TRLabs Report

on

Query for Sensor Networks: Issues, Techniques, and Directions

By

Sayed Ahmed
sayed@cs.umanitoba.ca

May 26, 2004

Supervisor
Prof. Rasit M. Eskicioglu
rasit@cs.umanitoba.ca

Abstract

Advances in hardware for sensor nodes that combine sensing devices, embedded processors, and communication components have made the large-scale deployment of sensor networks a reality. Various sensor network applications ranging from habitat monitoring to military applications require sensor nodes to collect huge amount of data over a continuous time period. For the placement, management, and processing of the sensor data, a data storage, management and query processing policy is necessary. Previously, sensor data was only carried to the base stations and stored there for offline processing and querying, and there were no storage at the sensor nodes. However, now-a-days the view of the sensor database is treated as a distributed database. For efficiency purpose and to conserve battery power sensor data is also temporarily stored in tiny databases at sensor nodes and some processing is also done at the tiny sensor nodes. Besides, the individual sensor network spread over a large geographical area can also be interconnected to form a wide area sensor network like the Internet. The Intel IrisNet is such an example. In such configuration, the base station databases are definitely distributed over a large area and the data can also be replicated to intermediate workstations to provide faster query response. Efficient querying and caching strategy for this wide area sensor database is provided. In this paper, we motivate the view of the sensor database as a distributed database and attempt to identify the key query processing techniques used in sensor networks, evaluate them in terms of efficiency, scalability, and applicability and outline the most challenging future research directions. We start by identifying a set of design goals and challenges related to query processing over sensor networks. The evaluation of the techniques is guided by the distinctive query processing features supported by both conventional and wide area sensor networks.

Keywords: Sensor networks, conventional and wide area sensor networks, query processor, distributed database, streaming data, continuous queries.

1 Introduction: Motivation and Objectives

Recent improvements in technology have enabled the widespread deployment of sensor networks consisting of a large number of sensor nodes with sensing, computation, and communication capabilities. Sensor networks aim at providing an efficient and effective bridge between physical and computational worlds. Typical sensor network applications show a great deal of diversity including contaminant transport monitoring, marine microorganisms analysis, habitat sensing, seismic and home monitoring, and more. From small, low-powered, and wireless sensors to web cams, microphones, and pressure gauges, pervasive sensors provide excellent new services for these applications. Querying and monitoring the physical world is one of these services bearing utmost importance. Consider for example, a conventional sensor network used to detect the presence and location of a specific bird in a forest matching the birds murmurings and physical characteristics. Sensors are arbitrarily and densely deployed throughout the forest and the sensors collect data over a continuous time period. The data can be directly transferred to the base stations for analyzing the existence of a bird at the base stations while storing the data in tiny databases in sensor nodes temporarily, analyzing the data and coordinating with neighbors also performing some aggregation and operation to decide whether the data is useful or not and then sending the useful data to the base stations can effectively reduce communications and preserve battery life. For such application, a tiny database is required at each or most of the sensor nodes, an efficient query plan is needed to decide which intermediate nodes will store and perform aggregation, processing or filtering also a plan is needed to decide what part of query processing should be done in-network and what part should be carried out in the base stations. Now consider a Trip Planner service, another example of wide area sensor network, used for finding out the current traffic condition of streets leading to a users destination. The user enters into a PDA (or a car navigation system) his origin and destination and gets back information regarding the traffic condition of every possible street that leads to the destination. A three-fold requirement is envisioned to create such a Trip Planner service. First, sensors networks located in different locations spread over the streets capable of determining the current traffic rate. Second, a sensor database is needed that can store dynamic information such as the current traffic rate and the speed of vehicles along with some static information such as possible ways to reach the destination. Third, a web accessible query processing system is needed to provide replies to high-level user queries.

Unfortunately, the resource constraints related to sensor nodes such as computation, communication, power consumption, uncertainty in sensor readings have posed a numerous challenges in query processing for sensor networks. Therefore, research on efficient query processing for conventional sensor networks has become an important activity. Besides in wide area sensor network an easy web accessible interface is needed for the users to pose their queries also directing the queries to the appropriate site is very important as response time is critical. To this end, databases in sensor networks are recently attaining a great attention of researchers.

So far, reasonable work has been done on different query processing techniques in the context of both conventional and wide area sensor networks. Conventional sensor networks consist of very small, power-constrained, wireless sensors having very simple processors, little memory and limited sensing capabilities. The Directed diffusion [12], the Cougar approach [3, 23, 24], the TAG [16], the Fjords [15] and some other techniques have been devised for querying the conventional networks. These query processing techniques operate directly on continuous and never-ending streams of sensor data and take special care on using low power as the sensor nodes are power-constrained. Wide area sensor networks interconnect conventional sensor networks spread over a large geographical area. Besides, wide area sensor networks can have large (powered and high-bit-rate), intelligent sensors [10] like web cams, pressure gauges with storage and processing power. Web cams can be placed in all the parking spaces in a wide region to overlook the free parking spaces. The web cams of a parking space form a local sensor network while networks in different places create a wide area sensor network as a whole when interconnected to each other or when all of them are connected to the Internet. Databases used by these networks are also dispersed and contain data that correspond to the most recent updates by the sensors. Unfortunately, little research has been conducted on the query processing in wide area sensor databases. The IrisNet (Internet-scale Resource-intensive Sensor Network Services) project works with wide area sensor databases and develops a novel query processing technique that is applicable to hierarchical distributed databases created by any wide area sensor network.

Our overall strategic goal is to identify the key features adopted by current query processing techniques for sensor networks, evaluate them in terms of efficiency, scalability, applicability, and reliability and to outline the most challenging future research directions for the development of new techniques. We envision providing emphasis on:

Identifying the requirements and challenges for sensor networks queries: The requirements for query processing may vary depending on the types of sensor networks and the applications being deployed. Nevertheless, a number of common requirements are shared among all sensor network applications regardless of the usage such as a simple query language, aggregation of query data, less query response time, and multi query management. At the same time, sensor networks pose a number of query processing challenges due to the resource constrained sensor nodes (low power, small size, and less storage) and the nature of sensor data.

Identifying the interoperability and integration issues: Query processing techniques should be compatible to both conventional and wide area sensor networks. Besides, integration of conventional and wide area sensor databases should be supported to provide a common query language interface.

Optimum design: The first tendency could be to optimize each and every component to the maximal extent. Identifying the area where to put the main optimization effort is important. For example, supporting more in-node aggregation operations, filtering sensor data, and supporting the usage of intermediate result for different queries could be important design issues.

Identifying trade-offs due to resource constraint: A number of possible trade-offs should be considered depending on the sensor node resources. For example, increasing node computation to minimize the amount of communication in order for the power consumption to be kept a minimum could be a design goal where the underlying network architecture supports sufficient local storage.

The remainder of the paper is organized as follows. Section 2 outlines important query processing challenges. Section 3 then presents different query processing techniques and evaluates them. Next, section 4 provides an integrated (conventional and wide area sensor database) query processing architecture, section 5 provides the challenging future research directions for the development of new querying techniques. Finally, the paper ends with conclusions in section 6.

2 Query Processing Challenges

In this section, an overview of research problems for query processing in sensor networks is presented in the context of both conventional and wide area sensor networks. Mentionable, as conventional sensor networks are part of the wide area sensor networks therefore challenges for conventional sensor networks are also part of the wide area sensor networks.

2.1 Conventional Sensor Network Databases

Streaming Data: In traditional database systems, data is static and the query processor pulls data from the disk, organizes the movement and handling of data but sensor networks involve streaming data where data is continuously pushed to the query processor, which must react to the arriving data. The data arrival rate sometimes is extremely high or bursty, hence processing time or memory usage should be handled carefully. Data is dynamic in nature so must be processed dynamically as it arrives and can be spooled to the disk in background mode [20].

Continuous Queries: Monitoring applications are common sensor applications where queries remain continuously active. When new data comes or an event happens, data are supplied to the stored and continuously active queries whereas in a traditional database system, the arrival of queries initiates access to the static database. The streams pushed to continuous queries can be effectively infinite leading the query processors to include different query semantics, non-blocking query operators, and tolerance of failure [20]. Continuous queries can be extremely long lived, which makes them vulnerable to changes over time in performance or data characteristics and qualities.

Resource Constrained Sensor Nodes: Sensor nodes are resource-constrained having very low power, limited memory, very simple processors, and limited sensing capabilities. Besides, wireless communications consume much battery power. Therefore, the query processor should be designed so as to overcome these limitations [23].

Aggregation: Aggregation refers to collecting, filtering, and processing sensor data in the network, and supplying the result to the central nodes. Aggregation reduces wireless communication and hereby minimizes power usage. Hence, an efficient aggregation plan is needed considering the volatility of the underlying data [23].

Query Languages: Technological development is leading to various sensor based applications driving to diverse requirements for the query language. Hence, the development of a user friendly, efficient, generalized query language is at a difficult point [23].

Query Optimization: A complex query can consist of a lot of parameters and operators along with diversified user requirements. A great variety of optimization options are also available for such complex queries but it is the query processors responsibility to find a good plan within a short computational time to execute these queries [23].

Catalog Management: To generate a good plan within a short time for user queries, the query optimizer needs to know the current status of the network. Data about the network topology and status of the network helps to calculate the costs and benefits of different query plans and make a good selection among them [23].

Multi Query Optimization: Simultaneous multiple query support and optimization is another challenging issue [23].

2.2 Wide Area Sensor Network Databases

In addition to the above mentioned query processing challenges wide area sensor networks offer some unique query processing challenges not common to the conventional sensor networks. They are as follows:

Data Partitioning and Placement: Physical placement and keeping track of sensor data is an important issue due to the huge volume of data and multiple and geographically dispersed databases. The data is partitioned in such a way that query processor can find appropriate site for providing results within a short period [8].

Caching: An efficient caching strategy can help a lot. If data is cached depending on the query and application in different sites they can provide dramatic decrease in query response time [8].

Query Processing and Query Language: The query language should support diverse applications and at the same time should be efficient and user friendly. The query processing technique should be able to handle a large number of sensors, diverse queries, multiple concurrent queries, and large volumes of data [8].

3 Different Query Processing Techniques

This section presents different query processing techniques for both conventional and wide area sensor networks with their relative advantages and limitations. First we discuss different query processing issues in conventional sensor networks that work as the terminal of wide area sensor networks. Therefore, query processing and optimization architectures in conventional sensor networks affect the wide area sensor network in a great extent. Besides, a conventional sensor networks in many instances are treated as distributed databases such as the Cougar Approach, TinyDB. As a result distributed query processing techniques are also applicable to these instances.

3.1 Conventional Sensor Networks

3.1.1 Directed Diffusion

Directed diffusion defines a data centric dissemination and co-ordination paradigm for delivering sensed data to the user. User queries or tasks are inserted as descriptive interest messages through a sink or entry node such as Type = four footed animal, Interval = 20 ms Duration = 10s, Rectangle = [-100,100,200,400]. The interest query traverses from the sink node to the destination nodes using broadcast/multicast to the neighbors. During the traverse, gradients are created to keep track of delivering nodes from which interest messages are received with some identification of the interest message. Target nodes and neighbors co-operate to sense an object and the results traverse back to the sink following the gradients to the sink [12]. Each node contains an interest cache to store the distinct interests. From the same sink multiple queries can be posed or different users can pose different or same queries from multiple sites. So an interest cache contains multiple distinct interest entries. Under each distinct interest entry there are several gradients containing data rate and duration fields with the address of the node from where interest message is received. When an object is detected data messages traverse through gradients from sources to sinks. Each node also contains cache for distinct data messages so that the same message is not forwarded multiple times. Here we see at each node a storage is needed. The size of the storage is dependent on the user requirements and the number of queries and events. Efficient query plan is needed to better utilize the stored data also aggregation on the stored data can provide faster and accurate event detection. Co-ordination plan among multiple sensors (i.e. database sites) can help [12].

Directed Diffusion is non-declarative and significantly lower-level and not much user friendly. Hence Directed Diffusion pays a lot in respect of usability and it is not much suitable for aggregate queries. However, in directed diffusion, limited data aggregation is performed on every node on the path from the source of the data to the sink of the interest. Directed diffusion has significant energy efficiency; even with relatively un-optimized path selection it provides sufficient savings in energy efficiency [12].

3.1.2 Cougar Approach

Cougar approach mainly focuses on in-network distributed query processing. In case of centralized traditional sensor networks, sensors are preprogrammed, and always send data to the central node for offline querying and processing. In addition, the network and the applications are static. Besides, there is huge consumption of battery power due to huge wireless communications [3, 23, 24].

Cougar approach solves these problems to a great extent. The main idea of cougar approach is in-network data processing. The sensors are capable of storing some data and perform processing like aggregating records or eliminating irrelevant records and send only the necessary data to the base station. Besides, a declarative, user friendly query language is developed. Hence, users are shielded from the internal structure of sensor networks.

There is a query proxy layer in each sensor node in between network and application layer. Whenever the gateway nodes (adjacent to base) receive queries from the users at base stations a query optimizer residing in the gateway node generates distributed query plans. Query plan specifies data flow routes and exact computation plan i.e. which node will do computation and which node will collect data. The plan is then delivered to all the related sensors of this plan. Also control structures are needed to synchronize sensor behavior. Query optimizer assigns a leader for a group of sensors called slaves or non-leader nodes who will collect data and send data to the corresponding leader node. Leader nodes do computation [3, 23, 24]. To the end, Cougar approach introduces aggregation, in-network-query processing, query proxy layer, a declarative query language in addition to using less battery-power [3, 23, 24].

Cougar approach views the sensor network as a huge distributed database system where each sensor node is a database site that holds part of the data. So, cougar approach tries to adapt existing techniques from distributed and heterogeneous database systems to sensor networks. Cougar approach proposes a loosely coupled distributed architecture to provide aggregation and computation [3, 23, 24].

Query plan for a non-leader node is shown in figure 1. Non-leader nodes read sensor values using a scan operator periodically or depending on the event and then send data to the leader node.

The query plan for the leader node is shown in figure 2. The leader node in the figure2 contains an AVG operator to compute the average of all the sensor readings sent to him by the non-leader nodes under this leader node. The leader nodes can contain several application oriented operators and rules [3, 23, 24].

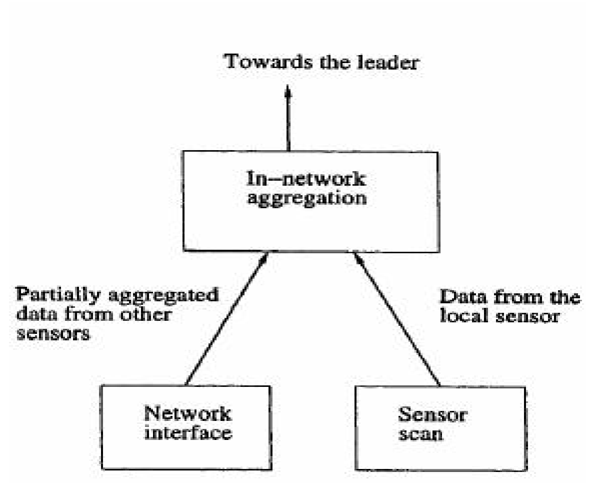
Towards the leader
In–network aggregation
Partially aggregated data from other sensors
Data from the local sensor
Network interface
Sensor scan

Figure 1: Query plan at non-leader nodes [23].

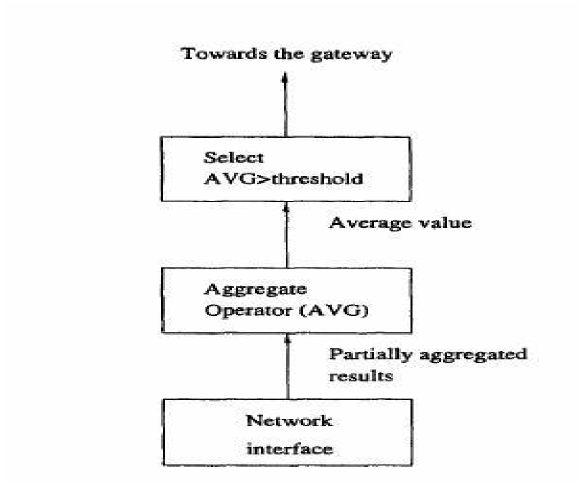

Figure 2: Query plan at leader nodes [23].

3.1.3 Acquisitional Query Processing (ACQP)

ACQP works in such a setting where the data to be processed does not exist at the sites when the query is issued but data is gathered by collecting samples from special sensing devices on the fly. ACQP uses different techniques to decide when and where to sample and also which data to be delivered to the central nodes with the combination of in-network aggregate processing like cougar approach. Based on these concepts, TinyDB, a DBMS for sensor networks is built. Unlike many solutions for sensor query, TinyDB systems do not need to be programmed with C code for sensors. Instead, similar to traditional DBMS systems, TinyDB provides a simple, SQL-like interface to specify the query. In addition, the query can integrate additional parameters such as the rate at which data should be refreshed. Upon receiving a user query, TinyDB collects data from the sensor nodes, filters, aggregates, and then delivers to a workstation using power efficient in network processing [14, 17].

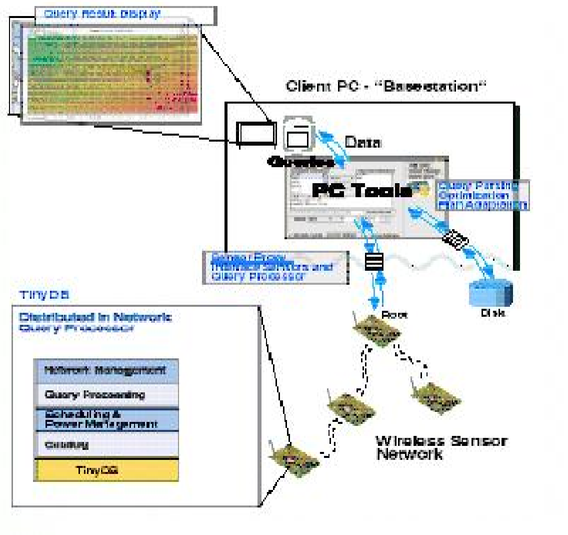

Figure 3: TinyDB Architecture [14].

3.1.4 TinyDB

TinyDB features are as below [2]

- TinyDB runs over TinyOS operating system for sensor networks.
- TinyDB provides a simple, SQL-like interface to specify user query integrated with additional parameters such as expected data rate to use.
- TinyDB, needs to be installed onto each mote in the sensor networks as well as at the base stations.
- TinyDB provides a Java API for writing applications that query sensor networks.
- TinyDB also comes with a simple graphical query-builder and result display system that uses the API.

In addition, TinyDB provides the following features [2] those indirectly help query processing.

- **Metadata Management:** In TinyDB, the metadata catalog maintains the attributes and commands available for querying and invocation. Attributes can be sensor readings or internal software/hardware parameters.

- **High Level Queries:** TinyDB uses a SQL like declarative query language that makes user queries as well as application programs to be written easily. With the change in sensor network topology or hardware the query semantics need not to be changed.
- **Network Topology:** TinyDB manages the network status and topology by tracking neighbors, maintaining routing tables, and Ensuring that every mote in the network can efficiently and (relatively) reliably deliver its data to the user.
- **Multiple Queries:** TinyDB allows simultaneous multiple queries that can be of different sample rates, and access different sensor types. TinyDB can effectively share works and intermediate results among simultaneous multiple queries.

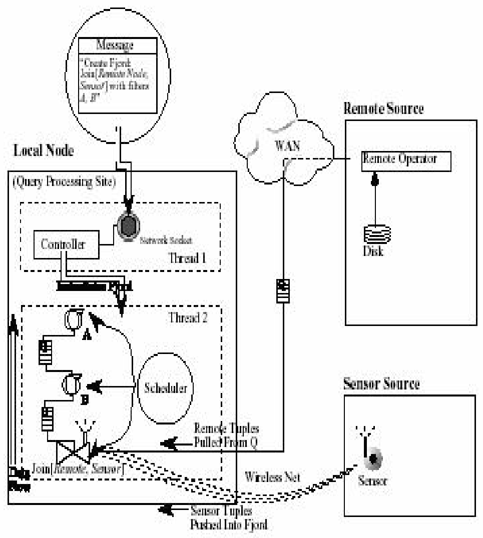

Figure 4: A Fjord: Join [Remote Node, Sensor] with filters A and B [15].

3.1.5 Fjords Architecture

The Fjords architecture [15] supports and handles multiple simultaneous queries over many sensors, and reduces sensor resource requirements while also ensures high query throughput. The low level architecture of a sensor network stream query processor is introduced using two techniques. First, Fjords architecture combines proxies, non-blocking operators, and conventional query plans [15]. Second, sensor proxies serve as a bridge between sensors and query plans, using sensors to facilitate query processing while being efficient to use their power, processor, and communications limitations.

The Fjords architecture provides support for combining streaming data with disk data i.e. previous data in the database. This database is queried by both traditional queries and continuous queries. Fjords allow integration of multiple queries in a single query plan and can efficiently handle multiple inputs and outputs concurrently. Traditional databases only contain static data but do not have much support to integrate streaming data with static data but for sensor network applications this combination is a must. Therefore, Fjords provides a hybrid database storage approach and querying support on the database. Such type of storage and querying approach is a must to handle continuously changing stream sensor data also for any system that handles variable nature data. Telegraph Query Processing System [5], tries to extend traditional query processing techniques to a wide variety of nontraditional as well as variable nature data sources. Telegraph when integrated with the Fjords, empowers a query processing system that can query over networks of wireless, battery powered sensor devices efficiently which is not possible using traditional query processing approaches [15]. A figure [15] representing Fjords architecture is given in figure 4.

3.1.6 Continuous Queries

Recently, there has grown much interests in querying streaming data, and continuous query processing system over streaming and continuous data from wide variety of sources. TelegraphCQ [5] is a recent project at Berkeley to handle continuous queries which focuses on the extreme adaptability of the query processor with streaming or continuous or constantly changing data and it has the required novel architecture to adapt. NiagaraCQ [6] is an XML-based engine that tests with different static plans for continuous queries and using the efficient plans built a continuous query processing system. Also in NiagaraCQ two queries having the same input can share a module. Babu and Widom [1] deal with STREAM, which focuses on computing approximate results, analyzing the memory requirements of posed queries and running queries in a bounded amount of memory. The Aurora [4] system works with quality-of-service for queries to meet the need for heterogeneous user requirements. The computation of correlated aggregate queries over streams is addressed in Gehrke et al. [9]. Gehrke et al. [9] also presents efficient techniques to obtain an approximate answer in a single pass. Yang et al. [22] discusses data structures for computing and maintaining aggregates over streams to support fast lookup of aggregate results based on time line. Sadri et al. [19] proposes a query language to express continuous queries named as SQL-TS. The SQL-TS is an extension to the SQL which expresses queries in terms of time-series of data [19].

3.2 Wide Area Sensor Networks

There is little research on wide area sensor databases while the Intel IrisNet project has done a major work to address the challenges to support a large number of sensors and high query volumes in wide area sensor databases. The sensor database is partitioned across multiple sites that operate in parallel. User queries are directed to the sites that contain the results. A site manager called organizing agent works in each site. Communications with the sensors are through sensor proxies called sensing agents, which are the powered and internet connected PCs. Sensor proxies collect, filter, and aggregate data then send update queries to the sites that own the data [8, 10].

To make queries flexible, easy to use and to support the growing trends to XML standard in the IrisNet XML database system is used. XPATH or XSLT for querying, and SixDML or XUpdate for updates are used. The distributed site hierarchy is represented as an XML document. The data in sensor networks is fairly diverse and heterogeneous, and hard to capture in a rigid data model. The schema of the data needs to be constantly adapted to support the growing and diversified needs of the users, and evolving capabilities of the sensors. Dynamic insertion and deletion of new fields is easy in XML. Besides, as sensors take data from a geographic location, it is quite natural to organize the sensor data into a geographic/political-boundary hierarchy (as depicted in Figure 5). Representing and making use of such a hierarchy is quite natural in XML, but very challenging in the relational, or an object-relational data model [8].

Sensor data is placed in any number of host nodes and near to the sensors but for efficient querying data are cached to anywhere depending on the needs of the queries and applications. An efficient partitioning and caching strategy supporting partial match caching is presented which ensures that even partial matches on cached data can be exploited and that correct answers are returned. A dynamic self starting distributed query is supported where the queries based on the XPATH query statement and the XML document find the least common ancestor (LCA) of the hierarchical database fragments where query results exist and a DNS like server finds the IP address of the LCA. Moreover, a novel query-evaluate-gather based on XSLT determines which data of the local database is part of the query result and how to collect the missing data [8].

The IrisNet has similar structure with DNS [18] and LDAP [21] but DNS, and LDAP support a small set of lookup queries whereas IrisNet supports rich query lan-

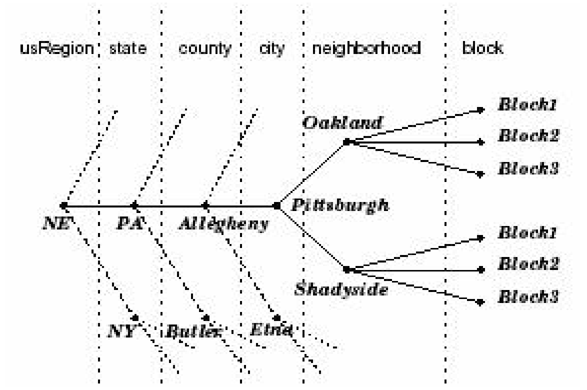

Figure 5: Site hierarchy [8].

guages. Latest works on peer-to-peer databases [7, 11] are quite similar to the IrisNet works. The IrisNet is mostly hierarchical but for performance reasons, the participating sites are treated as co-operating peers. The IrisNet differs from peer-to-peer databases because of its query processing part, caching, and partitioning strategies. However, DNS, and LDAP support a small set of lookup queries whereas the IrisNet supports rich query languages like XPATH and XSLT.

Latest works on peer-to-peer databases [7, 13] are quite similar to the IrisNet works. The IrisNet databases are organized mostly hierarchically but for performance reasons, the participating sites are treated as peers who co-operate with each other to store and query data. However, the IrisNet differs considerably in query processing part than these works because it uses XML and the XPATH query language.

3.2.1 Query Processing in Wide Area Sensor Networks

Consider the following XPATH query as provided in [8] as an example of a challenging query in wide area sensor networks using XPATH query.

```
   /usRegion[@id=NE] /state [@id=PA] /county [@id=Allegheny] /city[@id=Pittsburgh]
/neighborhood [@id=Oakland OR @id=Shadyside] /block [@id=1] /parkingSpace
[available=yes]
```

The query asks for the number of free parking spaces in Oakland and Shadyside. Suppose, the query is executed in a site containing the following XML document fragment [8].

```
  <usRegion@id=NE><state@id=PA><county@id=Allegheny><city@id=Pittsburgh>
<neighborhood@id=Oakland><block@id=1>
</parkingSpace><parkingSpace@id=1><available>yes</available></parkingSpace>
</parkingSpace><parkingSpace@id=2><available>no</available></parkingSpace>
</block></neighborhood>
</city></county></state></usRegion>
```

The query result contains a free parking space in Oakland, block one. The challenge is to decide whether this is the complete answer or not as the result does not say anything about Shadyside. Shadyside may not have a free parking space or the local database does not contain information about Shadyside. However, the query result does not indicate any of the two possible cases [8].

To remove the problem one might split the query into two queries, one for Oakland and another for Shadyside. But the result is all the same. Second, placeholders might be introduced in the database that tag missing data. However, data for all neighborhoods in Pittsburgh should be included in the site which is not a scalable solution. Finally, metadata like telling what part of the XML document is in the site database can be introduced. A good approach is to integrate the database itself with metadata on what part of the XML document is in the site database, as tag attributes associated with the relevant portion of the database. The attributes must indicate what data are present or missing also how to gather the missing data. However XPATH is not sufficiently powerful to utilize these tag attributes. Hence, XSLT is used [8].

3.2.2 Proposed Solution to Query Processing Challenges

In this section different parts and schemes of the IrisNet query processor will be discussed.

1. **Data Partitioning** There are two distinct parts for data partitioning. First, data ownership that indicates which node owns what part of the data, and second, data storage which describes the actual data stored at each site. To facilitate data partitioning certain nodes in a document are defined as IDable nodes (figure 6), who have an id attribute which is unique among its siblings with the same element name but whose ID type attribute (¡element name, id¿) is globally unique. The root as well as the parents of the IDable nodes is also IDable nodes. IDable

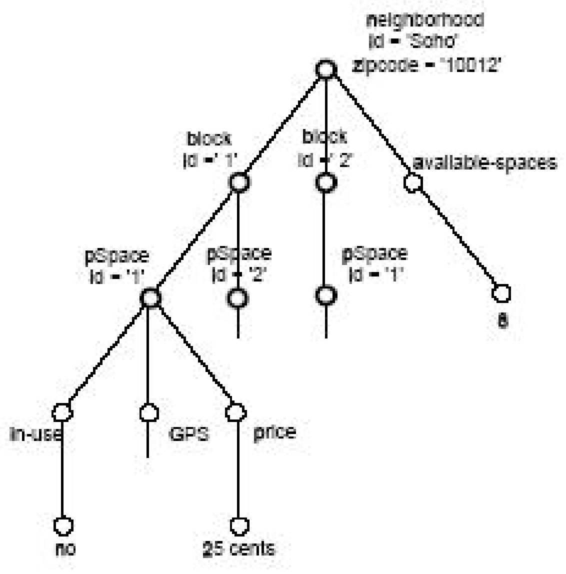

Figure 6: IDable nodes in a document [8].

nodes contain two types of information. First, local information which is defined to be the document fragment containing (1) all its attributes, (2) all non-IDable children and descendants, and (3) the IDs of IDable children second, local ID information which is defined to be the document fragment containing (1) ID of the node (2) the IDs of all its IDable children [8].

2. **Data Ownership & Storage**

   Each site owns an arbitrary set of nodes from the document following two rules [8]. First each node must be owned by exactly one site. Second, only IDable nodes can have owners different than their parents. This provides great flexibility in partitioning leading to many alternative partitions. For example a site may own a node, a few grandchildren, and cousins, but not the intervening nodes in the hierarchy. In spite of this flexibility, the algorithm is able to find the correct answer.

   The data, stored at each site is an arbitrary combination of either the local ID information or the local information of a set of nodes. Two constraints are maintained on the stored data. First, if a site owns a node it must store the local information for the node. Second, if a site stores the ID of a node the site must store the local ID information of its parent i.e. the local ID information of all the ancestors of this node is also stored [8].

To ensure efficient and correct query processing, a special attribute called status is maintained to summarize the stored data for a node. It has four values. First, owned i.e. the site owns the node and contains local information of the node and at least the local ID information of all its ancestors. Second, complete i.e. the site contains the same information like owned but does not own the node. Third, ID-complete i.e. the site has local ID information of this node and all its ancestors but does not contain all the local information for the node. Fourth, incomplete i.e. the site has only its ID. A site must fall into one of these categories [8].

Hence, when a site stores information about a node it also at least stores the IDs of all its IDable children and also the IDs of all the ancestors and their siblings. The site can use its document fragment to answer the query or it knows which parts are missing. The missing parts are always the IDable nodes and the information in the sub trees below them [8].

3. **Caching**

   The purposes of caching are:

   - Providing flexibility in data replication dynamically at multiple sites
   - In spite of dynamic caching maintaining efficient and correct query processing

   In the Intel IrisNet four caching strategy are provided

   - Storing Additional Data
   - Caching Query Results
   - Partial Match Caching
   - Evicting (Cached) Data

   Storing Additional Data: In this scheme a site can hold an additional data fragment along with its regular data fragment but the added document fragment

   - **(C1)** must be a union of local information or local ID information of some nodes [8].
   - **(C2)** if contains local information or local ID information for a node, should also contain the local ID information for its parent (i.e. all its ancestors) [8].

   This addition of new document fragment should not violate the basic data partitioning and fragmentation rules.

   Caching Query Results: In the IrisNet a query is forwarded to the LCA first and then it recursively goes down to the hierarchy to pull together the answer and whenever a (partial) answer is returned to a site, the answer is cached at the site. This approach has two benefits [8].

The same site can answer to later queries on the same data thus avoiding gathering up the answer again [8]. The creation of additional copies of the result that distribute the query workload over multiple sites [8].

Partial Match Caching: It guarantees that even partial matches on cached data can be utilized in generating correct answers efficiently [8].

Evicting (Cached) Data: Any data from any site can be removed but the removal must be in units of local informations or local ID informations of IDable nodes and all the data partitioning rules mentioned above remain valid [8].

4. **Finding Sites**

   Issue:

   When a query is posed in the wide area sensor network the query should be routed to the appropriate site using IP routing. In this section we shall discuss how the system determines the IP address of any site needed during query processing [8]

   IP addresses are needed when

   - an user poses a query at an workstation in the wide area network [8]
   - an organizing agent finds that a sub-query is needed to be posed to another site to generate the remaining answer [8]

   The solution uses DNS-style workstation names for nodes/sites and the names can be constructed from the XPATH queries themselves [8] Searching into DNS for the IP address of the desired site [8] Re-mapping of DNS entries whenever topology changes or the nodes are remapped [8].

   DNS Style Name IDable nodes containing part of the databases are uniquely identified using a sequence of the IDs which refers a path from the root to the node [8]. DNS-style names are simply the sequential concatenation of these IDs extracted from the XPATH queries [8]. Consider a query:

```
/usRegion[@id=NE] /state[@id=PA] /county[@id=Allegheny] /city [@id=Pittsburgh]
/neighborhood [@id=Oakland OR @id =Shadyside] /block [@id=1] /parkingSpace
[available=yes] [8]
```

   For the above query the DNS style name is pittsburgh.allegheny.pa.ne.parking.intel-iris.net Can be generated for any needed IDable node that contains missing data based only the information stored in the site database [8].

5. **Query-Evaluate-Gather**

   When an organizing agent receives an XPATH query, it generates an XSLT program from that query that executes the following algorithm [8].

   1. Let cur is the current node, tag (cur) is the element name of cur and P is the predicate set. Initially cur is set to the root of the document at the site.

   2. Depending on the status of cur in the document, there are four possibilities:

   (a) **status = incomplete:** Divide P such that P = Pid && Prest, and Pid only contains the predicates of the id. If evaluating P against current node returns true then form a sub-query which a post-processor step will use to evaluate the rest of the query. Otherwise, the post-processor step will not ask sub-query. If dividing P is not easy, the node may be part of the answer and evaluate the rest of the query as above using a sub-query.

   (b) **status = id-complete:** The actions are similar. But if tag (cur) is not in LOCAL-INFO-REQUIRED and P = Pid then recurse on the children. Otherwise, ask a sub-query to know the local information for this node, required in the answer.

   (c) **status = owned:** If P is satisfied, then recurse on the IDable children of cur, and if tag (cur) is in LOCAL-INFOREQUIRED copy the local information into the answer being formed. Otherwise, only local ID information is copied

   (d) **status = complete:** The predicates who do not specify consistency requirements the actions are similar to the above case.

6. **Experimental Results**

   In the experiment [8], an artificially generated database consisting of 2400 parking spaces containing two cities, three neighbourhoods per city, 20 blocks per neighbourhood, and 20 parking spaces per block were used. Four types of queries [8] were used. First type of queries collects data from a single block where the full path from the root to the block is specified. Second type of queries collects data from two blocks from a single neighbourhood. Third type of queries collects data from two blocks of two different neighbourhoods. Fourth type of queries collects data from two blocks from two different cities [8]. Four types of architectures (figure 7) were used to execute queries of these types along with a mixed

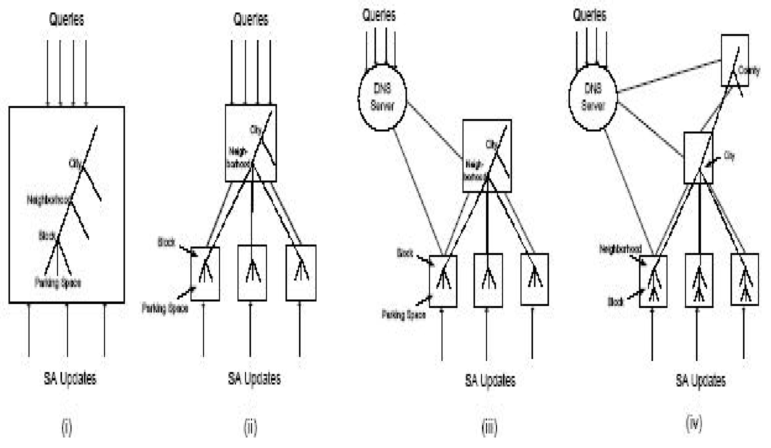

Figure 7: Architectures [8].

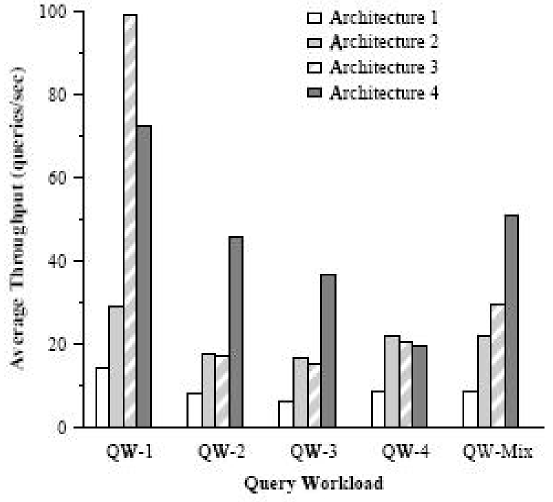

Figure 8: Throughputs [8].

query type. The centralized method (Architecture 1) is not scalable enough for querying, and can grip very few queries efficiently. Distributing only the updates (Architecture 2) can handle more updates, and improves query throughput by a factor of two over the centralized solution. Using DNS for self starting distributed queries (Architecture 3) the throughput for type 1 queries increases by a factor of 3 over Architecture 2. However, all other queries still go through the central server, which is the bottleneck for the other types of queries. The hierarchical distribution of data (Architecture 4) can handle all kinds of queries quite well.

It does perform 25% worse than Architecture 3 for type 1 queries, because it is using 25% fewer machines in processing these queries. However, it performs at least 60% better than the other architectures on the mixed workload [8]. The throughputs are provided in figure 8.

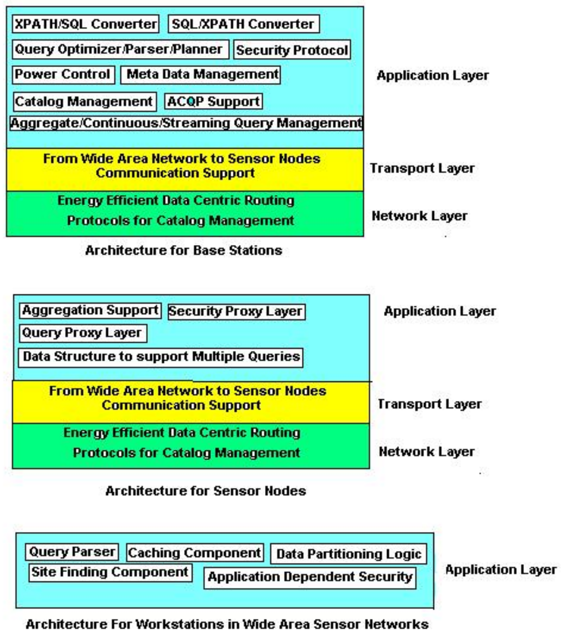
XPATH/SQL Converter
SQL/XPATH Converter
Query Optimizer/Parser/Planner
Security Protocol
Power Control
Meta Data Management
Catalog Management
ACQP Support
Aggregate/Continuous/Streaming Query Management
Application Layer
From Wide Area Network to Sensor Nodes Communication Support
Transport Layer
Energy Efficient Data Centric Routing Protocols for Catalog Management
Network Layer
Architecture for Base Stations
Aggregation Support
Security Proxy Layer
Query Proxy Layer
Data Structure to support Multiple Queries
Application Layer
From Wide Area Network to Sensor Nodes Communication Support
Transport Layer
Energy Efficient Data Centric Routing Protocols for Catalog Management
Network Layer
Architecture for Sensor Nodes
Query Parser
Caching Component
Data Partitioning Logic
Site Finding Component
Application Dependent Security
Application Layer
Architecture For Workstations in Wide Area Sensor Networks

Figure 9: Query Processing Architecture.

4 Proposed Architecture

Based on the above discussion and our study, we propose an architecture for query processing in future sensor networks.

1. Data aggregation and filtering capability in each sensor node as well as in the base stations.
2. A query proxy layer at each sensor node helping to the query plan and query optimization generated at the base stations.
3. A Query optimizer/parser is needed at base stations performing semantic analysis, normalization, optimization of the queries posed at the base stations. Also at each workstation in the wide area sensor networks that run sensor applications (depending on the application) a simple parser can be placed. This parser at the

workstations in wide area sensor networks can use XML document, the fragmentation strategy also the DNS like system to generate an efficient query plan in addition to doing semantic analysis, normalization and so on.

4. At base stations gateway components to make necessary transformations (XPATH to SQL or SQL to XPATH or a traditional query into a continuous/aggregate query flavor) for the interoperability and integration between wide and conventional sensor networks.
5. Security enabled service at each work station in wide area sensor database also in the base stations along with a simple security module at each sensor nodes. Security is not essential for every type of application so the security modules can be developed depending on the applications.
6. Caching, partitioning and site finding component at the workstations in wide area sensor network.
7. Catalog Management, Metadata Management is needed at base stations but sensor nodes should also support them.
8. Data structures, algorithms and architectural supports (i.e. multiple gateways) are needed to support multiple queries and share the common intermediate results among the queries.
9. TCP is not suitable for end to end connection for conventional sensor networks as sensor nodes are data centric attribute based and addressless. An appropriate transport layer protocol is needed at sensor nodes. Base station can work as a bridge between transport protocols for wide (TCP) and conventional sensor networks.
10. Energy efficient, data centric routing algorithms are required.

The architecture described above is shown in figure 9.

5 Future Directions

The analysis of current querying techniques for sensor networks opens a plethora of new research directions at the boundary of sensor database systems and networking:

1. We find that conventional sensor networks generally use SQL like declarative language while their wide area counterpart uses XML and XPATH/XSLT query language. So, coordination is needed between them to work together. Some proxy layer can work in the terminal base stations to convert XML to SQL. Also XML

query can be utilized in the power constrained sensor networks in future. We envision one step further integrating conventional and wide area sensor databases to provide a common query language interface so that no proxy layer/converter is needed. Besides, very changing nature of sensor data can well fit into XML representation. So, we suggest XML and XPATH queries in both wide and local area sensor networks. However, proper experiment and analysis would give the right decision.

2. We see sensor network applications treat sensor data as a stream and continuous. Besides base station database can be treated as hybrid where some static data is kept and continuously updated data are coming. Also the static data may expire after sometimes. In some applications queries are continuous and long-lived. Continuous queries can be active all the time or can act only in the presence of an event. Supporting aggregate queries which reduce much communication energy is also important. Hence, a generalized DBMS is required that can handle diversified types of applications, situations and user needs. Since the sensor nodes support limited storage, the code size should be small. Alternatively, the base/central node can load appropriate modules to the sensor nodes depending on the application. This on-demand loading of modules can use the concept of swapping/paging in the base station hard drive, which requires communication causing much power consumption. Hence, the algorithms need to be power efficient.

3. In wide area sensor networks, different data mining approaches can also be utilized over historical sensor data to find historical trends. Historical trends can be used to provide better answers to user queries. For example, data mining approaches can be used to find the trends how fast a parking space is filled or when the parking space becomes busy. So in addition to mentioning the available free space count data mining approaches will help the user know about the characteristics of the parking lot and take proper decision [8]. The statistical or data mining algorithms can be integrated within each database so that XPATH query using the XML document can retrieve and use the information from appropriate site.

4. None of the current querying techniques for sensor networks are concerned with database security. Integration of security mechanisms inside database can pre-

vent unauthorized user from querying. We propose to introduce a security layer in application layer of each sensor node. In the gateway node, the application layer can add some identification of the authority to the query and the application layer of the other sensor nodes will permit only the queries from valid authority. Besides Encryption techniques in query request and reply can be introduced but encrypting can use more energy leading to much research to utilize appropriate encryption algorithm. Justification of the need of the security and the encryption by the application is also required.

5. Continuous queries that are very frequent and natural to conventional sensor networks must also be supported and handled in wide area sensor databases to make the networks practically useful. Both architectural support and query language support also efficient optimization and query processing plans are needed to pose continuous queries over wide area sensor networks. Yet the Intel IrisNet has not implemented the support but they hope to support continuous queries where the various data structures that the IrisNet used to collect information from different organizing agents will be utilized for this purpose [8]. Continuous query processing supports are provided in several works mentioned previously in section 3; however most of them are for conventional sensor networks and are centralized. In our case the system is distributed, and integrated with wide area sensor networks.

6. Although the Intel IrisNet project supports aggregate queries; the caching infrastructure is not sufficient for caching results from aggregate queries. Consequently more sophisticated caching strategy is needed. View based semantic caching can help in this respect [8]. Also sophisticated algorithms for evicting of cached data are required.

6 Conclusions

The underlying query processing architectures for both wide and conventional sensor networks are important considerations affecting their performance, scalability, and applicability. In essence, current and future capabilities of sensor networks mostly depend on the query processing architecture. In this survey report, we identified the design objectives and challenges related to query processing over sensor networks and to this end investigated the key querying techniques with their relative advantages and limitations. The analysis of these techniques has enabled us to set a plethora of future research directions along with a future query processing architecture. The emphasis is placed on the interoperability and integration issues related to conventional and wide are sensor network query processing with an aim of pervasive deployment of sensor network applications.

References

[1] S. Babu and J. Widom. Continuous queries over data streams. *ACM SIGMOD Record*, 30(3):109–120, Sep 2001.

[2] UC Berkeley. Tinydb. http://telegraph.cs.berkeley.edu/tinydb/.

[3] P. Bonnet, J. E. Gehrke, and P. Seshadri. Towards sensor database systems. In *Proc. of the Second Int'l Conference on Mobile Data Management*, pages 3–14, 2001.

[4] D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, G. Seidman, M. Stonebraker, N. Tatbul, and S. B. Zdonik. Monitoring streams - a new class of data management applications. In *In Proc. of 28th conference of Very Large Databases(VLDB)*, pages 215–226, Aug. 2002.

[5] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. R. Madden, V. Raman, F. Reiss, , and M. A. Shah. Telegraphcq: Continuous dataflow processing for an uncertain world. In *CIDR Data Engineering Bulletin*, volume 26(1), Mar 2003.

[6] J. Chen, D. DeWitt, F. Tian, and Y. Wang. Niagaracq: A scalable continuous query system for internet databases. *ACM SIGMOD*, pages 379–390, May 2000.

[7] A. Crespo and H. Garcia-Molina. Routing indices for peer-to-peer systems. In *Int'l Conference on Distributed Computing Systems*, pages 23–30, Jul. 2002.

[8] A. Deshpande, S. Nath, P. B. Gibbons, and S. Seshan. Cache-and-query for wide area sensor databases. *ACM SIGMOD*, pages 503–514, 2003.

[9] J. Gehrke, F. Korn, and D. Srivastava. On computing correlated aggregates over continual data streams. *ACM SIGMOD*, 30(2):13 – 24, 2001.

[10] P. B. Gibbons, B. Karp, Y. Ke, S. Nath, and S. Seshan. Irisnet: An architecture for a world-wide sensor web. *IEEE Pervasive Computing*, 2(4):22–33, Oct.-Dec. 2003.

[11] M. Harren, J. Hellerstein, R. Huebsch, B. Loo, S. Shenker, and I. Stoica. Complex queries in dht-based peer-to peer networks. In *Int'l Workshop on Peer-to-Peer Sys-*

tems ACM Revised Papers from the First Int'l Workshop on Peer-to-Peer Systems, pages 242–259, Mar. 2002.

[12] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: a scalable and robust communication paradigm for sensor networks. In *Mobile Computing and Networking*, pages 56–67, 2000.

[13] P. Kalnis, W. S. Ng, B. C. Ooi, D. Papadias, and K. L. Tan. An adaptive peer-to-peer network for distributed caching of olap results. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 25–36, 2002.

[14] S. Madden. *The Design and Evaluation of a Query Processing Architecture for Sensor Networks*. PhD thesis, UC Berkeley, Fall 2003.

[15] S. Madden and M. J. Franklin. Fjording the stream: An architecture for queries over streaming sensor data. In *18th Int'l Conf. on Data Engineering*, pages 555–566, Feb. 2002.

[16] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. Tag: a tiny aggregation service for ad-hoc sensor networks. *SIGOPS Oper. Syst. Rev.*, 36(SI):131–146, 2002.

[17] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. The design of an acquisitional query processor for sensor networks. In *Proceedings of the 2003 ACM SIGMOD international conference on on Management of data*, pages 491–502, 2003.

[18] P. V. Mockapetris and K. J. Dunlap. Development of the domain name system. In *SIGCOMM*, volume 25(1), Jan 1995.

[19] R. Sadri, C. Zaniolo, A. Zarkesh, and J. Adibi. Optimization of sequence queries in database systems. In *PODS*, pages 71–81, May 2001.

[20] M. A. Shah, J. M. Hellerstein, S. Chandrasekaran, and M. J. Franklin. Flux: An adaptive partitioning operator for continuous query systems. In *ICDE*, pages 25–32, Mar 2003.

[21] M. Wahl, T. Howes, and S. Kille. Lightweight directory access protocol. RFC, Standards Track RFC 2251, M. Wahl, Critical Angle Inc., T. Howes, Netscape Communications Corp., S. Kille, Isode Limited, Dec 1997.

[22] J. Yang and J. Widom. Incremental computation and maintenance of temporal aggregates. In *Proceedings of the 17th International Conference on Data Engineering*, pages 51–60. IEEE Computer Society, 2001.

[23] Y. Yao and J. E. Gehrke. The cougar approach to in-network query processing in sensor networks. *ACM Sigmod Record*, 31(3), Sep 2002.

[24] Y. Yao and J. E. Gehrke. Query processing in sensor networks. In *First Biennial Conference on Innovative Data Systems Research (CIDR)*, Jan. 2003.