Distrib Parallel Databases (2013) 31:321–351
DOI 10.1007/s10619-012-7093-3

Adaptive data acquisition strategies for energy-efficient, smartphone-based, continuous processing of sensor streams

Lipyeow Lim · Archan Misra · Tianli Mo

Published online: 24 May 2012

Abstract There is a growing interest in applications that utilize continuous sensing of individual activity or context, via sensors embedded or associated with personal mobile devices (e.g., smartphones). Reducing the energy overheads of sensor data acquisition and processing is essential to ensure the successful continuous operation of such applications, especially on battery-limited mobile devices. To achieve this goal, this paper presents a framework, called ACQUA, for 'acquisition-cost' aware continuous query processing. ACQUA replaces the current paradigm, where the data is typically streamed (pushed) from the sensors to the one or more smartphones, with a pull-based asynchronous model, where a smartphone retrieves appropriate blocks of relevant sensor data from individual sensors, as an integral part of the query evaluation process. We describe algorithms that dynamically optimize the sequence (for complex stream queries with conjunctive and disjunctive predicates) in which such sensor data streams are retrieved by the query evaluation component, based on a combination of (a) the communication cost & selectivity properties of individual sensor streams, and (b) the occurrence of the stream predicates in multiple concurrently executing queries. We also show how a transformation of a group of stream queries into a disjunctive normal form provides us with significantly greater degrees of free-

This work is supported in part by the Singapore Ministry of Education Academic Research Fund Tier 2 under the research grant MOE2011-T2-1-001. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the granting agency or Singapore Management University.

L. Lim (✉) · T. Mo
Information and Computer Sciences Department, University of Hawai'i at Mānoa, Honolulu, USA
e-mail: lipyeow@hawaii.edu

T. Mo
e-mail: tianli@hawaii.edu

A. Misra
School of Information Systems, Singapore Management University, Singapore, Singapore
e-mail: archanm@smu.edu.sg

dom in choosing this sequence, in which individual sensor streams are retrieved and evaluated. While the algorithms can apply to a broad category of sensor-based applications, we specifically demonstrate their application to a scenario where multiple stream processing queries execute on a single smartphone, with the sensors transferring their data over an appropriate PAN technology, such as Bluetooth or IEEE 802.11. Extensive simulation experiments indicate that ACQUA's intelligent batch-oriented data acquisition process can result in as much as 80 % reduction in the energy overhead of continuous query processing, without any loss in the fidelity of the processing logic.

Keywords Mobile data management · Streams · Complex event processing · Energy efficiency · Activity recognition · Mobile Sensing

1 Introduction

There has been a recent explosion in the field of "mobile sensing", i.e., the use of personal mobile devices (more specifically, smartphones equipped with an increasingly sophisticated set of sensors, such as GPS, accelerometer & microphone) for near-real time sensing of an individual's activity or environmental context. A major function of the smartphone in this paradigm is to perform embedded query processing on the sensor data streams to extract appropriate *individual context* in near-real time, for use in a variety of applications, ranging from automatic activity updates for social networking applications (e.g., [2]) to dynamic threshold adaptation for adaptive remote health monitoring (e.g., [3]).

Most current sensor-driven context-aware applications are 'episodic'—they are activated by the user intermittently, and for short durations. It has been well documented that the continuous processing of even moderate-data rate streams (such as GPS or accelerometer) can cause commercial smartphone batteries to be depleted in as low as 4–5 hours (e.g., see [1]). *Continuous* execution of such mobile sensing applications thus requires advances in both the acquisition and processing of such sensor data, so as to reduce the energy overheads on the battery-constrained mobile devices. There are at least two distinct scenarios where the act of *acquiring* the sensor data streams, from the sensor sources to the query processing engine on the smartphone, constitutes a dominant component of this energy overhead, and involves the use of a Personal Area Network (PAN) or wireless LAN technology:

(A) In various health and wellness applications, the context sensing logic on an individual's smartphone utilizes a combination of both phone-embedded sensors (e.g., GPS, accelerometer, compass and microphone) and *external* wearable medical (e.g., ECG, EMG, Sp02) or environmental (e.g., temperature, pollution) sensors. These external sensors transmit data to the phone using a PAN technology (such as Bluetooth™ or IEEE 802.15.4) or even WiFi (IEEE 802.11).
(B) In recently-proposed social or cooperative-sensing applications (e.g., [13, 14]), the query processing engine utilizes data from sensors embedded in multiple proximal smartphones, which help to both lower the energy burden on an individual device and to provide better "surrounding" context. In these cases, the

data from the remote sensors is transferred between devices, using short-range wireless technologies that are either infrastructure-based (such as WiFi or femto-cellular) or device-to-device (such as FlashLinQ [15]).

Our work in this paper thus explores an approach to reduce the energy footprint of such continuous sensor data acquisition, primarily by **reducing the volume of sensor data that is transmitted wirelessly between a smartphone and its sensor sources**, **without** compromising the fidelity of the event processing logic.

In current approaches, the data acquisition process is based on a "push" model, where the sensors simply continuously transmit their samples to the smartphone (where the query processing engine is executing), and are effectively decoupled from the actual query execution. Our proposed new approach instead utilizes a phone-controlled "dynamic pull" model, where the smartphone *selectively* pulls only appropriate subsets of the sensor data streams. More importantly, the process of acquisition of the sensor data is now *coupled to the dynamic query execution state* of one or more concurrently executing queries. This new model exploits the intelligent programming and data filtering capabilities becoming commonplace on many emerging wearable sensor platforms (e.g., the SHIMMER platform [4]), whose data storage and transmission algorithms can be programmed 'over the air' in near real-time.

In this paper, we introduce a new Acquisition Cost-Aware QUery Adaptation (or **ACQUA**) framework, where the query processing engine on the smartphone dynamically modifies both the *order* and the *segments of data streams* that it will request from each individual sensor. The ACQUA framework first learns the selectivity properties of different sensor streams and then utilizes such estimated selectivity values to modify the sequence in which the smartphone acquires data from the sensors. This modification can potentially occur at each evaluation instant of the long-running continuous queries, and is based on the joint consideration of both the communication overheads of acquiring the sensor data and the selectivity properties.

The principal focus of this paper is on developing and evaluating the *algorithmic logic* for intelligently modifying the acquisition sequence for a set of executing complex stream processing queries, given a-priori knowledge of the communication cost and selectivity properties. We first describe the ACQUA algorithms for a set of complex long-running queries, consisting of a combination of disjunctive and conjunctive predicates over 'tumbling window'-based stream operators. We then show how the conversion of such queries into a Disjunctive Normal Form (DNF) representation enables ACQUA to best explore the different possible sequences for sequential sensor data acquisition. We then specifically consider the case '[A]' above (namely, a single smartphone retrieving data from wearable sensors via a PAN interface) to illustrate the specific characteristics of the communication overheads, and subsequently use an extensive set of simulation-based studies to quantify the energy savings achieved by ACQUA in this case.

Key contributions of this paper We believe the following to be our key contributions:

(a) We introduce, and develop an initial set of algorithms for, an 'acquisition cost' aware event processing paradigm, where the event engine on the smartphone dynamically optimizes the order in which it retrieves only relevant blocks of data streams from individual sensors. This order is adjusted dynamically both at the beginning of each evaluation instant, and also based on the evolving state of partially-executed query evaluations. We specifically describe algorithms to optimize the energy overheads of such data processing, taking into account both the wireless transfer cost and the query selectivity properties.

(b) We subsequently consider the impact that query representation has on the execution logic of ACQUA, and show how the DNF representation for our set of permitted queries provides the most flexibility in the ACQUA sequential stream selection logic.

(c) We then focus on the case of a single phone retrieving data from multiple sensors over a couple of representative PAN links (Bluetooth and WiFi). We carefully account for and exploit the significant transmission energy savings that result from the intermittent, "bursty", scheduled use of the PAN link between individual sensors and the phone to transfer the sensor data (as opposed to the continuous transmission of generated sensor data streams), as this allows the wireless radios to operate on a low duty cycle. We show how the use of such 'batched' data acquisition can be handled by the ACQUA algorithms.

The rest of the paper is organized as follows. Section 2 provides a brief survey of the related and prior work and establishes the documented trade-off between transmission energy efficiency and data 'batch' size for two representative PAN radio technologies. Section 3 captures the key objectives and issues that the ACQUA framework must consider and describes the component-level functional architecture of ACQUA. Section 4 provides a formal enumeration of the event query model and the operator set that we consider. Section 5 details the ACQUA sequence-computation algorithms and event processing logic. Section 6 then describes some of the key features to be considered while establishing the per-sample communication cost for retrieving the sensor streams, and Sect. 7 then presents simulation-based studies to evaluate the expected performance benefits, based on this detailed model. Finally, Sect. 8 concludes the paper with a discussion of open issues that we are working to address.

2 Related and prior work

The use of complex event processing of sensor data streams on a smartphone for detecting context on a smartphone has been previously explored in system prototypes such as Harmoni [3] (which used such context to dynamically change the definition of anomalous medical states) and CenceMe [2] (which applied rich operators on audio and accelerometer sensor streams to identify pre-defined human activities). To further reduce the energy overheads, the MediAlly prototype [9] used such inferred context to dynamically activate the collection of data from other external sensors. In contrast, our ACQUA framework seeks to optimize the data transfer during the process of context determination itself. There are at least two recent approaches that seek to optimize the query evaluation logic on a pervasive device, to take into account dynamic variations in the availability and operational parameters of nearby sensors. The Orchestrator framework [7] focuses on resource-sharing among multiple context-aware applications running independent stream queries—it selects an appropriate subset (from the currently available set) of sensors, so as to maximize the operational lifetime of the continuously executing applications. The ErdOS framework [13], on the other hand, views smartphone-based sensors as a shared resource and seeks to intelligently distribute the consumption of such resources—by effectively dynamically altering the *tasking* sequence for the sensors, it seeks to lower the energy burden of each individual device. For example, given a cluster of N GPS-equiped phones, ErdOS will task the individual GPS sensors in a round-robin fashion, effectively offering an N-fold reduction in the GPS sampling burden of an individual phone. In contrast to all these approaches, ACQUA seeks to optimize the actual *transfer* of data between the query processing engine & the sensors, rather than the sampling activity of each sensor.

Researchers have recently also investigated both hardware and software approaches to improve the energy-efficiency of continuous sensor sampling and query processing on smartphones. The *LittleRock* [5] prototype has demonstrated how the use of a special low-energy coprocessor can result in a two order-of-magnitude decrease in the *computational energy* spent in embedded processing of on-board sensor data streams. Our ACQUA framework can be viewed as complementary to such hardware or system-level innovations, as we seek to additionally reduce the *communication energy overheads* involved in acquiring the data wirelessly from additional external sensors. To address the challenges of energy-efficient *continuous* event processing on smartphones, the *Jigsaw* continuous sensing engine [6] has recently developed a pipelined stream processing architecture that adaptively triggers different sensors at different sampling rates to meet the context accuracy required by different applications. More recently, the SociableSense framework [14] has investigated the combination of cloud-based computation and adaptive sensor sampling to reduce the computational and sensing overheads during continuous mobile sensing.Our ACQUA framework can be considered complementary to these approaches, in that it focuses on optimizing the retrieval of stream segments from external sensors over a wireless network, rather than on optimizing the sensing fidelity of on-board sensors. More importantly, approaches such as Jigsaw or SociableSense consider only *extrinsic* sensor properties (such as its sampling rate); in contrast, ACQUA uses both the retrieved *values* of the sensor data tuples, and the intermediate query evaluation results, to alter the data retrieval and processing pipeline.

The BBQ [12] approach is among the closest to ACQUA, in its use of a model of the selectivity characteristics of each sensor source to optimize the data acquisition overhead. In particular, BBQ builds a multi-dimensional Gaussian probability density function of the sensors' likely data values, and then uses conditional probabilities to determine, in iterative fashion, the next sensor whose value is most likely to resolve a given query. ACQUA differs from [12] principally in its focus on continuous stream queries (as opposed to snapshot queries) and in the explicit consideration of the impact of batched acquisition of individual streams both on the wireless acquisition cost and the selectivity properties.

The exploitation of evaluation order in ACQUA is partly inspired by prior work on re-ordering expensive predicates in traditional relational database query processing [16, 17]. The key differences from the relational database setting is that (1) ACQUA addresses energy efficiency (as opposed to latency or throughput),

(2) ACQUA's query processing model is stream-based continuous queries (as opposed to standard SQL queries on relational data), and (3) ACQUA exploits not only evaluation order of predicates, but the order of acquisition of data stream segments as well.

3 The ACQUA functional architecture

We now consider the key properties that the ACQUA framework must consider, and then describe how the various functional components of ACQUA address these key design objectives. We first start by illustrating the basic principle of how the acquisition energy per sensor sample (i.e., a stream data tuple) and the selectivity characteristics of such tuples affect the ACQUA optimization framework.

Consider a hypothetical activity/wellness tracking application that seeks to detect an episode where an individual "walks for 10 minutes, while being exposed to an ambient temperature (95th percentile over the 10 minute window) of greater than 80 °F, while exhibiting an AVERAGE heart rate (over a 5 minute window) of >80 beats/min". Assume that this application uses an external wrist-worn device, equipped with accelerometer (sensor S_1, sampling at 100 samples/sec), heart rate (S_2, sampling at 1 sample/sec) and temperature (S_3, sampling at 1 sample/sec) sensors. We'll address many of the precise semantic aspects of this query later (e.g., do we use tumbling vs. moving windows for averaging?)—for now, note that is essentially a *conjunctive* query, where the context requires the simultaneous satisfaction of three separate predicates, related to accelerometer, HR and temperature data streams.

Assume that the probability of the accelerometer readings indicating that the user was walking for 10 min, denoted by $P(S_1)$, equals 0.95; likewise, the probability of '95th percentile of temp being greater than 80 °F', $P(S_2)$, equals 0.05 and the probability of 'AVG(HR)' being greater than 80, $P(S_3)$, equals 0.2. Furthermore, given the potentially different sample sizes and transmission rates for each sensor, let us assume that the acquisition energy costs, denoted by $E(S_i)$ are as follows: $E(S_1) = 0.2$ nJ/sample; $E(S_2) = 0.02$ nJ/sample and $E(S_3) = 0.01$ nJ/sample.

We then observe that the choice of the best acquisition sequence should take into account both the acquisition energy cost and the selectivity properties. More specifically, we should ideally retrieve the data chunk from the sensor that should have a low acquisition cost and also a high likelihood of helping to terminate the predicate evaluation. For the conjunctive query, we note that a single 'FALSE' predicate implies that the complex predicate is FALSE and that the subsequent steps of predicate evaluation can be aborted. Accordingly, in our formulation, we first compute the '*normalized acquisition cost*' (**NAC**) as a ratio of the acquisition cost normalized by the 'predicate being FALSE' probability. Accordingly, we get NAC(S_1) $= 100 \times 0.02/0.05$, NAC(S_2) $= 5 \times 0.02/0.95 = 0.105$ and NAC(S_3) $= 10 \times 0.01/0.8 = 0.125$. Based on these computations, it would follow that the best sequence of acquiring the sensor data streams for evaluating the conjunctive query above would be $\{S_2, S_3, S_1\}$.

Consider instead the *disjunctive* query counterpart that seeks to detect an episode where an individual was either walking for 10 min OR exposed to an ambient temperature (95th percentile over the 10 minute window) of greater than 80 °F, OR exhibited an AVERAGE heart rate (over a 5 minute window) of >80 beats/min". For such a disjunctive query, the processing can terminate as soon as there is a single 'TRUE' predicate. Accordingly, in this case, the NAC should be computed as a ratio of the acquisition cost normalized by the 'predicate being TRUE' probability. Plugging in the values from before, we have $\text{NAC}(S_1) = 100 \times 0.02/0.95 = 2.11$, $\text{NAC}(S_2) = 5 \times 0.02/0.05 = 2$ and $\text{NAC}(S_3) = 10 \times 0.01/0.2 = 0.5$. Accordingly, the best sequence of acquiring the sensor data streams is now $\{S_3, S_2, S_1\}$.

The above examples illustrate how the ACQUA framework needs to take into account both the stream's query selectivity properties as well as the different wireless communication costs (acquisition costs) associated with the different sensor streams. We next describe some of the additional real-life artifacts that the ACQUA framework must consider.

3.1 Functional requirements from the ACQUA framework

- **Accommodate Heterogeneity in Sensor Data Rates, Packet Sizes and Radio Characteristics:** Sensor data streams exhibit significant heterogeneity in terms of their sensor data rates (the number of sensor samples/sec), the data sizes (the bytes/sample) as well as the communication energy costs associated with their radio interfaces. As an illustration, Fig. 1 lists the data rates and sample sizes associated with a number of well-known medical and non-medical sensor streams. The communication energy costs will depend not just on the sensor type, but also on the specific wireless radio implementation on the embedded sensor device platforms. The ACQUA framework must thus be capable of incorporating different sensor data rates and wireless transmission characteristics in the query optimization framework.
- **Adapt to Dynamic Changes in Query Selectivity Properties:** To apply ACQUA, it is extremely important to have correct estimates for the query selectivity properties of different data streams. However, we need to keep in mind that these selectivity properties are not only individualized, but also vary dramatically over time due to changes in an individual's activity. For example, the likelihood of HR samples exceeding 80 might be very low when a person is engaged in sedentary office activity, but will be very high when the person is walking or working out in the gym. Accordingly, the ACQUA framework must be capable of using context to accurately predict (albeit statistically) the selectivity characteristics of different sensor streams.

Sensor type	Bits/ sensor channel	Channels/ device	Typical sampling frequency (Hz)
GPS	1408	1	1 Hz
SpO2	3000	1	3 Hz
ECG (cardiac)	12	6	256 Hz
Accelero-meter	64	3	100 Hz
Temperature	20	1	256 Hz

Fig. 1 Representative data rates for some common sensors

- **Take into Account other Objectives Besides Energy Minimization:** Operating with a heterogeneous set of sensors implies that energy minimization, while important, might not be the only objective of interest to a user of ACQUA. For example, it is possible that one of the N sensors might have very little battery capacity—in such a case, to extend the overall operational lifetime of the context detection activity, it might be more prudent to preferentially retrieve and process data from an alternative sensor, even though the selectivity characteristics of the alternative sensor may not be the highest.
- **Support Multiple Queries and Heterogeneous Time Window Semantics:** Mobile sensing & context awareness is now an intrinsic feature of a variety of smartphone applications, that may be potentially executing concurrently. Different applications may specify distinct predicates over a shared set of sensor streams—for example, the accelerometer sensor may be used to both evaluate step-counts in a wellness monitoring application and to understand the user's current mode of transport in a separate social networking application. The query predicates would differ not just in their predicate logic, but also in the time windows over which the stream query semantics are expressed. Accordingly, ACQUA must support a unified *application-independent* query representation framework and sequential data acquisition capability that is able of optimizing the evaluation sequence across *multiple concurrently executing stream queries*.

3.2 The ACQUA architecture

Figure 2 shows the ACQUA functional architecture for supporting the energy-efficient, dynamically varying, sequential data retrieval from different sensor streams. The figure describes the logically distinct components of the ACQUA architecture—as such, a specific implementation may implement multiple functional components

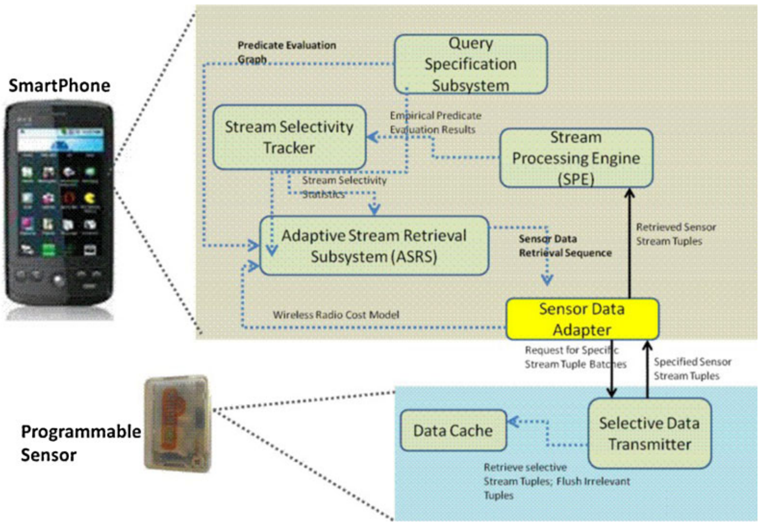

SmartPhone
Predicate Evaluation Graph
Query Specification Subsystem
Stream Selectivity Tracker
Empirical Predicate Evaluation Results
Stream Processing Engine (SPE)
Stream Selectivity Statistics
Adaptive Stream Retrieval Subsystem (ASRS)
Sensor Data Retrieval Sequence
Retrieved Sensor Stream Tuples
Wireless Radio Cost Model
Sensor Data Adapter
Request for Specific Stream Tuple Batches
Specified Sensor Stream Tuples
Programmable Sensor
Data Cache
Selective Data Transmitter
Retrieve selective Stream Tuples; Flush Irrelevant Tuples

Fig. 2 Functional component-level architecture of the ACQUA framework

separately or as a single sub-system (e.g., the *SelectivityTracker* component may be implemented either on the smartphone itself or on a back-end ‘cloud’ component). Moreover, the figure implicitly shows the interaction between a single smartphone and a single remote sensor; as mentioned in Sect. 1, ACQUA is equally applicable when the sensors are embedded in multiple nearby smartphones.

The heart of the ACQUA framework are the *Stream Selectivity Tracker (SST)* and the *Adaptive Stream Retrieval Subsystem (ASRS)* components. The SST is responsible for computing and establishing the selectivity properties of different sensor streams—in effect, computing the likely probability distribution of the values of each individual stream elements. To compute these values, ACQUA requires the SST to interface with the embedded Stream Processing Engine (SPE) to obtain the empirical observations of how the stream elements (individually or time-windows) satisfy different query predicates. The ASRS component is responsible for dynamically computing the sequence in which different (batches of) stream elements are retrieved by the smartphone from the locally-connected sensor. Note that the Stream Processing Engine (SPE) and the Sensor Data Adapter are pre-existing and non-ACQUA specific components needed to perform the basic functionality of (a) performing the appropriate query execution on the incoming data streams and (b) interfacing with the remote sensor(s) to retrieve the appropriate sensor samples. The Query Specification Subsystem is another ACQUA component that is responsible for receiving the various query specifications (associated with multiple applications) and for compiling them into a common Predicate Evaluation Graph. This graph is the data structure used by the ASRS algorithms to determine the preferred sequence in which data is pulled from individual sensor streams—the formal model for this graph will be presented shortly (in Sect. 4). To algorithmically determine the best evaluation sequence, the ASRS also requires the knowledge of the energy per sample profile associated with different sensor devices and radios—it receives these specifications from the corresponding Sensor Data Adapter. As mentioned before, the ACQUA framework requires some degree of embedded data processing and storage capability on each individual sensor. In particular, the sensor-resident ACQUA components include the Data Cache, which acts as a temporary local repository for the stream tuples that may or may not be eventually pulled by the smartphone, and the Selective Data Transmitter, which is responsible for receiving requests for specific subsets of the stream tuples and for transmitting (in batches) these requested subsets.

A fully functional ACQUA-based stream processing framework will require the implementation and integration of all these subsystems. The focus of this paper is, however, principally on the ASRS algorithms that determine the optimal data acquisition sequence, and on understanding the likely benefits of such selective retrieval of sensor data. Accordingly, for the rest of this paper, we will focus on the study of the ASRS algorithms, and implicitly assume the a-priori availability of the (a) stream selectivity statistics, (b) the predicate evaluation graph and (c) the wireless radio cost models (although we will study the creation of such cost models in Sect. 6).

4 The stream-oriented query model

We now focus on the basic ASRS algorithms for acquisition-cost-aware query processing. We first need to mathematically rigorously define the types of queries that we consider.

Query specification For this paper, we consider complex stream queries that are expressed as arbitrary conjunction or disjunction predicates over a set of stream-oriented SQL aggregate (e.g., MAX or AVG) or user-defined (e.g., determining the Fourier coefficients) functions, defined over a time-window of each individual sensor stream. Mathematically, an individual query specification Q can be formally expressed as:

$$\begin{aligned}
\mathrm{Q} &::= \mathit{Predicate} \mid (\,\mathrm{Q}\ \ \textbf{AND}\ \ \mathrm{Q}\,) \mid (\,\mathrm{Q}\ \ \textbf{OR}\ \ \mathrm{Q}\,)\\
\mathit{Predicate} &::= \mathit{AggFunc}\ (\ \mathit{SExp}, w\ )\ \mathit{CmpOp}\ \mathit{Const} \mid\ \textbf{NOT}\ \ \mathit{Predicate}\\
\mathit{SExp} &::= \mathit{StreamName} \mid \mathit{StreamExp}\ \mathit{ArithmeticOp}\ \mathit{Numeric}
\end{aligned}$$

where *AggFunction* can be any SQL aggregate function or user defined function applied over a time window $(t - w, t)$ of stream values (t being the current time), *CmpOp* can be any comparison operator such as $\{>, <, =\}$, *Const* denote a constant of any appropriate type, *StreamName* uniquely identifies a stream, *ArithmeticOp* denotes an arithmetic operator $\{+, -, \div, \times\}$, and *Numeric* denote a numeric constant. Three example queries are as follows:

Q1: $\mathrm{AVG}(A, 5) < 70$ AND ($\mathrm{MAX}(B, 4) > 100$ OR $C < 3$),
Q2: ($\mathrm{AVG}(A, 5) < 70$ AND $\mathrm{MAX}(B, 4) > 100$) OR
($C < 3$ AND $\mathrm{Speed}(D, 2) < 1.0$),
Q3: ($\mathrm{AVG}(A, 5) < 70$ AND $\mathrm{MAX}(B, 4) > 100$) OR
($C < 3$ AND $\mathrm{MIN}(B, 7) < 80$),

Query Q2 illustrates a query involving a user defined function for computing the average speed over the last two seconds of the values of stream D. Stream D could be displacement samples in the x, y, z axes from an accelerometer. Query Q3 illustrates a query where stream B appears in two predicates with different window sizes.

Evaluation period ω In this paper, we consider queries defined over *tumbling windows* of the individual sensor data streams. Formally, this implies the notion of a 'time shift' value $\omega(Q)$ associated with a query Q, such that the query is evaluated repeatedly at the time instants $t = (\omega, 2\omega, 3\omega, \ldots)$. Note that the time-shift value ω is distinct from the time windows associated with the individual predicates and operators of the query Q. For example, a specific query may be defined to perform an AVG(5) operation (i.e., an average of the last 5 seconds worth of sensor data with $\omega = 7$; in this case, the query would be evaluated at $t = 0$ over the stream tuples belonging to the time window $(-5, 0)$, and then again at time $t = 7$ over the time window $(2, 7)$. Figure 4 illustrates this relationship between the query *time shift* value, the predicate time windows and the individual stream tuple-generation rates—the first three figures

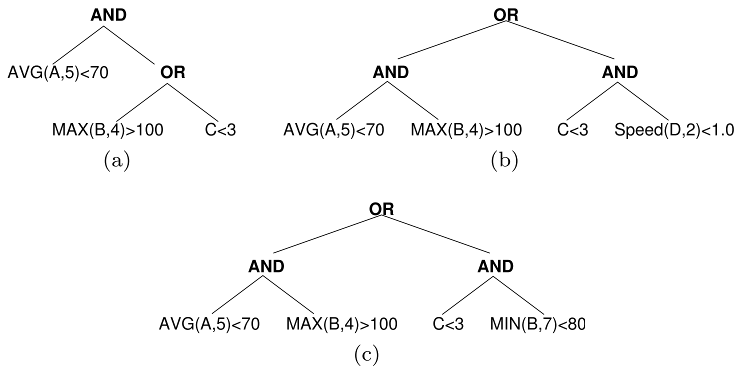


Fig. 3 Query trees for three example queries

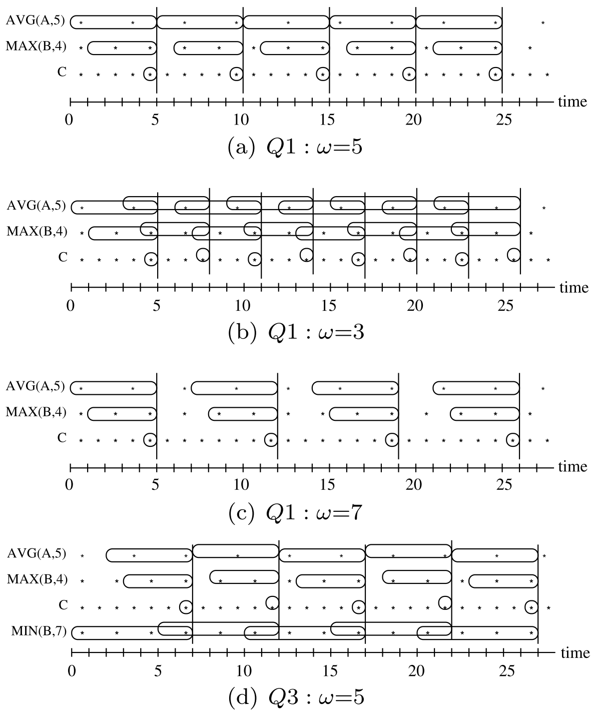


Fig. 4 Relationship between query evaluation period, predicate time windows, and stream rates. An *asterisk* denotes a sampled tuple of a particular stream at a particular time. The *rounded rectangles* denote the sample or the time window of samples required in the evaluation at each evaluation event

illustrate different cases for the query Q1 (Fig. 3(a)), while the fourth figure illustrates the evaluation for query Q3 (Fig. 3(c)) with $\omega = 5$.

An important consequence of this evaluation mode is the fact that *the stream tuples needed for the evaluation of the query at a particular time instant may or may not be distinct from the tuples needed at a subsequent time instant.* Consider query Q3 for the evaluation schedule in Fig. 4(d). Suppose each stream is associated with a buffer and the buffers are initially empty. At time $t = 7$, suppose the evaluation of Q3 requires the retrieval of the data elements $A : (2, 7]$, $B : (3, 7]$, $C : (6, 7]$. Now, at time $t = 12$, suppose $\text{MAX}(B, 4) > 100$ is evaluated first and is false; we need proceed to evaluate the right subtree of the top OR node (in Fig. 3(c)). Note now that the evaluation of $\text{MIN}(B, 7) < 80$ requires the stream tuples $B : (5, 12)$. However, a subset of this required set of tuples has already been previously acquired—only the samples $B : (7, 12]$ need to be acquired. *This example illustrates that, even with tumbling window queries, the acquisition cost for a particular sensor stream may be different at different evaluation instants, depending upon the data tuples that may have been acquired during prior event processing.*

5 The ASRS sequential retrieval algorithm

Having formally defined the semantics of our query, we now proceed to define the algorithm for computing the preferred data retrieval and evaluation sequence. A query is first compiled into a uniform *Query Tree* representation. We will first describe the algorithms using a query tree representation that directly corresponds to the specified query, and later describe enhancements that result from the transformation of the query tree to a specific Disjunctive Normal Form (DNF).

The root of a query tree represents the entire set of concurrent query predicates. An internal node is associated with a Boolean conjunction or disjunction operator and a leaf node is associated with a predicate. Figure 3 illustrates the query trees for the three example queries Q1, Q2, and Q3. For purposes of simplifying the exposition, we also make the following two assumptions in this section: (a) Each unique sensor stream in the query tree is associated with a single 'tumbling window' value, even though the same window of the sensor data can appear in multiple nodes of the query tree and be associated with multiple predicates, and (b) The unique sensor-specific 'tumbling window' value defines the basic batch size in which the smartphone's Sensor Data Adapter retrieves data from each sensor. (We will later relax the first assumption, in Sect. 5.3.)

5.1 ASRS query evaluation algorithm

Algorithm 1 defines the high-level logic of query evaluation. Intuitively, following the approach discussed in Sect. 3, the algorithm first computes the lowest expected cost of evaluating different portions of the query sub-trees, and thereby determines (using

Algorithm 1 PROCESSQUERY(q, P, ω)

Input: Query tree q, probability P of each subquery evaluating to true/false, evaluation period ω

Output: Alert Stream

```
1: loop
2:     $t \leftarrow$ current time
3:     CALCACQCOST($q, t, P, C$)
4:     if EVALQUERY($q, t, P, C$) = $true$ then
5:         output alert tuple
6:     sleep $\omega$ seconds
```

Algorithm 2 CALCACQCOST(q, t, P, C)

```
Input: Query tree $q$, current time $t$, probability function $P$, data acquisition cost function $C(\cdot)$
Output: Updates cost function $C(\cdot)$
1:  if $q$ is a predicate node then
2:      let $s$ be the stream that $q$ operates on, $w$ be the window size of $q$, $t_s$ be the latest time
        the buffer for $s$ was updated.
3:      $C(q) \leftarrow$ Calculate cost for acquiring the samples in time interval $(\max(t - w, t_s), t]$
        for stream $s$. (This is based on the transmission cost model for the specific wireless
        technology. Section 6 will detail this cost model, for WiFi and Bluetooth, in Eqs. (7)
        & (8), respectively.)
4:  else
5:      CALCACQCOST($q.left, t, P, C$)
6:      CALCACQCOST($q.right, t, P, C$)
7:      if $q.op =$ AND then
8:          $C(q) \leftarrow$ Eq. (1)
9:      else
10:         $C(q) \leftarrow$ Eq. (2)
```

the recursive Algorithm 2 CALCACQCOST) the optimal sequence for retrieving the data from the different sensor streams. Subsequently, the actual query is evaluated using the recursive Algorithm 3 EVALQUERY, which essentially follows the specified sequence to evaluate the sub-trees. As mentioned previously, the actual retrieval of data tuples for a given stream needs to consider the relevant tuples that have already been retrieved (at prior instants or while processing other parts of the query tree): Line 3 in Algorithm 3 achieves this by adjusting the window of data tuples that are actually retrieved from the corresponding sensor.

Algorithm 2 CALCACQCOST computes the acquisition cost of evaluating a query subtree according to the evaluation sequence determined by NAC. This computation uses the per-sample data transmission cost, which we assume has been provided to ACQUA. At a query tree node with an AND operator, we recursively calculate the data acquisition cost of the left and right subtrees (henceforth denoted by L and R respectively). Since the acquisition cost of the current node is dependent on the evaluation order of its children, we use the NAC for the left and right children to determine the evaluation order. The NAC for L and R are computed using the probability of L and R evaluating to true or false. Note that the NAC for the children can be computed, because the acquisition cost of the children has already been computed recursively. Once the order of evaluation (LR or RL) of the children is computed, we can now calculate the acquisition cost for a query node with an AND operator as,

$$C(q) = \begin{cases} P(q.\mathit{left}) \times [C(q.\mathit{left}) + C(q.\mathit{right})] & \\ \quad + P(\neg q.\mathit{left}) \times C(q.\mathit{left}) & \text{if LR} \\ P(q.\mathit{right}) \times [C(q.\mathit{left}) + C(q.\mathit{right})] & \\ \quad + P(\neg q.\mathit{right}) \times C(q.\mathit{right}) & \text{if RL} \end{cases} \tag{1}$$

Algorithm 3 EVALQUERY(q, t, P, C)

```
Input: Query tree $q$, current time $t$, probability function $P$, data acquisition cost function $C(\cdot)$
Output: Truth value of $q$
1: if $q$ is a predicate node then
2:     let $s$ be the stream that $q$ operates on, $w$ be the window size of $q$, $t_s$ be the latest time the buffer for $s$ was updated.
3:     Acquire the samples in time interval $(\max(t-w, t_s), t]$ for stream $s$.
4:     Update $C(\cdot)$ if $s$ is used in multiple predicates
5:     $truthval \leftarrow$ evaluate predicate $q$
6:     return $truthval$
7: else
8:     if $q.op =$ AND then
9:         $leftshortcircuits \leftarrow P(\neg q.left)$
10:        $rightshortcircuits \leftarrow P(\neg q.right)$
11:        $shortcircuitval \leftarrow false$
12:    else
13:        $leftshortcircuits \leftarrow P(q.left)$
14:        $rightshortcircuits \leftarrow P(q.right)$
15:        $shortcircuitval \leftarrow true$
16:    $evalorder \leftarrow (q.left, q.right)$
17:    if $\frac{C(q.left)}{leftshortcircuits} > \frac{C(q.right)}{rightshortcircuits}$ then
18:        $evalorder \leftarrow (q.right, q.left)$
19:    for all $q' \in evalorder$ do
20:        $truthval \leftarrow$ EVALQUERY$(q', t, P, C)$
21:        if $truthval = shortcircuitval$ then
22:            return $truthval$
23:    return $\neg shortcircuitval$
```

A similar analysis can be applied for a query node with an OR operator, resulting in a acquisition cost of,

$$C(q) = \begin{cases} P(\neg q.left) \times [C(q.left) + C(q.right)] & \\ \quad + P(q.left) \times C(q.left) & \text{if LR} \\ P(\neg q.right) \times [C(q.left) + C(q.right)] & \\ \quad + P(q.right) \times C(q.right) & \text{if RL} \end{cases} \tag{2}$$

We note that the calculation is dependent on the probability function $P(\cdot)$ for each node in the query tree evaluating to true or false. These probabilities can be obtained statically from historical executions or more dynamically by keeping counters for the truth value of the evaluation of each query tree node.

The recursion ends when a predicate node is reached. A predicate node is associated with a stream and the cost of evaluating a predicate node is calculated using Eqs. (7) or (8) depending on the transmission type (802.11 or Bluetooth), assuming the current state of the buffer associated with the stream. No actual data acquisition occurs in this computation. The result of CALCACQCOST is that the acquisition cost function $C(\cdot)$ is now updated for each node in the query tree based on the current snapshot of the buffers for all the dependent streams. We are now ready to evaluate the query using the updated cost function $C(\cdot)$ and the probabilities function $P(\cdot)$ for each node in the query tree.

The actual acquisition of sensor data and the evaluation of the predicates occur in the recursive Algorithm 3 EVALQUERY. The base case occurs at the predicate (leaf) nodes of the query tree. The required data tuples are retrieved from the dependent stream if they are not already in the stream buffer (Line 3). In queries where a particular stream is involved in multiple predicates, a data acquisition may change the acquisition cost of another predicate on the same stream. Line 4 updates the cost function $C(\cdot)$ for the query tree nodes affected by the stream buffer update. Finally, the predicate is evaluated and the truth value returned.

For the recursive case, EVALQUERY computes the NAC of the query node's left and right subtrees using $C(\cdot)$ and $P(\cdot)$. The subtree with the lower NAC is recursively evaluated first. If the truth value of the evaluation results in a short circuit, the other subtree need not be evaluated and hence no data is acquired for the that subtree.

5.2 Using disjunctive normal form

A key limitation of the binary query tree representation presented thus far is that it is unable to capture all possible permutations of sensor data acquisition in the solution space. Consider the following query.

Q4: AVG$(A, 5)<70$ OR (MAX$(B, 4)>100$ AND ($C <3$ AND MIN$(D, 3) < 15$))

Figure 5(a) shows the *binary* query tree representation of the query. Since EVALQUERY re-orders the left and right children of each internal node according to the NAC, the number of possible ordering (permutations) for the binary query tree of Q4 is $2 \times 2 \times 2 = 8$, since each of the three internal node can permute its left and right children.

Suppose query Q4 is rewritten to an equivalent disjunctive normal form (DNF) and represented as a DNF tree as shown in Fig. 5(b). A DNF tree is a three level tree consisting of an OR operator at the root, AND operator at the second level, and predicates at the leaf level. The number of different permutations for the DNF tree with two internal nodes is $2 \times 3! = 12$. Hence, while the two representation are equivalent in terms of the semantics of the query (i.e. the truth table), they are not equivalent in terms of the space of possible evaluation order. More specifically, from Fig. 5(a), we can see that ACQUA's depth-first evaluation algorithm, will be unable

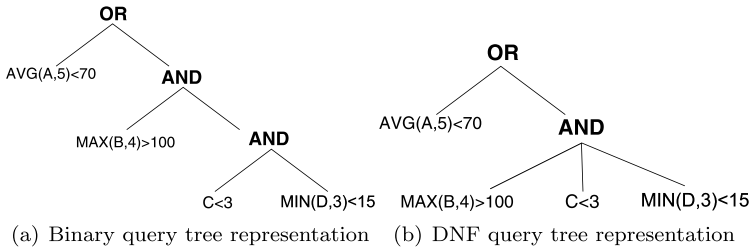


Fig. 5 The binary query tree and DNF query tree representations of query Q4

Algorithm 4 CALCACQCOSTDNF(q, t, P, C)

```
Input: Query tree $q$, current time $t$, probability function $P$, data acquisition cost function $C(\cdot)$
Output: Updates cost function $C(\cdot)$
1: if $q$ is a predicate node then
2:     let $s$ be the stream that $q$ operates on, $w$ be the window size of $q$, $t_s$ be the latest time
       the buffer for $s$ was updated.
3:     $C(q) \leftarrow$ Calculate cost for acquiring the samples in time interval $(\max(t-w, t_s), t]$ for
       stream $s$ using Eqs. (7) or (8).
4: else
5:     for all $i \in q.children$ do
6:         CALCACQCOSTDNF$(i, t, P, C)$
7:     if $q.op =$ AND then
8:         $C(q) \leftarrow$ Eqs. (3) & (5)
9:     else
10:        $C(q) \leftarrow$ Eqs. (3) & (4)
```

to execute the valid evaluation sequence $\{(\mathrm{MAX}(B, 4) > 100), (\mathrm{MIN}(D, 3) < 15), (\mathrm{AVG}(E, 7) > 50)\}$. To address that limitation, we modify our algorithms to use a DNF tree representation of the query instead.

With the new DNF query tree representation, the main algorithm PROCESSQUERY requires no modification, but the CALCACQCOST and EVALQUERY algorithms are modified to accommodate a DNF tree and they are named CALCACQCOSTDNF and EVALQUERYDNF respectively.

Algorithm 4 CALCACQCOSTDNF computes the data acquisition cost of query evaluation for each node in the DNF query tree. For each node with OR or AND operator, CALCACQCOSTDNF recursively computes the data acquisition cost of all subtrees. Once the acquisition cost of the subtrees are computed, the NAC of the subtrees are computed using the probabilities associated with the subtrees, and an evaluation order for the subtrees are computed. Given an evaluation order of the subtrees, the acquisition cost of the current node, denoted by q, is computed using the following recurrence relations,

$$Cost(q) = \begin{cases} \text{Eqs. (7) or (8)} & \text{if } q \text{ is predicate} \\ C_{OR}(children(q)) & \text{if } q \text{ is OR node} \\ C_{AND}(children(q)) & \text{if } q \text{ is AND node} \end{cases} \tag{3}$$

$$C_{OR}(q_i, \ldots, q_j) = \begin{cases} Cost(q_i) & \text{if } i = j \\ P(q_i) \times Cost(q_i) & \text{otherwise} \\ \quad + P(\neg q_{i,j}) \times C_{OR}(q_{i+1}, \ldots, q_j) & \end{cases} \tag{4}$$

$$C_{AND}(q_i, \ldots, q_j) = \begin{cases} Cost(q_i) & \text{if } i = j \\ P(\neg q_i) \times Cost(q_i) & \text{otherwise} \\ \quad + P(q_{i,j}) \times C_{AND}(q_{i+1}, \ldots, q_j) & \end{cases} \tag{5}$$

Algorithm 5 EVALQUERYDNF(q, t, P, C)

```
Input: Query tree $q$, current time $t$, probability function $P$, data acquisition cost function $C(\cdot)$
Output: Truth value of $q$
 1: if $q$ is a predicate node then
 2:     let $s$ be the stream that $q$ operates on, $w$ be the window size of $q$, $t_s$ be the latest time
        the buffer for $s$ was updated.
 3:     Acquire the samples in time interval $(\max(t - w, t_s), t]$ for stream $s$.
 4:     Update $C(\cdot)$ if $s$ is used in multiple predicates
 5:     $truthval \leftarrow$ evaluate predicate $q$
 6:     return $truthval$
 7: else
 8:     if $q.op =$ AND then
 9:         for all $i \in q.children$ do
10:             $\text{NAC}_{AND}(i) \leftarrow \frac{C(i)}{P(i=false)}$
11:         $evalorder \leftarrow qchildren$ sorted by $\text{NAC}_{AND}(\cdot)$ ascending
12:         for all $q' \in evalorder$ do
13:             $truthval \leftarrow$ EVALQUERYDNF($q', t, P, C$)
14:             if $truthval = false$ then
15:                 return $false$
16:         return $true$
17:     else
18:         for all $i \in qchildren$ do
19:             $\text{NAC}_{OR}(i) \leftarrow \frac{C(i)}{P(i=true)}$
20:         $evalorder \leftarrow qchildren$ sorted by $\text{NAC}_{OR}(\cdot)$ ascending
21:         for all $q' \in evalorder$ do
22:             $truthval \leftarrow$ EVALQUERYDNF($q', t, P, C$)
23:             if $truthval = true$ then
24:                 return $true$
25:         return $false$
```

where $children(q)$ denotes the collection of children nodes $q_i, q_{i+1}, \ldots, q_j$ of node q.

Algorithm 5 EVALQUERYDNF is similar to EVALQUERY except that the NAC of all the children nodes are computed, the children nodes are ranked using the NAC, and the children nodes are recursively evaluated according to the ranked order with shortcircuiting.

5.3 Optimizing for multiple predicates on the same stream

The ACQUA algorithms presented thus far implicitly assume that a particular sensor stream is associated with only a single unique predicate in the query tree (e.g., sensor A is associated only with the predicate $\text{AVG}(A, 5) < 70$ in the query trees illustrated in Fig. 3). In reality, especially when the same sensor is used by multiple concurrent applications, a particular sensor data stream may participate in multiple predicates in the composite DNF query. Moreover, these predicates on the same stream may specify different window sizes on that stream. In the top-down evaluation procedures as outlined in EVALQUERY and EVALQUERYDNF (Algorithms 3–5), the evaluation order is based on sorting the predicates based on the NAC regardless

Algorithm 6 PROCESSQUERYBOTTOMUP(q, P, ω)

Input: Query tree q, probability P of each subquery evaluating to true/false, evaluation period ω

Output: Alert Stream

```
 1: loop
 2:    $t \leftarrow$ current time
 3:    $SOrder \leftarrow$ sort input streams according to Eq. (6)
 4:    for all $s \in SOrder$ do
 5:       Acquire the maximum window of data from $s$
 6:       for all predicate $i$ dependent on $s$ do
 7:          Evaluate predicate $i$
 8:          Propagate truth value of $i$ up the DNF query tree $q$
 9:          if $q = true$ then
10:             output alert tuple
11:             goto Line 16
12:          else if $q = false$ then
13:             goto Line 16
14:          else
15:             continue
16:    sleep $\omega$ seconds
```

of whether the *different predicates are accessing the same data streams*. To ensure that the NAC is accurate for a predicate of which data has already been partially acquired by the evaluation of a predicate preceding it in the evaluation order, a stop-gap measure of re-calculating the NAC for all unevaluated predicates which share data streams with at least one other predicate is used (Algorithm 3, Line 4 and Algorithm 5, Line 4). Consider query Q5 as an illustrative example. Stream A is used by the two predicates, $\text{AVG}(A, 10) > 70$ and $\text{MAX}(A, 3) > 100$. If at time t, the predicate $\text{AVG}(A, 10) > 70$ is evaluated first, then the data acquisition cost for $\text{MAX}(A, 3) > 100$ is zero, because the buffer contains all the tuples from time $t - 10$ to t which subsumes the window required by the $\text{MAX}(A, 3) > 100$.

Re-calculating the NAC after the evaluation of each predicate is, however, clearly an inefficient approach and would be impractically expensive for the Stream Processing Engine. In this sub-section, we thus investigate an alternate approach based on *bottom-up evaluation* of the DNF query tree. The new algorithm, PROCESSQUERYBOTTOMUP, is outlined in Algorithm 6. The first key difference with the previous algorithms is that the list of streams required by the given query is first sorted (Line 3) to get a stream acquisition order. Intuitively, the stream acquisition order needs to capture two criteria:

1. Streams that enable more predicates in the query to be evaluated should be ranked higher.
2. Streams that enable higher probability of shortcircuiting the evaluation should be ranked higher.

Clearly, a naive approach to search through all possible permutations of stream acquisition order is impractical, and we design a heuristic ranking function instead. For a DNF query tree q, let q_i denote the i-th AND node, q_{ij} denote the j-th predicate node

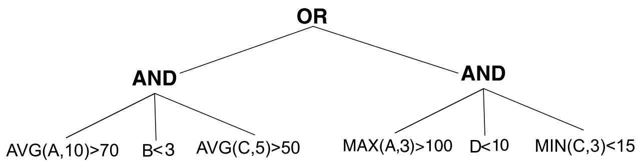


Fig. 6 The DNF query tree representation of query Q5

of the i-th AND node, $Q(s)$ be the set of predicate nodes that depend on stream s. The rank of a stream s is defined as

$$R(s) = \frac{1}{C_{max}(s)} \sum_{q_{ij} \in Q(s)} P(\neg q_{ij}) \times n_i, \tag{6}$$

where $C_{max}(s)$ denotes the maximum window size for stream s over all the predicates that depend on stream s, n_i denotes the number of predicate nodes that are children of the i-th AND node, and N denotes the total number of predicates or leaves in the DNF query tree. The intuition for the heuristic is as follows. If there are many predicates that depend on stream s, the size of $Q(s)$ would be larger, and hence $R(s)$ would be larger. If a predicate q_{ij} has high probability of being false, it would more likely shortcircuit the q_i AND node, hence it should be ranked higher. If a predicate q_{ij} can shortcircuit an q_i AND node with a large number of predicates (i.e. n_i is large), it should be ranked higher. There are probably alternate heuristic functions that would work as well and it is not within the scope of this paper to investigate the spectrum of heuristic functions.

After obtaining the stream acquisition order, we start to evaluate the query in a bottom-up fashion. For each stream in the stream acquisition order, we acquire the maximum window of data that is needed by all the predicates dependent on that stream. For each of those dependent predicates, we then evaluate the predicate and propagate the truth value up the DNF query tree to update the truth value of the nodes in the leaf-to-root path. The truth value of each node in the DNF tree is initialized to value 'unknown' at the beginning of each evaluation period. Evaluating a predicate is equivalent to determining the truth value of a leaf node in the DNF tree. This truth value is propagated up the tree to determine if the truth value of the root node can be decided taking into account the usual shortcircuiting rules. Once the truth value of the root node is determined to be 'true' or 'false', no further sensor data acquisition and predicate evaluation is needed.

As an illustrative example, consider the following query (Fig. 6 shows the corresponding DNF query tree).

Q5: (AVG(A, 10)>70 AND B <3 AND (AVG(C, 5)<15) OR
(MAX(A, 3)>100 AND D <10 AND (AVG(C, 3)<15)

Using Algorithm 6 PROCESSQUERYBOTTOMUP, we first sort the streams using the heuristic ranking function (Eq. (6)). For query Q5, the input streams are A, B, C, D. Suppose the ranking is,

$$R(A) > R(C) > R(B) > R(D).$$

We then evaluate all the predicates associated with stream A first, followed by B, then C and D. We first acquire the maximum window of data from stream A which is $\max(10, 3) = 10$. For stream A, suppose we first evaluate the predicate $\mathrm{MAX}(A, 3) > 100$. The truth value of the predicate is then propagated up the DNF query tree. Suppose the predicate is false, the AND-node (the second AND in the tree) that is the parent of the predicate is then shortcuited and becomes false. The truth value of the root OR-node, however, is still unknown. Hence the algorithm proceeds to the next predicate associated with stream A, namely, $\mathrm{AVG}(A, 10) > 70$. Note that the evaluation order of the predicates associated with a stream no longer matters from an energy perspective, because all the data required for their evaluation has been acquired. Suppose the predicate $\mathrm{AVG}(A, 10) > 70$ evaluates to true, the truth value is propagated up the tree. This time no shortcircuiting happened and the truth value of the root is still unknown, so the algorithm proceeds to the next stream. The maximum window of data, namely 5 seconds, for stream C is acquired, and the algorithm proceeds to evaluate the predicates $\mathrm{AVG}(C, 5) > 50$ and $\mathrm{MIN}(C, 3) < 15$. Suppose $\mathrm{AVG}(C, 5) > 50$ is false. The truth value is propagated up the DNF tree. This time the AND-node (the first AND in the tree) that is the parent of the predicate is shortcircuited and becomes false. Since both AND-nodes in the DNF tree are now false, the truth value of the root OR-node is false. No further data acquisition or predicate evaluation is required for this evaluation instant ω.

6 Wireless technologies & the per-sample data acquisition cost

The ACQUA algorithms described so far assume a specific energy cost associated with the acquisition of each individual sensor stream tuple. In this section, we now consider the challenge of characterizing this cost, for the practical scenario where a smartphone is retrieving data from nearby, possibly wearable sensors, using a local wireless technology. In all practical cases of interest, the query processing engine will acquire data intermittently and in *batches*, i.e., transfer multiple sensor samples as part of a single transmission activity. We shall show that the accurate computation of this energy cost is not trivial, and must consider key technology-specific characteristics of the wireless radios.

To study this phenomena in detail, we utilize prior work that accurately captures the key characteristics of two specific wireless technologies—WiFi (IEEE 802.11) and Bluetooth. Although the transmission power and energy associated with data transfers, as well as the link bandwidth, will be unique for each specific technology, we believe that these two widely-used radio technologies represent two broad classes of PAN wireless technologies. In particular, IEEE 802.11 represents a high-power, high-data rate PAN technology, while Bluetooth represents a low-power, low-data rate alternative.

As we will shortly see, both of these technologies have two distinct modes—a low-power ‘idle’ mode (where the radio lies dormant and consumes significantly lower power) and an ‘active’ mode (where the radio is actually capable of engaging in packet transmission or reception activity). In general, let P_a be the power consumption in active mode, and P_i ($P_i \ll P_a$) be the power consumed in the ‘idle’ mode. Also, let B be the transmission bandwidth (bps) of the radio link, when active. We consider the case of a generic sensor, operating under a sampling frequency of f Hz with a sample size of S bits (resulting in a data generation rate, R, given by $R = f * S$). We consider the communication energy overhead as a function of N, the number of sensor samples that are batched by the sensor and then transmitted in a burst to the smartphone. Figure 8 summarizes the key parameters associated with batched transmission in 802.11 and Bluetooth, which we now discuss.

IEEE 802.11 Commercial IEEE 802.11 radios can operate in two states—a normal 'active' mode (when the radio interface receives or transmits packets) and a Power Save Mode (PSM), where the radio periodically wakes up to check if there any pending transmissions or receptions. The following are two key relevant properties associated with 802.11 hardware:

– Due to the switching characteristics of the radio hardware, there is typically a lower bound on the minimal idle time Th_{idle}, below which the radio cannot enter the PSM mode (typically, this is around 100 ms) [8, 10].
– There is a fixed, *duration-independent* switching energy E_{switch} spent when a radio transitions from the PSM to the 'active' mode.

Accordingly, it follows that the total transmission time for the N samples, generated over a time interval of $\frac{N}{f}$, equals $\frac{N*S}{B}$ and the total energy E_t consumed over this time interval equals:

$$E_t = \begin{cases} P_i * (\frac{N}{f} - \frac{N*S}{B}) + P_a * \frac{N*S}{B} + E_{switch} & \text{if } \frac{N}{f} - \frac{N*S}{B} > Th_{idle} \\ P_a * \frac{N}{f} & \text{otherwise} \end{cases} \tag{7}$$

The second case corresponds to the situation where the 'idle' time for the 802.11 radio is not enough for it to switch into the low-power PSM state. On the other hand, if the idle time is large enough, the energy per sample progressively diminishes, as the fixed cost of switching to a low power state is amortized over a longer 'idle' time.

Bluetooth Bluetooth radios typically operate in three states: transmit, receive or sleep, each of which has a different power consumption profile [11]. As we focus principally on the smartphone, which primarily receives data from an external sensor, we denote its active energy consumption P_a as the energy spent in actively receiving data. We consider the Bluetooth version 2.0+ EDR and assume, for analytical tractability, that a single sensor device attaches as a slave to the master located on the smartphone. While the low-power mode results in significantly low power consumption, note that there is a latency T_{switch} involved in switching from the non-associated low-power mode to the associated-active mode. Accordingly, any data transfer duration would consist of the total time spent in transfer $\frac{N*S}{B}$, plus the additional time T_{switch}. Accordingly, the total energy consumed in transmitting the sensor stream in batches of N samples is given by:

$$E_t = P_i * \left(\frac{N}{f} - \frac{N*S}{B} - T_{switch}\right) + P_a * (\frac{N*S}{B} + T_{switch}) \tag{8}$$

Figure 7 plots the resulting energy overhead (energy per sample) for both IEEE 802.11 and Bluetooth (computed by using Eqs. (7) and (8)), as a function of the

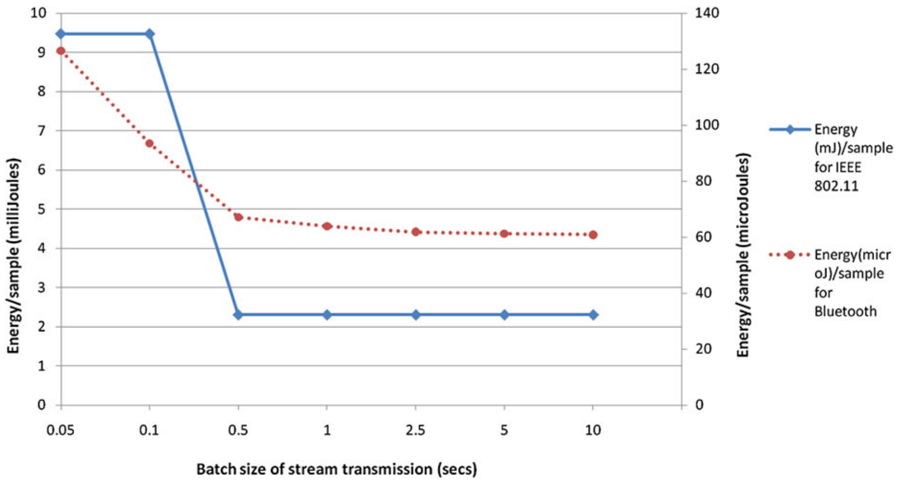


Fig. 7 The impact of batched transmissions on the transmission energy overhead per sample. The figure plots the energy/sample for both 802.11 and Bluetooth interfaces, as a function of the batch duration, for a typical accelerometer sensor

Fig. 8 The energy overheads for IEEE 802.11g & Bluetooth radios

	IEEE 802.11	Bluetooth 2.0+EDR
P_a	947 mW	60 mW
P_i	231 mW	5 mW
B	54 Mbps	1 Mbps
E_{switch}	14 µJoule	–
Th_{idle}	100 ms	–
T_{switch}	–	6 msec

batch size N, for a representative accelerometer sensor, with $S = 192$ bits/sample and $f = 100$ Hz. It is clear that the choice of the batch size N, for a given radio technology, has a significant effect on the energy efficiency of the data acquisition process, and is thus an important design parameter for the ACQUA framework. For the specific accelerometer sensor considered here, the energy overhead for 802.11-based transmissions becomes dramatically lower when the sensor tuples are transferred in batches of 500 msec or greater; for the Bluetooth interface, a batch duration of 5 sec or higher is more efficient.

7 Performance evaluation and results

We now describe the result of simulation-based studies to quantify the performance gains (in terms of the reduction in energy overheads) of our proposed ACQUA algorithms. Our studies are conducted using a Perl-based simulator which accepts as input both a query tree and probability distributions on the values of individual data streams. The simulation results use the energy-per-bit cost models derived in Sect. 6

for 802.11 wifi and Bluetooth. Synthetic traces of sensor-generated data tuples were then generated to reflect the probability distributions and fed into the simulator, which then applied the algorithms of Sect. 5 to compute the sequence of data that would be actually retrieved by an ACQUA-based implementation. Results are presented by averaging over 5 one-hour long traces and also include the 95 % confidence intervals.

To quantify our performance gains, we compared five different evaluation algorithms:

1. **Naive:** The naive retrieval algorithm requires each sensor to simply upload (in batched mode) its generated stream tuples to the SPE. Accordingly, while this algorithm utilizes batched transmission to reduce the energy overheads associated with the use of the PAN wireless interface, it does not exploit the selectivity properties to reduce the amount of sensor data that is actually needed by the SPE.
2. **ASRS-dynamic:** The ASRS-dynamic algorithm corresponds to the procedure described in Sect. 5 and requires the dynamic modification of the acquisition cost functions after each data retrieval and evaluation, to account for both the stream tuples already present in the smartphone buffer and the already-resolved (‘short-circuited’) query subtrees.
3. **ASRS-static:** While this algorithm’s logic is broadly similar to ASRS-dynamic, it computes an optimal sequence only once (at the beginning of the simulation) based on the selectivity characteristics and the communication costs, and then applies the EVALQUERY procedure to evaluate the query tree at successive ‘time shift’ instants. Accordingly, it does not perform the dynamic update of NAC values, based on the dynamically evolving state of the query processing state.
4. **ASRS-DNF:** This algorithm is a modification of ASRS-dynamic methods that uses DNF query trees instead of binary query trees. Since DNF query trees capture a bigger set of possible evaluation orders, we expect the algorithm to find an ordering that is more energy efficient than ASRS-dynamic.
5. **ASRS-MultiPred:** This algorithm is fundamentally different from the previous algorithms in that a bottom-up query evaluation strategy is used to further optimize the data acquisition order for sensor streams that are used by multiple predicates.

While *Naive* helps to quantify the performance gains expected from our sequential acquisition strategy, ASRS-static helps us to isolate and understand the performance gains that arise from the dynamic consideration of the evolving query state. ASRS-DNF helps us to determine the performance gains from using a disjunctive normal form representation of the query tree. Whereas the other algorithms use top-down query evaluation, ASRS-MultiPred uses a bottom-up query evaluation that exposes additional optimization opportunities.

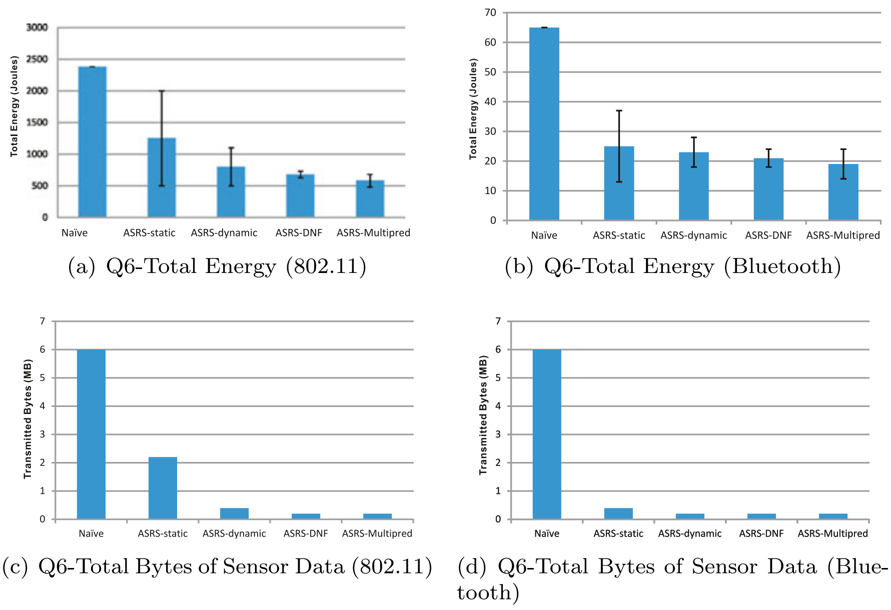
Given our focus on understanding the expected benefits of our proposed ASRS algorithms, we experimented with a large number of different queries, both manually crafted as well as synthetically generated. For ease of exposition, we focus on one relatively-simple but representative query that generates an alert if:

Q6: (((AVG(SP02,5)$<$98 %) AND (SPREAD(Accel,10)$<2g$))
AND (AVG(HR,10)$<$75)) OR ((AVG(SP02,10)$<$95)
AND (SPREAD(Accel,10)$>4g$) AND (AVG(HR,10)$>$100)).

Intuitively, query Q6 generates alerts either if the user's Sp02 values drop below 98 % while the user is resting, or if the Sp02 values drop below 95 % while the individual is engaged in vigorous activity (e.g., running). The accelerometer and Sp02 sampling rates and data sizes are adapted from Fig. 1, while the heart rate sensor has a sampling frequency of 0.5 Hz and a sample size of 32 bits. We experimented with both 802.11 and Bluetooth-based wireless transmission models. The underlying data traces are generated using the normal distribution $N(\mu,\sigma)$ (with appropriate truncation to avoid underflow below 0 or overflow above 100 %) on each of these sensors as follows: Sp02 as $N(96,4)$, HR as $N(80,40)$ and Accel as $N(0,10)$. μ,σ have been carefully selected on the basis of real world data. Every sensor data is one-hour (3600 sec) trace according to its own frequency.

7.1 Evaluation with a fixed time-shift value

Figure 9 plots the total data acquisition energy (in Joules, over the 1 hour evaluation duration) for each of the five algorithms, for the case of Bluetooth and 802.11-based PAN technologies respectively. These results correspond to a query with a time-shift value of $\omega = 10$ sec. We can see that our approach of sequential retrieval and evaluation of individual sensor streams, while taking into account their respective acquisition costs and selectivity characteristics, results in significant energy savings, compared to the naive approach where the data is pushed (albeit in batches) from each sensor. For 802.11 based transmissions, the ASRS algorithms result in ∼50 % to ∼80 % reduction in energy overheads compared to the Naive scheme.



(a) Q6-Total Energy (802.11) (b) Q6-Total Energy (Bluetooth) (c) Q6-Total Bytes of Sensor Data (802.11) (d) Q6-Total Bytes of Sensor Data (Bluetooth)

Fig. 9 Comparative energy (Joules) and data transfer (Bytes) for processing query Q6 with $\omega = 10$ sec. The left column shows the result for 802.11 energy model and the right column shows the results for Bluetooth energy models

For Bluetooth-based data transfers, the energy reductions are equally dramatic, with ASRS-static, ASRS-dynamic, ASRS-DNF and ASRS-MultiPred achieving around ∼60 %, ∼65 %, ∼70 % and ∼73 % reduction in energy overheads respectively. Figures 9(c) and 9(d) shows the number of bytes of sensor data transferred to the smartphone in order to process the query and the results validates the effectiveness of the ordering the predicates for shortcircuit query evaluation in reducing the amount of data acquired.

ASRS-dynamic outperforms ASRS-static by ∼30 % demonstrating that changing the data acquisition and query evaluation order dynamically at each time shift does translate to more energy savings. By taking the dynamic state of a query and the contents of the data buffer into account, the dynamic approach is able to further reduce the energy overhead, compared to the static counterpart. The gains are, however, not as dramatic for the Bluetooth interface (even though ASRS-dynamic has significantly lower variance than ASRS-static)—this is most likely due to the non-negligible T_{switch} overhead in Bluetooth, which implies that Bluetooth does not provide as great an advantage for very short-sized data transfers compared to larger batch sizes. The figures thus reveal that the relative performance of the algorithms depend significantly on the fine-grained features of the PAN radio technology, implying that the ACQUA algorithms need to be carefully tailored to the characteristics of the specific PAN technology adopted.

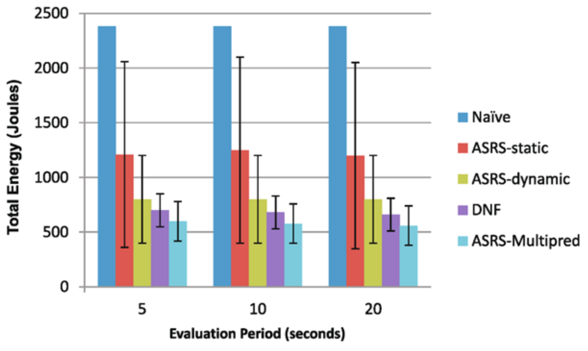
ASRS-DNF and ASRS-MultiPred when compared with Naive are especially impressive with energy savings between ∼75 % and ∼80 %. Moreover, their confidence intervals are much tighter compared to ASRS-dynamic and ASRS-static demonstrating the consistency of their good performance. That ASRS-DNF improves on ASRS-dynamic on the average energy consumed and more significantly on the confidence interval shows that the ASRS method is indeed able to find a better data acquisition order using the DNF query tree representation. We do note that the performance gains are query dependent. There are some queries whereby the larger permutation space afforded by the DNF representation does not result in finding a better data acquisition order and hence does not affect the energy cost significantly.

While the difference in the average energy consumed using the ASRS-dynamic and ASRS-DNF methods is small, the energy savings of ASRS-MultiPred are significant compared with ASRS-DNF. This result validates the effectiveness of using a bottom-up evaluation strategy together with a stream acquisition order heuristic as opposed to the predicate-based acquisition order used in ASRS-dynamic and ASRS-static. ASRS-MultiPred optimizes the data acquisition energy especially for queries containing multiple predicates on the same sensor stream. In general, we expect real world queries to have multiple predicates on the same sensor stream especially after converting them to DNF. Hence, the superior performance of ASRS-MultiPred compared to all the other methods is a very positive and encouraging result.

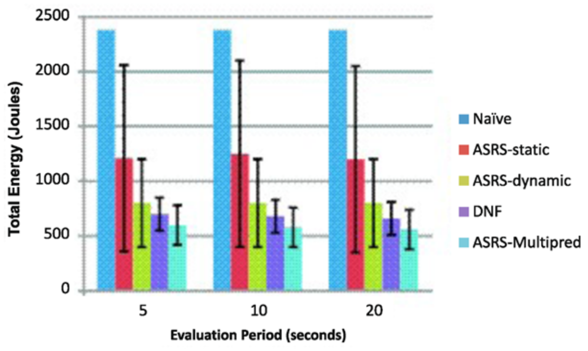
7.2 Evaluation under varying time-shift values

We also experimented on Q5 with different values of the 'time shift' window ω, i.e., by altering the frequency with which our 'tumbling window' query is evaluated. Figure 10(a) shows the energy overheads for the five algorithms for three different

Fig. 10 Comparative energy overheads, under varying ω values, for 802.11-based and Bluetooth data transfers



(a) Evaluation Period (seconds) for 802.11



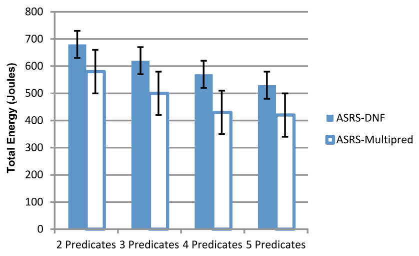
(b) Evaluation Period (seconds) for Bluetooth

values of $\omega = \{5 \text{ sec}, 10 \text{ sec}, 20 \text{ sec}\}$, for the case of IEEE 802.11-based sensor data transfers; $\omega = 3$ implies an overlap of time windows of successive evaluation instants. Figure 10(b) shows the energy consumed for the case of Bluetooth. It is interesting to observe that the relative gains are fairly independent of ω. In particular, when $\omega = 20$, there is no overlap between the evaluation window and the 'time shift' values; accordingly, the evaluation at a subsequent instant always starts with an empty buffer of data tuples. Nonetheless, the ASRS-dynamic algorithm is able to outperform the static variant, by better adapting its data acquisition sequence to take account of the intermediate query evaluations state (i.e., by eliminating data acquisition for those sub-trees that have already been 'short-circuited').

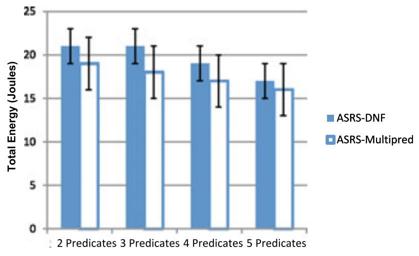
7.3 Evaluation under varying query characteristics

In this section, we investigate how different properties of query affect the performance of ASRS-DNF and ASRS-MultiPred. In our first experiment, we hand-crafted the following four queries each with a different number of predicates on stream A:

Q7: (((**AVG(A,5)**<98 %) AND (SPREAD(B,10)<2*g*))
AND (AVG(C,10)<75)) OR ((**AVG(A,10)**>92 %)
AND (SPREAD(D,10)>4*g*) AND (AVG(E,10)>100)).



(a) 802.11



(b) Bluetooth

Fig. 11 Energy overhead for four hand-crafted queries with different number of predicates on the same stream

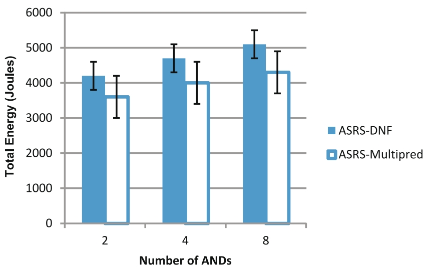
Q8: (((**AVG(A,5)**<98 %) AND (SPREAD(B,10)<2*g*))
AND (AVG(C,10)<75)) OR ((**AVG(A,10)**>92 %)
AND (SPREAD(D,10)>4*g*) AND (**AVG(A,10)**<95 %)).

Q9: (((**AVG(A,5)**<98 %) AND (**MIN(A,7)**>95 %))
AND (AVG(C,10)<75)) OR ((AVG(**AVG(A,10)**>92 %)
AND (SPREAD(D,10)>4*g*) AND (**AVG(A,10)**<95 %)).

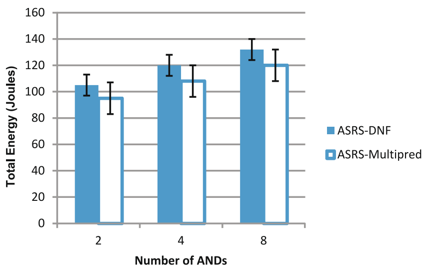
Q10: (((**AVG(A,5)**<98 %) AND (**MIN(A,7)**>95 %))
AND (AVG(C,10)<75)) OR ((AVG(**AVG(A,10)**>92 %)
AND (**MAX(A,3)**>99) AND (**AVG(A,10)**<95 %)).

Each of these queries contain exactly six predicates and we vary the number of predicates that are dependent on stream A. Figure 11 shows the energy consumed by ASRS-DNF and ASRS-MultiPred for the above four queries. Observe that as the number of predicates on stream A increases, the energy consumed by the two ASRS methods decreases. However, this is expected, because the total number of predicates in each query remains fixed, so increasing the number of predicates on stream A reduces the total sensor data required by the query as well. For 802.11-based transfers, we also observe that when there are two predicates on stream A, ASRS-MultiPred's energy overhead is ∼14 % less than ASRS-DNF and when there are four predicates on stream A, the difference increases to ∼24 %. However, the difference for Bluetooth is less dramatic. Note that since we keep the number of predicates in the query fixed at six, the more predicates that use the same stream, the less total input data is required by the query. Moreover, the window size of the predicates also affects the total input data required by the query. These complex relationships complicates our analysis of how different query characteristics affect the ASRS algorithms and one approach to explore this space is to evaluate the algorithms on randomly generated queries.

To further investigate the effects of how different query characteristics affect the performance of ASRS-DNF and ASRS-MultiPred, we run our experiments on randomly generated synthetic DNF query trees instead of hand-crafting queries. The synthetic queries are generated by varying (1) n_{and} the number of AND nodes, (2) n_{pred}



(a) 802.11



(b) Bluetooth

Fig. 12 Fix $n_{pred} = 4$, $n_{stream} = 4$, and vary n_{and}

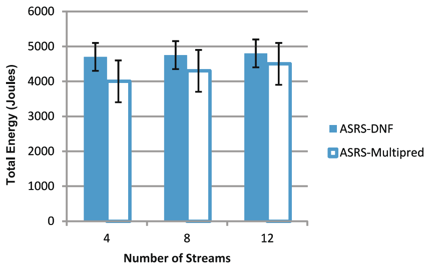
the number predicates under each AND node, and (3) n_{stream} the number of distinct sensor streams. We generate the queries randomly using the following rules for a given setting of $n_{and}, n_{pred}, n_{stream}$. Recall that a predicate consists of an aggregation function, a stream, a window, a comparison operator and a constant value (see definition of predicate in Sect. 4).

1. The root node must be an OR-node.
2. The second level of the query tree consists of exactly n_{and} AND-nodes.
3. Foreach AND-node generate exactly n_{pref} predicate nodes as follows,
4. Randomly pick an aggregation function,
5. Randomly pick one stream from the pool of n_{stream} streams,
6. Randomly pick a window size in a user-specified range,
7. Randomly pick a comparison operator among $\{>, <, =\}$,
8. Randomly pick a constant value.

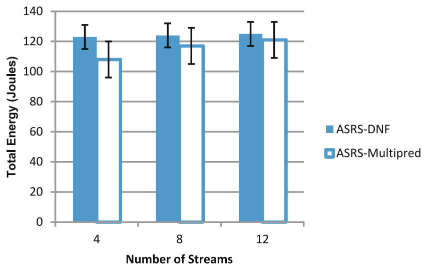
For the next three experiments, we fix two out of the three query properties $\{n_{and}, n_{pred}, n_{stream}\}$ and vary the other one. For real world applications, we expect n_{and} to be no more than 10, n_{pred} no more than six, and n_{stream} no more than 12. The experiments are run on 802.11-based and Bluetooth PAN technologies for one hour of simulation time and the time-shift value is still $\omega = 10$ sec.

Varying n_{and} In this experiment, we fix $n_{stream} = 4$, $n_{pred} = 4$, and vary n_{and}. Figure 12 shows the total data acquisition energy (in Joules). Observe that the growth in energy overhead is relatively stable compared to the number of AND-nodes which is increasing exponentially. Note that increasing the number of AND-nodes with a fixed number of predicates per AND-node results in a larger query tree (linear growth w.r.t. n_{and}). A larger query tree is likely to require more input data for evaluation. However, since the number of streams remain fixed, the number predicates that depend on the same streams also increases. For 802.11-based transfers, we do observe a widening gap between ASRS-DNF and ASRS-MultiPred, but for Bluetooth, the growth of the gap is much slower.

Varying n_{stream} In this experiment, we fix $n_{and} = 4$, $n_{pred} = 4$, and vary n_{stream}. Increasing the number of streams would mean that the number of predicates that depend on the same stream decreases—we expect that the performance gap between

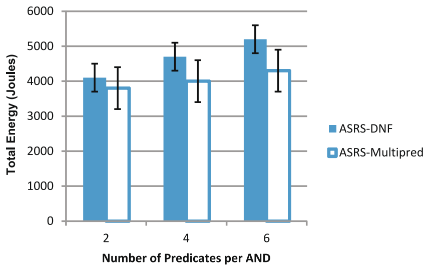


(a) 802.11

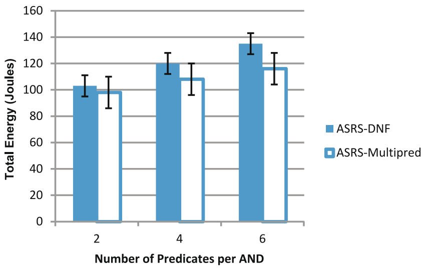


(b) Bluetooth

Fig. 13 Fix $n_{pred} = 4$, $n_{and} = 4$, and vary n_{stream}



(a) 802.11



(b) Bluetooth

Fig. 14 Fix $n_{stream} = 4$, $n_{and} = 4$, and vary n_{pred}

ASRS-DNF and ASRS-MultiPred would narrow. Figure 13 indeed shows that as the number of streams increases, the difference in energy overhead for ASRS-DNF and ASRS-MultiPred narrows, because ASRS-MultiPred is designed specifically to optimize for multiple predicates on the same streams. We also note that we rarely observe the case when ASRS-MultiPred is worse than ASRS-DNF in our experiments.

Varying n_{pred} In this experiment, we fix $n_{stream} = 4$, $n_{and} = 4$, and vary n_{pred}. Increasing the number of predicates while fixing the number of streams would increase the number of predicates that depend on the same stream. We would expect ASRS-MultiPred to perform better than ASRS-DNF. Figure 14 shows that the performance gap between ASRS-DNF and ASRS-MultiPred widens as the number of predicates increases as expected.

8 Conclusion and future work

In this paper, we have motivated the ACQUA framework for energy-efficient continuous evaluation of complex queries over sensor-generated data on a smartphone. The key to the ACQUA framework is the sequential retrieval of subsets of data tuples from each individual stream, with the preferred sequence being determined by considering both the query selectivity properties of the individual data stream and the sensor-specific energy overheads incurred by the sensor in transmitting the data over a PAN wireless network to the smartphone. We described four algorithms that consider in detail the transmission costs arising from batched transmission of sensor data tuples. While ASRS-static determines an optimal retrieval sequence once when the query is submitted for execution, ASRS-dynamic re-evaluates the optimal retrieval sequence at each evaluation instant, taking into consideration the state of both the stream buffers and the partially evaluated query. ASRS-DNF uses an alternate disjunctive normal form query tree representation that permits a richer set of retrieval sequences. ASRS-MultiPred further exploits bottom-up query evaluation to optimized for multiple predicates operating on the same stream. Our results on synthetic traces indicate that the ACQUA approach can result in $\sim$80 % reduction in the energy overheads of continuous query processing. Note that in all cases, there is *no degradation in the fidelity of the processing logic*, i.e., the semantics of the queries are completely preserved.

We conclude by emphasizing that the overall ACQUA effort has several open questions that need to be explored further. The algorithms presented in this paper *assume* the availability of the selectivity statistics for each sensor stream. Our ongoing work encompasses two orthogonal threads.

– **Systems Research:** We are implementing ACQUA on an Android-based smartphone platform, with special focus on online-learning algorithms to estimate the selectivity statistics from the history of sensor-generated data. Subsequently, user studies with real-life sensor traces will be used to quantify the performance gains of the ACQUA framework using real-life, instead of currently-used synthetically generated, sensor traces. We are also working to extend ACQUA's processing logic to the case of distributed query processing over multiple proximate smartphones [14], where multiple stream processing engines (one on each smartphone) execute over a shared repository of sensors, offering additional opportunities for query optimization.
– **Algorithmic Research:** We are working to extend the ASRS algorithms to include additional query semantics, such as the support of *sliding window* queries and on techniques to further improve the dynamic computation of the cost functions, given the statistical characteristics of the data tuples already available in the system buffer. (For example, while the probability of $\mathrm{AVG}(S5, 10) > 40$ may be generically 0.8, the probability at a specific instant should be different if 8 out of 10 samples in that evaluation window are already buffered and are all observed to be less than 10.)

References

1. Gaonkar, S., Li, J., Roy Choudhury, R., Cox, L., Schmidt, A.: Micro-Blog: sharing and querying content through mobile phones and social participation. In: Proceedings of ACM Mobisys'08, June 2008
2. Miluzzo, E.: Sensing meets mobile social networks: the design, implementation and evaluation of the CenceMe application. In: Proceedings of ACM Conference on Embedded Networked Sensor Systems (SenSys '08), November 2008
3. Mohomed, I., Misra, A., Ebling, M., Jerome, W.: Context-aware and personalized event filtering for low-overhead continuous remote health monitoring. In: IEEE WoWMoM, June 2008

4. The SHIMMER sensor platform (2012). http://shimmer-research.com
5. Priyantha, B., Lymberopoulos, D., Liu, J.: Enabling energy efficient continuous sensing on mobile phones with LittleRock. In: Proceedings of IPSN, April 2010
6. Lu, H., Yang, J., Lu, Z., Lane, N., Choudhury, T., Campbell, A.: The Jigsaw continuous sensing engine for mobile phone applications. In: Proceedings of ACM Conference on Embedded Networked Sensor Systems (SenSys '10), November 2010
7. Kang, S., et al.: Orchestrator: an active resource orchestration framework for mobile context monitoring in Sensor-rich mobile environments. In: Proceedings of the 8th IEEE International Conference on Pervasive Computing and Communications (PerCom), March 2010
8. Liu, J., Zhong, L.: Micro power management of active 802.11 interfaces. In: Proceedings of ACM Mobisys'08, June 2008
9. Roychoudhury, A., Falchuk, B., Misra, A.: MediAlly: a provenance-aware remote health monitoring middleware. In: 8th IEEE International Conference on Pervasive Computing and Communications (PerCom), March 2010
10. Dogar, F., Steenkiste, P., Papagiannaki, D.: Catnap: exploiting high bandwidth wireless interfaces to save energy for mobile devices. In: Proceedings of ACM Mobisys'10, June 2010
11. Jang, K., Lee, T., Kang, H., Park, J.: Efficient power management policy in Bluetooth. IEICE Trans. Commun. (2001)
12. Deshpande, A., Guestrin, C., Madden, S., Hellerstein, J., Hong, W.: Model driven data acquisition in sensor networks. In: Proceedings of VLDB, pp. 144–155. Morgan Kaufmann, San Mateo (2004)
13. Vallina-Rodriguez, N., Crowcroft, J.: ErdOS: achieving energy savings in mobile OS. In: ACM MobiArch, June, 2011
14. Rachuri, K., Mascolo, C., Musolesi, M., Rentfrow, P.: SociableSense: exploring the trade-offs of adaptive sampling and computation offloading for social sensing. In: ACM Mobicom, September 2011
15. Corson, M., Laroia, R., Li, J., Park, V., Richardson, T., Tsirtsis, G.: Toward proximity-aware internetworking. IEEE Wireless Commun. (2010)
16. Hellerstein, J.M., Stonebraker, M.: Predicate migration: optimizing queries with expensive predicates. In: SIGMOD International Conference on Management of Data (1993)
17. Kemper, A., Moerkotte, G., Peithner, K., Steinbrunn, M.: Optimizing disjunctive queries with expensive predicates. In: Snodgrass, R.T., Winslett, M. (eds.) Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data (SIGMOD '94), pp. 336–347. ACM Press, New York (1994). doi:10.1145/191839.191906