1st IEEE International Workshop on Networking Meets Databases (NetDB), in cooperation with
21st IEEE Conference on Data Engineering (ICDE 2005)

Data Acquisition in Sensor Networks with Large Memories

D. Zeinalipour-Yazti, S. Neema, V. Kalogeraki, D. Gunopulos, W. Najjar
University of California - Riverside
{csyiazti,sneema,vana,dg,najjar}@cs.ucr.edu

Abstract

Sensor Networks will soon become ubiquitous, making them essential tools for monitoring the activity and evolution of our surrounding environment. However such environments are expected to generate vast amounts of temporal data that needs to be processed in a power-effective manner. To this date sensor nodes feature small amounts of memory which mostly limits their capabilities to queries that only refer to the current point in time. In this paper we initiate a study on the deployment of large memories at sensor nodes. Such an approach gives birth to an array of new temporal *and* top-k *queries which have been extensively studied by the database community. Our discussion is in the context of the* RISE (RIverside SEnsor) *hardware platform, in which sensor nodes feature external flash card memories that provides them several Megabytes of storage.*

1. Introduction

The improvements in hardware design along with the wide availability of economically viable embedded sensor systems will soon drive us into the ubiquitous silicon era. Sensor networks will become essential tools for monitoring the activity and evolution of our surrounding environment. Applications have already emerged in environmental monitoring, seismic and structural monitoring, factory and process automation monitoring and a large array of other applications [9, 5, 6, 8].

Conventional approaches to monitoring have focused on dense deployed networks that either transfer the data to a central sink node or perform in-network computation and generate alerts when certain events arise. An important attribute of these applications is that the time interval between consecutive query re-evaluations (*epoch*) is small because we want to achieve quick reaction to various alerts. For example, a query might continuously manipulate the temperature at some region in order to identify fires or other extreme situations (e.g. *"Find which sensors record a temperature>120F?"*). Therefore the querying node (*sink*) must continuously maintain an updated view of the values recorded at the sensors [8, 5]. In such *short-epoch* applications, the frequency of updates and the timely delivery of information from the sensors play a vital role in the overall success of the system. On the other hand, a class of applications that were not addressed to this date are *long-epoch* applications. In these applications the user needs an answer to his query more sparsely (e.g. weekly or monthly) although the sensor node has to read values from the surrounding environment frequently (e.g. every second). The user might then ask: *"At the day and time on which we have the highest average temperature in the last month?"*. In order to evaluate such a query using current techniques, would require each sensor to report all its values for the last month.[1] This happens because the data is fragmented across the different nodes and an answer to the query can only be obtained after accessing all distributed relations in their entirety. We call this type of *in-situ* data fragmentation *vertical partition*, because each sensor's timeseries is one dimension. This makes it a challenging task to answer user queries efficiently.

Our Contribution: In this paper we initiate a study on the deployment and use of sensor networks characterized by large memories at sensor nodes. This will allow each sensor node to accumulate measurements over a large window of time enabling the network to efficiently answer temporal queries. We also address the issue of answering top-k queries, where the system only returns the k highest ranked answers. An example of a top-k query might be *"Find the three moments on which we had the highest average temperature in the last month?"*.

Temporal and top-k queries are useful in a number of contexts. Our work is motivated by the requirements of the Bio-Complexity and the James Reserve Projects at the Center of Conservation Biology (CCB) at UC Riverside.[2] CCB is working towards the conservation and restoration of species and ecosystems by collecting, evaluating scientific information (Figure 1). The bio-complexity project is designed to develop the kinds of instruments that can monitor the soil environment directly, rather than in laboratory recreations.

We are currently developing the *RISE platform*, in which

[1] Alternatively the sink could continuously gather the data in a centralized fashion, but this would become prohibitively expensive.

[2] http://www.ccb.ucr.edu/

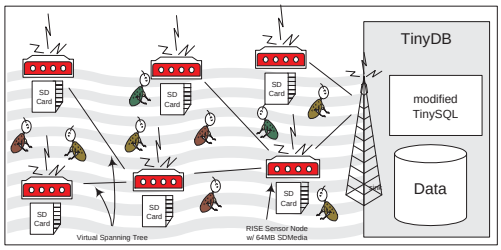


Figure 1. Soil-Organism Monitoring Application: Each sensor stores locally on external flash memory the CO_2 levels over a large window of time.

sensors feature a large external memory (SD flash memory). RISE sensors are able to store measurements of Carbon-dioxide levels in the soil as well as ambient sound from the surrounding environment over a large period of time. This will allow scientists to monitor the long-term behavior of certain soil micro-organisms and bird species.

We address the efficient evaluation of top-k queries in our platform by sketching an algorithm that estimates some threshold below which tuples do not need to be fetched from the sensor nodes. Key ideas of our algorithm are to transmit only the necessary information towards the querying node and to perform calculation in the network rather than in a centralized way.

2 The RISE Platform

The *RISE (RIverside SEnsor)* platform employs a System-on-Chip interfaced with a large external storage memory in the form of off-the-shelf SD (Secure Digital) card to develop a new paradigm of "sense and store" as opposed to the prevalent "sense and send". The RISE platform was conceived by observing the twin trends of falling flash memory prices[3] and the need of larger memories on sensory devices for more efficient querying, processing and communication. Also, higher levels of device integration at low cost and size now provide us with single chip solutions for most of the sensory and communication needs, reducing complexity and improving performance. The RISE wireless platform is built around the Chipcon CC1010 System on Chip (SoC), which together with just a few external passive components and the required sensors constitutes a powerful, robust and versatile wireless embedded sensor system. The following is a description of the important components of the RISE platform (summarized in table 1):

1. The MicroController Unit (MCU): The Chipcon CC1010 SoC is a true single-chip RF transceiver with an integrated high performance 8051 microcontroller and high end features which include a 32KB flash memory, an SPI (Serial Peripheral Interface) bus, DES encryption, 26 general I/O pins and many other components constituting it appropriate for a multitude of sensory and computation needs.

[3] According to research firm IDC, flash memory card sales will increase from about 100 million units this year to 315 million annually in 2007 together with a sustained fall in price.

Characteristic	Capability
MCU	
Processor	24 MHz 8051 core
On-Chip Flash Memory	32 KB
Current (On,Idle,Off) at 14 MHz	14.8 mA, 8.2 mA, 0.2 μA
Radio (RF Transceiver)	
Communication Rate	76.8 kbits/s
Communication Range	250m at 868/915 MHz
Current (Receive,Send at 10dBm)	11.9 mA, 26.6 mA
SPI bus (Interfacing MCU with SD Card)	
Data rate	Up to 3 Mbps
Data block length	512 bytes
SD Card	
Current (Read, Write, Sleep)	60mA, 80 mA, 500μA
Access Time	200μsec (max)
Read/Write (256-512MB card)	10MB/sec

Table 1. Characteristics of the RISE platform.

2. The SD-Card interface: An SD-Card (Secure Digital Card) has been interfaced to the main chip using the SPI bus equipping the RISE platform with a large external storage memory (up to a 1 GB!). Data can be buffered on the 32KB flash memory for reading and writing efficiently on the SD-Card. Data is transferred to the SDCard in blocks of 512 bytes at a maximum rate of 3 Mbps. We are currently working on developing tiny access method structures which will allow efficient sorted and random access to local data.

3. The OS & Compilation: To facilitate ease and modularity of programming, we have ported the most prevalent design environment, the TinyOS (version 1.1) and NesC (version 1.2alpha1), facilitating easier and modular programming, interfacing of an SD-CARD and developing the reactive methodology of query based response on large datasets stored locally on the nodes.

4. Deployed Sensors: The platform has a temperature sensor and is also being interfaced with a CO_2 sensor and a microphone.

Note that the energy cost of writing to flash memory is much cheaper than the RF transmission cost even in the case of a single hop. Given the characteristics outlined on Table 1 along with a 3V voltage, it is easy to derive that a transmitting one byte over the RF radio requires 8.379 mJ while storing the same byte on the flash card requires 0.480 mJ. Although accessing the external flash card can only be performed in blocks of 512 bytes, the 32KB on-chip flash memory allows us to buffer a page before it is written out. This in combination of the fact that the energy required for the transmission of one byte is roughly equivalent to executing 1120 CPU instructions, makes local storage and processing highly desirable.

3 The Query Processing Framework

In this section we expand on the class of queries we will consider in the RISE platform. This class represents queries that are interesting and important in our framework that is characterized by long-epochs and large storage capacities in individual sensors. We also describe and contrast alternative frameworks that have been proposed for data acquisition in sensor networks.

3.1 Temporal and top-k queries in RISE

We assume that a query dissemination mechanism similar to the one described in [6, 5] creates a "virtual" *Query Spanning Tree (QST)* interconnecting all nodes in the network. This provides each node with the next hop towards the sink (See Figure 2.) Alternatively each node could maintain multiple parents in order to achieve fault tolerance [1].

Let $G(V, E)$ denote an undirected and connected network graph that interconnects n sensors in V using the edge set E. The edges in E, represent the virtual connections (i.e. nodes are within communication radius) between the sensors in V. Also assume that each sensor has enough storage to record a window of m measurements. Each measurement has the form (ts, val), where ts denotes the timestamp on which the measurement was taken and val the recording at that particular time moment.[4] Essentially each sensor v_i has locally the following timeseries $list(v_i) = o_{i1}, o_{i2}, \ldots, o_{im}$, where o_{ij} denotes the recording of the i^{th} sensor node at the j^{th} time moment. Each time moment could logically be viewed as a collection of n timeseries each with m values. A node can maintain several lists (e.g. temperature, humidity, others); for simplicity we assume that only one such time-series is being maintained. We look at two main classes of queries.

Temporal Queries: The queries we consider will allow the user to find the state of the sensor network at different time intervals, but also to identify intervals that certain conditions hold. Examples of such queries are: *"Find the time intervals such that the sensor values satisfy a given condition,"* and *"Given a sequence of values, identify time intervals that show similar sequences in the values recorded by the sensor."*

Top-k Queries: An example of a top-k query is *"Find the k time instances with the highest average reading across all sensors."* More formally, consider $Q = (q_1, q_2, \ldots, q_n)$, a top-k query with n attributes. Each attribute of Q refers to the corresponding attribute of an object and the query attempts to find the k objects which have the maximum value in the following scoring function:

$Score(o_i) = \sum_{j=1}^{n} w_j * sim_j(q, o_i)$, where $sim_j(q, o_i)$, is some similarity function which evaluates the j^{th} attribute (sensor) of the query q against the j^{th} attribute of an object o_i and returns a value in the domain [0,1] (1 denotes the highest similarity). Since each sensor might have a different factor of importance, we also use a weight factor w_j ($w_j > 0$), which adjusts the significance of each attribute according to the user preferences. Note that, similarly to [3], we require the score function to be *monotone*. A function is monotone if the following property holds: if $sim_j(q, o_1) > sim_j(q, o_2)$ ($\forall j \in m$) then $Score(o_1) > Score(o_2)$. This is true when $w_j > 0$.

4 Query Evaluation Techniques

From the sink's point of view, denoted as v', the data in this scenario is vertically fragmented across the network. Therefore answering such a query would require v' to gather the whole space of $n * m$ values. In section 4 we sketch the TJA algorithm which alleviates the burden of transferring everything to the sink.

4.1 A Taxonomy of Data Gathering Techniques

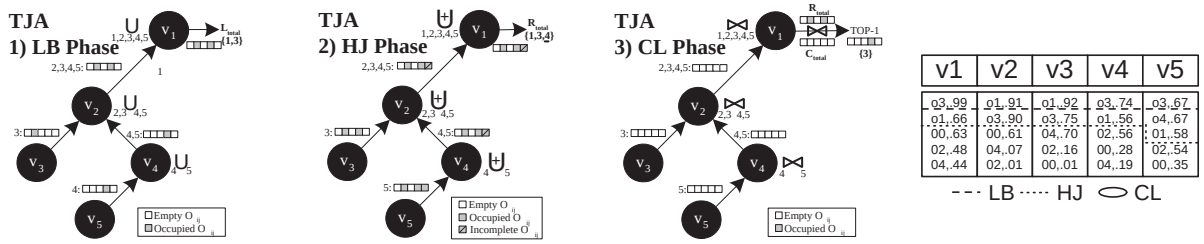
Below we provide a taxonomy of data gathering techniques as a function of the available storage available at each node:

i) Sense and Send (SS): In this naive case each sensor node propagates its generated value towards its parent every time such value becomes available. This is, according to the terminology of [1], the LIST approach.

ii) Sense, Merge and Send (SMS): In this scheme, each node aggregates the values coming from its children before forwarding its values to its parent. This is essentially the TAG approach [6]. In this scheme, all aggregates can not be treated in the same way. For example *Distributive Aggregates (e.g. Sum, Max, Min, Count)* can locally be aggregated into one value. *Holistic Aggregates (e.g. Median)* on the other hand can not be treated in the same way as aggregation into one value can produce the wrong result.

iii) Sense, Store, Merge and Send (SSMS): This is the scheme deployed in our platform, RISE. Each sensor node maintains locally in the flash memory a window of m measurements. This window evolves with time, and therefore, once the limit of the available memory is reached, at each new time moment the oldest measurement is deleted. We note however that given the capacities of flash cards m can be very large. Registered queries can perform some local aggregation, if the correctness of the query outcome is not violated, before values are propagated towards the parent. Note that this is not possible in current systems such as TinyDB. In TinyDB users are only allowed to define fixed size materialization points through the `STORAGE POINT`

[4] Sensors are time synchronized through a lower layer mechanism (e.g. The Operating System).



v1	v2	v3	v4	v5
o3,.99	o1,.91	o1,.92	o3,.74	o3,.67
o1,.66	o3,.90	o3,.75	o1,.56	o4,.67
00,.63	00,.61	04,.70	02,.56	01,.58
02,.48	04,.07	02,.16	00,.28	02,.54
04,.44	02,.01	00,.01	04,.19	00,.35

Figure 2. The QST for the three phases of the TJA Algorithm: 1) Lower Bound (LB), 2) Hierarchical Join (HJ) and 3) Clean-Up (CL) Phase. The table shows the objects qualifying in each phase.

clause. This allows each sensor to gather locally in a buffer some measurements. However the measurements can not be utilized until the materialization point is created in its entirety. Furthermore, even if there was enough on-board memory to store MBs of data in such a point, the absence of efficient access methods makes the retrieval of the desired values quite expensive.

The three gathering techniques outlined above basically represent the scale of available memory at the sensor nodes (i.e. $SSMS \supset SMS \supset SS$). We believe that although the SMS approach offers in practice the most efficient way to cope with short-epoch applications, the SSMS approach will be more practical for long-epoch applications.

We note that under the SS model evaluating the kinds of queries we propose requires sending all information to the sink. Under the SMS model we can design algorithms that perform aggregation or more sophisticated computation in the network, there are however significant limitations. Due to the short-epoch emphasis of this model when information gets older than the window of interest, we have to discard this information or we have to transmit it for permanent storage to the sink (or other specially designated nodes in the network).

4.2 Providing Local Access Methods

Efficiently evaluating the queries described above requires efficient access to the data that is stored on the "external" flash memory. Therefore we plan to deploy the certain access methods (indexes) directly at the sensor nodes. These access methods will serve as primitive operations for the efficient execution of a wide spectrum of queries. Since the flash card on each node v_i can only hold m pages ($o_{i0}..o_{im}$) the available memory is organized as a circular array, in which the newest o_{ij} pair always replaces the oldest o_{ij} pair. Note that this *sorted file organization* allows each sensor to have random or sorted access *by timestamp* in $O(1)$, without the requirement of any index. Next we address how indexes become useful, when we need to have access *by value* as well as the involved challenges.

i) Random Access By Value: An example of such operation is to locally load the records that have a temperature of 70F. In order to fetch records by their value we will use a *static* hash index. We use a static index for simplicity reasons and because the construction of a dynamic hashing index, such as *extendible* or *linear*, might be considerably more power demanding (e.g. due to page splits during insertions).

ii) Sorted Access By Value: An example of such operation is to locally load the records that have a temperature between 50F-70F. In order to fetch records by their value we will use a simple B+ tree index. This index is a minimalistic version of its counterpart found in a real database system. It consists of a small number of non-leaf index pages which provide pointers to the leaf pages. Depending on the sample rate of a query, we expect to have a number of insertion and deletion that need to be handled efficiently. For this purpose we plan to keep the non-leaf pages in the MCU flash memory (32KB) which has space for tens of such pages (each page is 512 bytes).

4.3 Efficient Top-k Query Evaluation in RISE

We now sketch a *Threshold Join Algorithm* which is an efficient top-k query processing algorithm for sensor networks. Our algorithm decreases the number of objects that are required to be transmitted from each sensor, by using an additional probing and filtering phase. More specifically, the algorithm consists of three phases:

1) the *Lower Bound* phase, in which the sink collects the union of the top-k results from all nodes in the network (denoted as L_{sink}=$\{l_1, l_2, \ldots, l_o\}$, $o \geq k$),

2) the *Hierarchical Joining* phase, in which each node uses L_{sink} for eliminating anything that has a value below the least ranked item in L_{sink},

3) the *Clean-Up* phase, in which the actual top-k results are identified.

Figure 2 shows a graphical illustration of the execution of the three phases of our algorithm. We have tested our algorithm in a Peer-to-Peer network using a real dataset of temperature measurements collected at 32 sites in Washington and Oregon.[5] Each site (node) maintained the aver-

[5] http://www-k12.atmos.washington.edu/k12/grayskies/ .

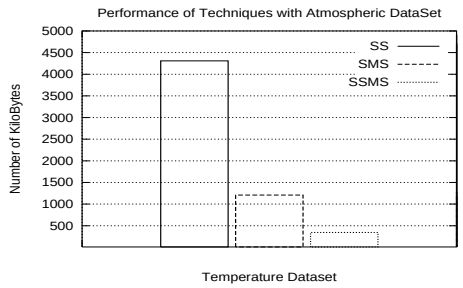


Figure 3. Number of bytes transmitted in the *SS*, *SMS*, and *SSMS* models using atmospheric data.

age temperature on an hourly basis for 208 days between June 2003 and June 2004 (i.e. 4990 time moments), and our query was to find the 10 moments at which the average temperature was the highest. Our algorithm uses efficient index structures to execute efficiently. In Figure 3 we compare our approach with the *SS* approach (sending all data over the network), and our results indicate that *SS* consumes an order of magnitude more network bytes than the *SSMS* approach. We also compare our approach with a simpler approach that computes the scores of all tuples in the network, combining partial results as data are transmitted to the sink. This approach does not use any index methods, and can be implemented in the *SMS* framework. Our preliminary results show that our approach significantly outperforms this technique.

4.4 TinySQL Extensions

In order to be able to run top-k queries using the TinySQL [6] syntax, we plan to add the `LIMIT TO` k clause, which will identify that the user is only interested in getting the k highest ranked results. Additionally we will incorporate the `SENSE_FREQUENCY` clause, which will define the rate at which the sensors have to read measurements from their environment. We believe that the declarative nature of TinySQL, in which the user specifies what he wants to retrieve without specifying how to get it, is highly appropriate for running queries in sensor networks. Such an approach provides simplicity and insulates the user from the physical storage details.

5 Related Work

Systems which propose a declarative approach for querying sensor networks include TinyDB[5] and Cougar[8]. These systems achieve energy reduction by pushing aggregation and selections in the network rather than processing everything at the sink. Both approaches are optimized for sensor nodes with limited storage and relatively short-epochs, while our techniques are designated for sensors with larger memories and longer epochs. In *Data Centric Routing (DCR)*, such as directed diffusion [4], besides in-network aggregation low-latency paths are established between the sink and the sensors. Such an approach is supplementary to our framework. In Data Centric Storage (DCS) [7] data with the same name (e.g. humidity measurements) are stored at the same node in the network, offering therefore efficient location and retrieval. However the overhead of relocating the data in the network will become very expensive if the network generates many MBs of GBs of data. We believe that DCS is not appropriate for the class of applications we discussed in this paper. Local compression techniques, such as the one proposed in [2], would improve the efficiency of our framework and their investigation will be a topic of future research.

6 Conclusions

In this paper we discussed many of the data management issues that arise in the context of the *RISE* sensor network platform. In RISE, sensors feature large memories which creates a new paradigm for power conservation in long epoch applications. We believe that many applications can benefit from a large local storage, as such storage can be used for local aggregation or compression before transmitting the results towards the sink. We expect that this in addition with the provisioning of efficient access methods will also provide a powerful new framework to cope with new types of queries, such as temporal or top-k, that have not been addressed adequately to this date. In the future we plan to investigate the effectiveness of our framework.

References

[1] Considine J., Li F., Kollios G., Byers J., "Approximate Aggregation Techniques for Sensor Databases". In *ICDE'04*, Boston, MA, 2004.

[2] Deligiannakis A., Kotidis Y., Roussopoulos N. "Compressing historical information in sensor networks", In *SIGMOD'04*, Paris, France, 2004.

[3] Fagin R., "Fuzzy Queries In Multimedia Database Systems", In *PODS'98*, Seattle, WA, 1998.

[4] Intanagonwiwat C., Govindan R. Estrin D. "Directed diffusion: A scalable and robust communication paradigm for sensor networks", In *MobiCOM'00*, Boston, MA, 2000.

[5] Madden S.R., Franklin M.J., Hellerstein J.M., Hong W., "The Design of an Acquisitional Query Processor for Sensor Networks", In *SIGMOD'03*, San Diego, CA, 2003.

[6] Madden S.R., Franklin M.J., Hellerstein J.M, Hong W.. "TAG: a Tiny AGgregation Service for Ad-Hoc Sensor Networks", In *OSDI'02*, Boston, MA, 2002.

[7] Shenker S., Ratnasamy S., Karp B., Govindan R., Estrin D., "Data-centric storage in sensornets", ACM SIGCOMM Computer Communication Review, vol 33-1, 2003

[8] Yao Y., Gehrke J.E., "Query Processing in Sensor Networks", In *CIDR'03*, Asilomar, CA, 2003.

[9] Crossbow Technology Inc. http://www.xbow.com/