decision made at each step is suboptimal which leads to a reasonable selection of candidates. Our experimental results verified that the algorithm indeed saves energy (Sec 4).

3.6 Quality of Data (QoD)

The proposed SAQP scheme is to design to save energy (and performs as expected). What is interesting to point out is that it achieves this without compromising the Quality of Data (QoD) returned to users (compared to GHT). In fact, in some instances, SAQP offers better QoD than GHT. To illustrate this, consider a scenario that one node detected two consecutive events and one query is issued after the two events are detected. Ideally both events should be returned to the query issuer (assuming both events satisfy the query). One situation is that the first store-index (or store-event, under GHT) message, the query and the second store-index (or store-event, under GHT) message reach the I-node (or consolidator node, under GHT) in the proper order. Under GHT, the query issuer will get only the first event since the second event is not stored yet. However, under SAQP the query issuer would be able to get both events if this O-node is selected as a responder (because no M-view is available to cover the first event detected by this O-node). Thus, SAQP could achieve a better QoD. In all other cases, SAQP provides the same QoD as GHT.

4 Experiments

4.1 Simulation Setup

We compare our scheme with the original GHT approach and an Index-based GHT approach (IGHT). Under the GHT approach the events are sent to the consolidator node and upon query request the events are returned to the query issuer from the consolidator node directly. Under the IGHT approach, events are stored locally and indexes to the events are sent to the consolidator node. Whenever a query is sent to the consolidator node, a set of nodes that contain events which satisfy the query is first determined based on the indexes. The query is then forwarded to those nodes to be answered. The reason to compare our scheme with IGHT is to show that simply introducing an index node does not work well. In fact, in most cases, the IGHT scheme seems to consume more energy. For this reason, we focus on the comparison between SAQP and GHT when we discuss the experimental results.

Symbol	Defination	Setting
N	Num of Sensor Nodes	400
X * Y	Size of the Sensing Region	400m * 400m
E_{trans}	Transmitter Electronics	50nj/bit
E_{rec}	Receiver Electronics	50nj/bit
E_{amp}	Transmit Amplifier	0.1nj/bit/m ²
E	number of events	100
Q	number of queries	100
ESize	event size	8bytes
RSize	range size	4bytes
ISize	index size	4bytes
D	domain of the scalar attribute	0-100
Z_s	skewness of zipf distribution	0.5
C	Confining factor	0.5
Δt	time interval	50

Table 4: Default simulation parameters

We use two performance metrics to compare the three schemes. One is the total energy consumption and the other one is the average query response time. The total energy consumption includes the energy consumed in storing the events and also the energy consumed in processing queries. In the paper, the response time is defined as the time elapsed from issuing a query to the network until receiving the first event in the query results. We estimate the response time as the number of routing hops.

We use a well-known energy model [3], where the energy cost of sending and receiving a message with size k bits between two sensor nodes with distance d is given by:

$$E_{transmit} = E_{trans} * k + E_{amp} * d^2 \quad (3)$$

$$E_{receive} = E_{rec} * k \quad (4)$$

where E_{trans} and E_{rec} is the energy consumed in enabling the transmitter and receiver electronics respectively, while E_{amp} is the the energy consumed in the amplifier.

We designed our workload as follows. At each time tick, there is either one event or one query generated from a randomly selected sensor node. Each event is associated with the node from which it is generated and the timestamp at which it is generated, and also includes a scalar attribute. The event size is a varied parameter. The value of the scalar attribute follows a uniform distribution within its data domain.

In order to control the spatial locality of queries, we generate queries from a confined region centered at the center of the sensing region. The confined region is controlled by a confining factor C , denoting that all queries are generated from a region centered at the center of the sensing region with size $(C * X) \times (C * Y)$ where $X \times Y$ defines the size of the whole sensing region. A smaller C means a higher query spatial locality.

The range of a range query has the following form: $(v_{low} : v_{high}, t - \Delta t : t)$. It represents a two dimensional

range query with the first dimension to be the scalar attribute and the second dimension to be the time. Clearly, SAQP not only has the ability to support multi-dimensional range queries, but also treats timestamps natively as one of the query dimensions. The (sub)range on the first dimension of the generated queries has fixed length and the center of the range follows a zipf distribution with various skewness parameter Z_s . A smaller Z_s means that the distribution is less skewed, i.e., when $Z_s = 0$, the query center becomes uniformly distributed. The (sub)range on the second dimension represents the user interest in all events generated in a time frame $[t - \Delta t : t]$. In the simulation we have set t to be the time when the query is initiated. However, t can be set to any time in the past if it is explicitly specified in a user query. Δt could be set to any interval to reflect different user interest as well. In the simulation we have conducted experiments using different Δt values to observe this effect.

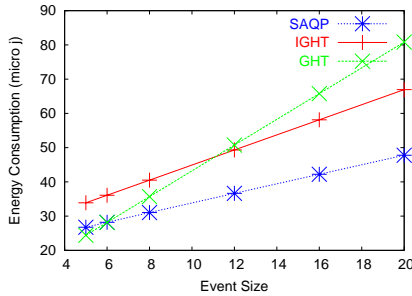
Table 4 shows the meaning and default values of the system variables we have used in the simulation. Note that I_{Size} is at least 4 bytes if we assume 1 byte for node id, 1 byte for scalar attribute and 2 bytes for timestamp. E_{Size} is at least 5 bytes since it contains the event details and is greater than I_{Size} . The R_{Size} is set to be 4 bytes since queries in the simulation have 2 dimensions and each dimension has a low and high value. During query splitting, the size of a query message is determined by multiplying R_{Size} with the number of ranges included in the query.

4.2 Experimental Results

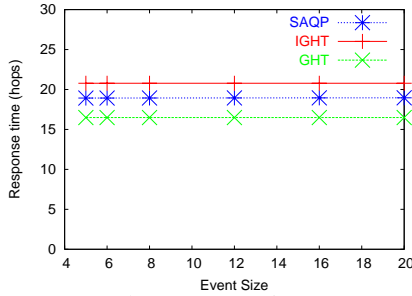
We have run a set of experiments to evaluate the proposed SAQP scheme. In each experiment, we vary one parameter and run the experiment 100 times using different random seeds. The results are plotted using the average of 100 runs.

As a norm of the results of these experiments, the total energy consumption under SAQP is always less than the one using IGHT. The reason for this is straightforward. The energy consumption under IGHT is the lower bound on the energy consumption under SAQP since if no M-nodes are selected as responders, SAQP is exactly the same as IGHT. Under SAQP, an M-node will be selected as a responder only if doing so saves energy compared to IGHT.

The SAQP scheme also consumes less energy compared to GHT. There are two major reasons. First, the cost of storing events under GHT is higher than the cost of storing indexes under SAQP. Second, in GHT, when the distance from the consolidator node to the Q-node is large, sending results from the consolidator node back to the Q-node is energy-consuming. However under SAQP, query results may be returned from a nearby M-node of the Q-node, thus reducing the energy consumption dramatically. SAQP has slightly higher response time than GHT, since GHT returns results from the consolidator node to query node directly,

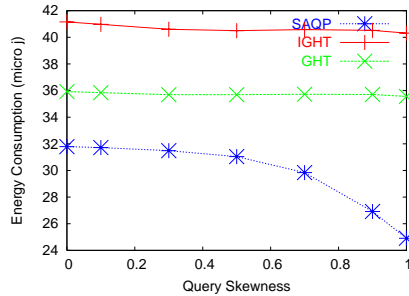


(a) Energy cost

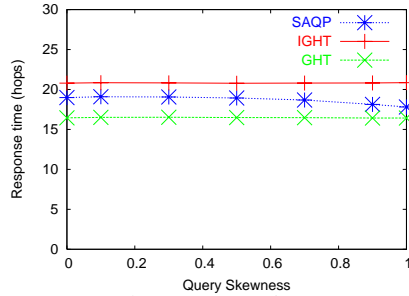


(b) Response time

Figure 2: Effect of event size

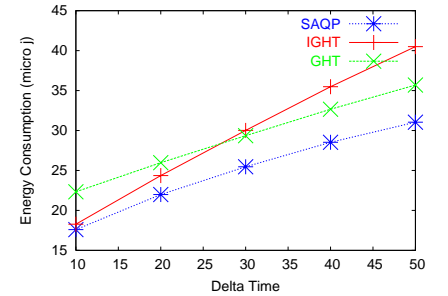


(a) Energy cost

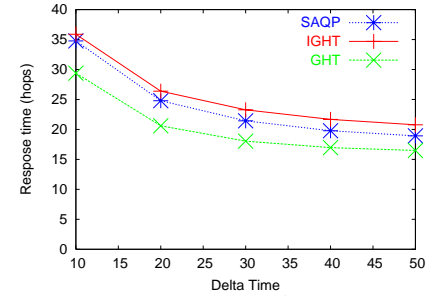


(b) Response time

Figure 3: Effect of query skewness



(a) Energy cost



(b) Response time

Figure 4: Effect of Δt

whereas SAQP needs to forward the query to responders. However, as we can see from the experiments, the difference is only 2 or 3 hops where the response time for GHT is at least 15 hops. This is because it is often the case that a nearby M-node of a Q-node is selected to answer part of the query. For the same reason, SAQP has better response time than IGHT in all experiments.

Event Size Figure 2 shows the total energy consumption and the response time when the size of events varies from 5 to 20 bytes. The total energy consumption increases linearly in all three cases. However, SAQP has a much smaller slope, which is expected because serving query results from a nearby M-node will reduce the cost dramatically. Note that the index size is 4 bytes, and if the event size is too close to it (e.g., 5 bytes), the SAQP scheme consumes slightly more energy than GHT does. However, this is acceptable, since normally an index would be useful only if its size is sufficiently smaller than the original data size.

Query Skewness Figure 3 shows the total energy consumption and the response time when Z_s changes from 0 to 1. GHT and IGHT are not affected by this parameter. However, under the SAQP scheme, if queries are skewed (which represents a situation that user queries have a lot of similarity), more queries can utilize the previous query results from M-views. Therefore, energy consumption for SAQP decreases when Z_s increases.

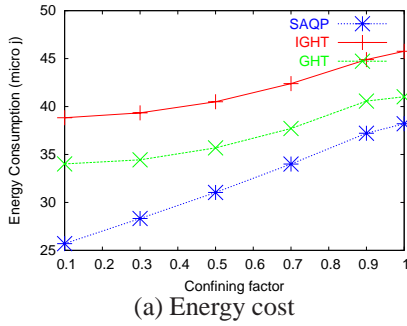
Time Interval Figure 4 shows the total energy consumption and response time when the time interval Δt varies. When

Δt is small, energy consumption under SAQP is close to that under IGHT because there will be fewer queries generated in a short time interval and thus fewer M-nodes available to respond. However, when Δt becomes larger, more queries will be generated, and thus more M-nodes could be selected to responder to a future query, resulting in the decrease of consumed energy. Under all cases SAQP consumes less energy than GHT.

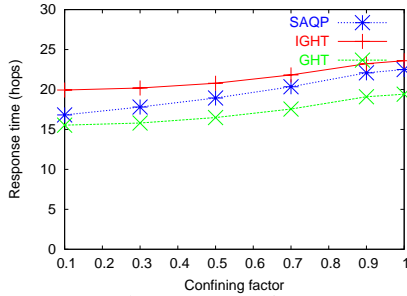
Query Locality Figure 5 shows the results when varying the confining factor C . $C=1$ is the case without any query locality. A small confining factor results in higher query locality, which is taken advantage by SAQP to reduce energy consumption. The energy consumed (SAQP vs. IGHT) is 25.7 mJ vs. 34.2 mJ when C equals to 0.1 (25% improvement) and 38.2 mJ vs. 41.0 mJ when C equals to 1 (6.8% improvement), which means SAQP will save even more energy when C is small. The reason for this is that there is a higher chance to find nearby M-nodes to respond if queries are more “crowded” within a confined region.

Number of Queries In Figure 6 the number of query varies from 100 to 1000. The increase of energy consumption under SAQP is much slower than the other two schemes. This is expected as more queries result in more M-Views created and higher chances to use an M-node to answer a query.

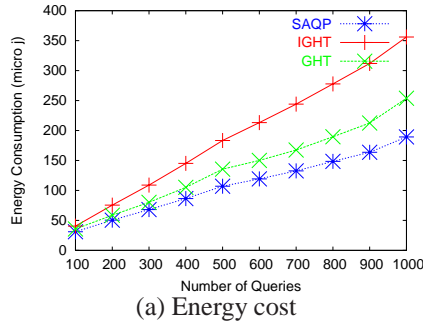
Number of Events In the last experiment we vary the number of events from 100 to 1000 (Figure 7). The energy consumption of GHT increases dramatically when the number of events increases. This is because all events are sent to the



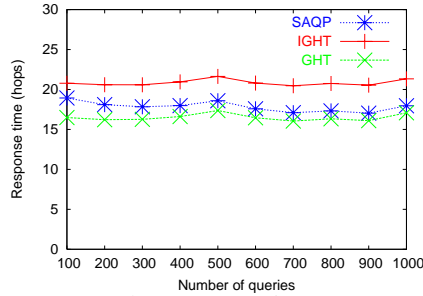
(a) Energy cost



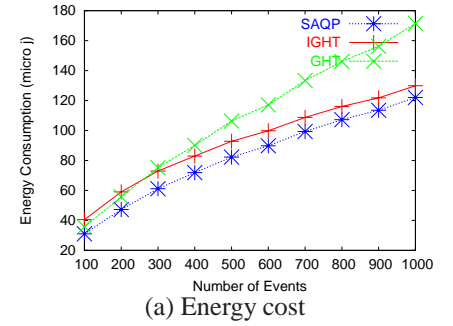
(b) Response time



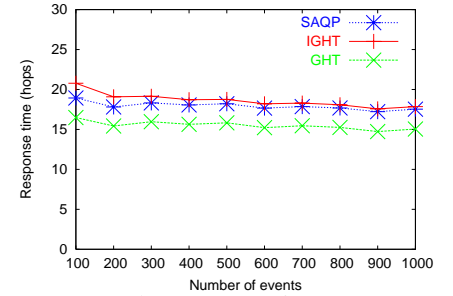
(a) Energy cost



(b) Response time



(a) Energy cost



(b) Response time

Figure 5: Effect of query locality

Figure 6: Effect of number of queries

Figure 7: Effect of number of events

consolidator node to be stored. Our SAQP scheme consistently consumes less energy compared to IGHT.

5 Conclusions

Skewed access patterns are everywhere; as such we expect to frequently have similarities among queries submitted to sensor networks. In this paper, we proposed a similarity-aware query processing scheme that creates materialized views in sensor networks and utilizes the materialized views to answer future queries that are similar to past ones. We illustrate that using other schemes which ignore query similarities wastes energy by repeating unnecessary processing and communication. By using our query split strategy and candidate selection algorithm, our proposed query processing scheme reduces energy consumption, with a slight increase in response time, without compromising QoD.

References

- [1] M. Aly, N. Morsillo, P. K. Chrysanthis, and K. Pruhs. Zone sharing: A hot-spots decomposition scheme for data-centric storage in sensor networks. In *DMSN*, 2005.
- [2] B. Greenstein, D. Estrin, R. Govindan, S. Ratnasamy, and S. Shgenker. DIFS: A distributed index for features in sensor networks. *Elsevier Journal of Ad Hoc Networks*, 2003.
- [3] W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. In *HICSS*, 2000.
- [4] C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heidemann, and F. Silva. Directed diffusion for wireless sensor networking. *IEEE/ACM Trans. Netw.*, 11(1), 2003.
- [5] B. Karp and H. T. Kung. GPSR: Greedy perimeters stateless routing for wireless networks. In *MOBICOM*, 2000.
- [6] X. Li, Y. J. Kim, R. Govindan, and W. Hong. Multi-dimensional range queries in sensor networks. In *SenSys*, 2003.
- [7] S. Madden, M. J. Franklin, J. Hellerstein, and W. Hong. TAG: A Tiny AGgregation service for ad-hoc sensor networks. In *OSDI*, 2002.
- [8] S. Ratnasamy, B. Karp, L. Yin, F. Yu, D. Estrin, R. Govindan, and S. Shenker. GHT: A geographic hash table for data-centric storage in sensornets. In *WSNA*, 2002.
- [9] M. A. Sharaf, J. Beaver, A. Labrinidis, and P. K. Chrysanthis. TiNA: A scheme for temporal coherency-aware in-network aggregation. In *MobiDE*, 2003.
- [10] S. Shenker, S. Ratnasamy, B. Karp, R. Govindan, and D. Estrin. Data-centric storage in sensornets. In *the First ACM SIGCOMM Workshop on Hot Topics in Networks*, 2002.
- [11] Y. Yao and J. Gehrke. The COUGAR approach to in-network query processing in sensor networks. *SIGMOD Record*, 2002.