QUASAR: Quality Aware Sensing Architecture

Iosif Lazaridis, Qi Han, Xingbo Yu, Sharad Mehrotra,
Nalini Venkatasubramanian, Dmitri V. Kalashnikov, Weiwen Yang
Information and Computer Science
University of California, Irvine
{iosif, qhan, xyu, sharad, nalini, dvk, wyang}@ics.uci.edu

Abstract

Sensor devices are promising to revolutionize our interaction with the physical world by allowing continuous monitoring and reaction to natural and artificial processes at an unprecedented level of spatial and temporal resolution. As sensors become smaller, cheaper and more configurable, systems incorporating large numbers of them become feasible. Besides the technological aspects of sensor design, a critical factor enabling future sensor-driven applications will be the availability of an integrated infrastructure taking care of the onus of data management. Ideally, accessing sensor data should be no difficult or inconvenient than using simple SQL.

In this paper we investigate some of the issues that such an infrastructure must address. Unlike conventional distributed database systems, a sensor data architecture must handle extremely high data generation rates from a large number of small autonomous components. And, unlike the emerging paradigm of data streams, it is infeasible to think that all this data can be streamed into the query processing site, due to severe bandwidth and energy constraints of battery-operated wireless sensors. Thus, sensing data architectures must become *quality-aware*, regulating the quality of data at all levels of the distributed system, and supporting user applications' quality requirements in the most efficient manner possible.

1 Introduction

With the advances in computational, communication, and sensing capabilities, large scale sensor-based distributed environments are becoming a reality. These will allow us to continuously monitor and record the state of the physical world which can be used for a variety of purposes. It can be used to gain a better understanding of the physical world - e.g., data from embedded loop sensors on highways can be analyzed to understand the emergent traffic patterns. It can also be used to dynamically optimize the process that drives the physical world – e.g., real-time traffic conditions can be used to control the traffic flow. Sensor enriched communication and information infrastructures have the potential to revolutionize almost every aspect of human life benefiting application domains such as transportation, medicine, surveillance, security, defense, science and engineering. An integral component of such an infrastructure is a data management system that allows seamless access to data dispersed across a hierarchy of storage, communication, and processing units – from sensor devices where data originates to large databases where the data generated is stored and/or analyzed.

Designing a scalable data management solution to drive distributed sensor applications poses many significant challenges. Given the limited computational, communication, and storage resources at the sensors, a traditional distributed database approach in which sensors function as nodes in a distributed system might not be a feasible option. In order to facilitate complex query processing and analysis, data might need to be migrated to repositories that reside at (more powerful) servers. An alternative solution, where sensor data is continuously collected at a (logically) centralized database, as in the emerging *data streams* model [2] might also be infeasible due to the dynamic nature of sensor environments. More specifically, sensor readings may change very frequently/continuously; blindly transmitting sensor updates to the server constitutes a major source of power drain [1] in battery-operated sensors. In addition to the high energy cost, the frequent communication imposes severe network/storage overheads.

The problem of effective data collection in highly dynamic environments has recently been studied in [8, 13, 5, 4]. The key observation is that a large number of sensor applications can tolerate a certain degree of error in data. Consequently, the commu-

nication overhead between the data producers (sensors) and the server can be alleviated by exploiting the applications' error tolerance. Data imprecision, of course, impacts application quality. For example, in an application such as target tracking in a sensor network, error in sensor intensity readings may result in error in localizing the target. Similarly, the result of a query for average temperature in a given region may be imprecise due to data error.

In our work, we use the notion of application error tolerance to enhance overall system performance while guaranteeing desired levels of application quality in sensor environments. This paper discusses some of the challenges and open problems that must be addressed to enable quality-aware sensor data architectures. It summarizes and expands on our work in the QUASAR (Quality-Aware Sensing Architecture) project at UC Irvine [11] whose long-term goal is to integrate various ideas at the sensor, middleware and application levels into a unified system which will be easily customized to individual sensing applications, but will be generic and modular enough to be useful to a large class of such applications.

2 Application Taxonomy

The first step towards building a general purpose sensing architecture is to differentiate between different application types that will use this architecture to access sensor-generated data. Often, sensor-based systems are built with narrow application goals in mind. Consider, e.g., a simple application using chemical sensors to track pollutants in water streams and reporting these periodically. We anticipate that as sensor network infrastructures become more sophisticated, they will have to accommodate several concurrent applications, some of whose requirements may conflict in terms of timeliness, reliability and data accuracy. It is important for future sensing architectures (a) to accommodate alternative application types, and (b) to ensure that their conflicting requirements mesh with each other gracefully.

To assist in identifying the needs of future sensor networks, we have developed a taxonomy of potential application classes. The first division relates to the temporal aspect of sensor data. In particular, a generic taxonomy must subsume applications that focus on historical (past) data, e.g., in order to detect patterns over time and build time-varying models. Real-time data collection is not critical here, but high-quality and reliable archival of sensor-generated data is. Other applications are interested in current sensor values. This is especially true for monitoring applications (e.g., intrusion detection systems) or those that use sensor inputs to actuate control interventions. Finally, there are applications that are interested in forecasting future sensor values, where human operators involved in decision making processes can avail of information about trends in sensor values. An example of this is route selection in intelligent transportation applications, where techniques to predict traffic conditions (estimated from traffic monitoring devices) among various route candidates can help in reducing travel times and latencies.

A second division of applications relates to the pattern of access to the sensor data. In some cases, e.g. pollutant tracking, the target application is known *before* sensor data is collected. Thus, one can set up the data collection framework in a manner that is optimal for the particular application. In particular, if multiple such applications co-exist, as in e.g., multiple continuous queries [7] then one could exploit the overlap in applications' data needs to optimize the amount of data communication. In other cases, the application type (e.g., requests for aggregate pollutant levels over spatial grids) is known, but not the query instances, which are unknown and arrive in an *ad-hoc* manner. Finally, there is the case where the specific application type may not be known beforehand. For example, one might instrument a network of traffic monitoring sensors for the purpose of monitoring speed levels in different segments of the road network. However, sensors in such an infrastructure may potentially be used in the future for very different applications: e.g., to control traffic signals, or disseminate optimal routes in real-time to drivers.

3 Architectural Issues

In this section we first examine the capabilities and limitations of modern sensors. Then we show how imprecise replication is essential for sensor-based systems to scale gracefully. This involves maintaining information in multiple locations, at different levels of quality especially using data compression and prediction techniques, but also in-network processing of both queries and data.

3.1 Modeling Sensors

At the most basic level, a sensor is nothing more than a device which monitors a physical process (e.g., the fluctuations of temperature), converts the sampled value into an appropriate digital format and forwards it to wherever this information can be utilized. Modern sensors, however, are much more complex.

Most importantly, the addition of a programmable CPU and memory on-board sensors has meant that unlike earlier "dumb" devices, these can now store and process data, both autonomously, and in concert with their peers and the data infrastructure at large. For example, as we will see, sensors can pool their resources to achieve application goals, such as tracking mobile objects, or apply intelligent algorithms to the sampled data to minimize transmission costs and improve system performance.

The main problem that sensor-based data architectures face is that sensors have finite energy reserves and these are expended during normal operation. Levels of energy drain are not uniform during various sensor operating modes. Table 1 shows the representative power consumption of a typical sensor (Berkeley mote) at different radio modes, where the transmission range is set to approximately 20m and provides a transmission rate of 19.2 Kbps [9].

radio mode	power consumption(mW)
Tx	14.88
Rx	12.50
Idle	12.36
Off	0.016

Table 1: Sensor power consumption

Transmission (Tx) and reception (Rx), or idle listening (Idle) cannot happen simultaneously on a single radio, e.g., as in the one utilized by Mica Mote [9]. Other proposed sensors, like the μAMPS node [10] have dual radios, hence they can do both at the same time. We can accommodate both sensor types in our architecture. Dual radios ensure that neither incoming or outgoing messages are delayed due to competition for the radio channel, but this happens at the cost of increased sensor complexity and roughly double energy drain when both radios are turned on. Sensors of both kinds also have the option of powering down their radio(s) completely, going to the Off mode. In the Off mode, sensors continue monitoring their environment regularly, but don't communicate: this delays incoming messages to the sensor, but outgoing messages are not necessarily affected, since the sensor can decide to enter Tx mode whenever it has data to transmit.

3.2 Imprecise Replication

It is anticipated that sensor networks will be widely distributed geographically and based on unreliable wireless links. Hence, applications requesting data must incur a high cost in terms of response time if they have to initiate direct communication with the sensors. This is intractable when (i) multiple data items need to be retrieved, and derivative information be generated by combining sensor values, (ii) multiple queries seek to access the same sensors simultaneously. Moreover, while sensors can carry on some computation, they might not be able to accommodate the load associated with expensive data-centric operations, like the evaluation of complex operators (e.g., joins), data mining tasks (clustering), or other expensive analytical operations that are best achieved with powerful machines.

Thus, the "no-server" model, whereby sensors achieve application objectives mostly on their own, without a hierarchical superstructure involving more powerful machines, is not feasible.[1]

For this reason, we stress the importance of *imprecise data replication* which allows data to be cached at different locations of the system, including powerful machines, at various degrees of quality. Note that imprecision is itself a feature of the sensing process itself, since sensing itself introduces uncertainty: it captures the underlying process only inasmuch as sensing technology allows. Additional imprecision is introduced due to the loss or corruption of information during transmission. Imprecision can also be a choice though, as it has the benefit of keeping system performance in acceptable levels: by intentionally imprecise replication, communication and its resultant ills (bandwidth/energy consumption) are addressed.

3.3 Compression and Prediction

In previous work [5] we showed how time series generated by sensors can be reflected at a remote query processing site in approximate form. We noted the importance of the problem of *data archival* where we want to capture a version of a time series of prespecified quality for future reference. To minimize network load, the self-correlation of time series values can be exploited using a compression algorithm. An optimal instance optimal online algorithm is proposed which generates the least number of segments under a bound on the L_∞ metric.

Unfortunately, implementing such a policy blindly may minimize network data transfer, but works against applications that require prompt data migration [5] from the sensors to the central site when this

[1] Exploiting such a model is still important though, since it might be necessary in situations where such a superstructure could not be deployed, e.g., in enemy territory, or could not be depended upon, e.g., the system must still be functional even if all centralized components go off-line during a terrorist attack or natural disaster.

acts not only as a data archive but as a query processing center. To ensure that such applications are not penalized, sensors can additionally model the behavior of the underlying time series and transmit predictive models to the query processing site. For example, one might detect an increasing trend in ambient temperature and report a model consisting of the temperature's rate of increase. This will allow queries to estimate temperature without direct communication with the sensor and the associated introduced latency. The sensor will have to promise to update its predictive model whenever the underlying process deviates from the prediction. At the other end, queries can decide whether they are willing to tolerate the provided quality guarantees, or whether they need to communicate ("probe") the sensor(s) directly.

This framework both utilizes sensors' processing capacities in that they now perform useful computation for compression and model building that can be used by queries running at the query processing site. This, however does not occur at the expense of the sensors themselves who rather benefit from it as this effort is used to help reduce the volume of transmitted data, and thus, as we have noted previously prolong their battery life.

3.4 In-Network Processing

In the previous section we hinted at how sensors can compress and model the data that they generate so as to preserve their energy resources while at the same time avoiding flooding the system with data which it cannot realistically sustain at the rates in which it is generated. Sometimes, the network, which in this case might consist of other sensors acting as intermediaries or even wired components does not need to be passive. Rather, it can act both to achieve application goals, and to enhance performance.

E.g.,, consider the task of tracking a moving object using acoustic sensors [13]. Sensors cannot individually track objects, since sound intensity at a given location is insufficient information for localizing it. However, a number of sensors working together in concert could achieve this goal. Thus, the application objective can be achieved in-network via sensor-to-sensor interactions. Secondly we have considered the problem of in-network evaluation of continuous aggregation queries [12], where we proposed algorithms that (i) combine sensors into an approximate aggregation tree (AAT), with the goal of maintaining aggregate information at various levels of quality so that (ii) queries can be answered without communicating with all individual sensors but rather with those nodes of the AAT which suffice for their requested quality tolerance, finally (iii) quality levels are adaptively adjusted as query/data patterns change.

3.5 Fault Tolerance and Timeliness

Imprecise replication presents an important semantic problem. We have said that sensors, sitting at the bottom of the data replication hierarchy are responsible for ensuring the validity of the cached replicas of information across the system. Sensors, however, break down, starve for energy, or lose network connectivity. Messages get lost or corrupted in transit. Hence, applications must be able to distinguish whether the cached replicas have not been refreshed because they remain valid representations of the real world, or whether because a fault has occurred.

Fortunately, this fits rather well with the predictive framework used to generate cached replicas. The quality of these decays at rates which can be estimated either by the sensors themselves during model generation or by the central site, e.g., by noting the frequency with which these tend to become invalid. Subsequently, probabilistic estimates of their expected time until invalidation can be achieved. If these are surpassed then a fault may have occurred which can then be treated in a number of different ways, e.g., by tagging suspect values as such, or by initiating communication with sensors to determine whether a fault has indeed occurred, or whether the delay in communication is normal.

Additionally, even when sensors are always functional and messages don't get lost, there is no *a priori* guarantee that messages will be delivered in a timely manner. Without such a guarantee, the absence of a message represents the same semantic quagmire as before. To address this, a real-time, deadline-driven scheduling algorithm was proposed [4] which ensures timely data delivery.

4 Complex Queries and Quality Specification

Previously, we showed how data collection protocols ensure the propagation of sensor-generated data efficiently via the network to centralized locations. Queries that run at these locations must (a) specify their answer quality requirements, which allow them to be answered without communicating with all sensor units that they reference, (b) be evaluated efficiently without violating these requirements. Here are three examples of the kinds of queries that we

might want to ask in a hypothetical database hosting sensor-generated information.

- Q1: "Which sensors in spatial grid R have temperature values above critical threshold x?"
- Q2: "What is the average temperature reported by sensors within $100m$ of sensors reporting values above x?"
- Q3: "What are the pressure values of sensors that are in grid R whose temperature values are above x?"

Q1 is a normal selection, both on non-sensor (spatial location) and sensor (temperature) data. It could be used to detect e.g., fires in a heavily forested area soon after they get started, allowing for a rapid response. Q2 is an aggregation over sensor-generated information, with a selection condition also on sensor-generated information. Imagine that it could be used to detect false alarms: if a sensor reports a very high temperature value but it's neighbors don't, then this might indicate a broken sensor and not a fire. Q3 asks for sensor-generated data (pressure) based on a selection that involves both non-sensor and sensor-generated (temperature) information.

It is easy to specify what answer quality means for aggregation queries such as Q2. If we knew the precise sensor values, we would be able to answer Q2 precisely, e.g., 34^oC. The query writer can simply indicate that e.g., he wants the given answer to be within $\pm 1^oC$ of the exact answer. Thus, e.g., approximate answer 34.3^oC would satisfy this requirement.

Things get more difficult as queries become more complex, especially if their answers are in the form of sets. Judging the quality of such results is more difficult. E.g., for Q3, uncertainty is introduced both with respect to the selection condition (e.g., "is sensor XYZ *really* above x?") and with respect to the values that appear in the output (e.g., pressure values that are returned are themselves imprecise). We thus break up answer quality into two components [6]:

- *Set-based Quality.*— This involves our uncertainty in determining whether a tuple belongs in the output or not
- *Value-based Quality.*— This involves our uncertainty in determining the precise values of tuples that are returned

Queries can now easily specify their quality requirements. Such queries are termed *Quality-Aware Queries*, or QaQs. For example, we might re-write Q3 as a QaQ as follows: "Report the pressure values (within $\pm 5mbar$) of at least 50% of the sensors that are in grid R and whose temperature value is above x, ensuring that at least 95% of the returned results meet this condition." Such a query could be answered satisfactorily if: (i) all returned tuples were guaranteed to have their pressure attribute within $\pm 5mbar$, (ii) if in reality there were e.g., 1,200 such sensors, then at least 600 of these should be returned, (iii) if e.g., 1,000 are returned in total, then 950 of these should in fact meet the selection condition.

4.1 Joint Data Access and Probing Optimization

As we have seen, queries will now be posed as QaQs, with quality requirements, and the system must work towards meeting these requirements in the most efficient manner possible. In other words, it should return an approximate result that matches the quality specification with minimal cost. This differs from traditional database management where efficiency is again the issue, but there is no ambiguity in the result that ought to be returned to the user.

A second difference is that of *probing*, which we define informally as requesting higher-precision versions of imprecise objects stored in the database. Probing entails network communication, which as we have stressed continuously is expensive in the case of sensors. Hence, one is tempted to try to minimize the cost of probing, which usually boils down to issuing the minimum number of probes in order to achieve a certain level of answer quality.

But, such an approach is narrow-sighted, ignoring the fact that probing is not the *only* cost that gets paid during query evaluation. Traditional data access operations (table and index scans, evaluation of operators) are also important. It is rather the *combined cost* of data access[2] and probing that needs to be optimized. Ignoring this could have catastrophic results. Here is an example of an extreme case: imagine that your query is a join of two tables, $R \bowtie_C S$. If the join condition C is on some imprecisely represented attributes, then in the worst case it might be that $|R \times C|$ might be "candidate" answers for this query. For example, if $|R| = |S| = 1000$ then as many as 10^6 answers might need to be tested. This is the strategy that we would be tempted to follow if we wanted to minimize probing, since it would allow us to order candidate answers and probe those that have

[2]The issue of data access itself is not straightforward, since it might be possible to approximate query results during the data access process itself, either by sampling or by using a specialized data structure, e.g., MRA-Tree [**?**].

a high chance of being valid answers. But, think of the alternative strategy of probing all tuples in the two relations, i.e., 2,000 probes. This would result in outputting only the pairs which do indeed match according to condition C. These may be quite fewer, e.g., $0.5 \cdot 10^5$. In other words, the second plan would save the cost of outputting and testing $500,000$ tuples for a cost of $2,000$ probes. It is not entirely clear which of the two plans would be more efficient, or if indeed a mixture of the two strategies would be more efficient than both.

5 Current Work

Currently, we are working to further explore certain aspects of our architecture. We list some of the more important directions here: (i) to further investigate sensor operation modalities, especially in the context of a real testbed as opposed to simulation,[3] (ii) to further investigate the issues involved in in-network processing, both in order to achieve application goals (as in tracking), but also as a means of performance enhancement (as in devising smart ways to shift data around) that achieves the same distribution of data quality across the system but with minimal cost, (iii) to add probabilistic quality guarantees (as in e.g., [3]) to our architecture in addition to our present deterministic ones, (iv) to generalize our work on QaQ optimization in the presence of multiple access methods, and for a general class of queries, eventually of SQL-level of complexity.

6 Conclusions

We have identified some key concerns that must be addressed in next-generation sensor-based data architectures. These revolve around four main axes: (i) the need to utilize sensors' increased capabilities, (ii) to accommodate sensors' energy and bandwidth limitations, (iii) to provide to applications a way of interacting with sensor-generated data that is as simple as SQL, and (iv) to handle the flow of data of varying quality in a way that allows application requirements to be achieved efficiently.

[3] Experimentation with real sensors is rewarding, however it is not always the optimal course when trying to study issues of scale since very large sensors deployment projects (in the order of millions of devices) are still not frequent and are beyond the means of an academic project.

References

[1] I. Akyildiz, W. Su, Y. Sankarasubramanian, and E. Cayirci. Wireless sensor networks: A survey. *Computer Networks*, 38(4), March 2002.

[2] B. Babcock, S. Babu, M. Datar, R. M. i, and J. Widom. Models and issues in data stream systems. In *Symposium on Principles of Database Systems (PODS)*, 2002.

[3] R. Cheng, D. V. Kalashnikov, and S. Prabhakar. Evaluating probabilistic queries over imprecise data. In *ACM SIGMOD Conference*, 2003.

[4] Q. Han and N. Venkatasubramanian. Addressing timeliness/accuracy/cost tradeoffs in information collection for dynamic environments. In *IEEE RTSS*, 2003.

[5] I. Lazaridis and S. Mehrotra. Capturing sensor generated time series with quality guarantees. In *IEEE ICDE Conference*, 2003.

[6] I. Lazaridis and S. Mehrotra. Approximate selection queries over imprecise data. In *IEEE ICDE Conference*, 2004.

[7] C. Olston, J. Jiang, and J. Widom. Adaptive filters for continuous queries over distributed data streams. In *ACM SIGMOD Conference*, 2003.

[8] C. Olston, B. T. Loo, and J. Widom. Adaptive precision setting for cached approximate values. In *ACM SIGMOD Conference*, 2001.

[9] RF Monolithics Inc., http://www.rfm.com/. *ASH Transceiver TR1000 Data Sheet.*

[10] A. Sinha and A. Chandrakasan. Dynamic power management in wireless sensor networks. *IEEE Design and Test of Computers*, 18(2), March/April 2001.

[11] UC Irvine, http://www-db.ics.uci.edu/pages/research/quasar/index.shtml. *QUASAR Project.*

[12] X. Yu, S. Mehrotra, N. Venkatasubramanian, and W. Yang. Approximate monitoring by aggregation-oriented clustering inwireless sensor networks. (under submission), 2003.

[13] X. Yu, K. Niyogi, S. Mehrotra, and N. Venkatasubramanian. Adaptive middleware for distributed sensor environments. *IEEE DS Online*, 4(5), May 2003.