Rethinking Data Management for Storage-centric Sensor Networks[*]

Yanlei Diao, Deepak Ganesan, Gaurav Mathur, and Prashant Shenoy
{yanlei, dganesan, gmathur, shenoy}@cs.umass.edu
Department of Computer Science
University of Massachusetts, Amherst, MA 01003

ABSTRACT

Data management in wireless sensor networks has been an area of significant research in recent years. Many existing sensor data management systems view sensor data as a continuous stream that is sensed, filtered, processed, and aggregated as it "flows" from sensors to users. We argue that technology trends in flash memories and embedded platforms call for re-thinking this architecture. We articulate a vision of a storage-centric sensor network where sensor nodes will be equipped with high-capacity and energy-efficient local flash storage. We argue that the data management infrastructure will need substantial redesign to fully exploit the presence of local storage and processing capability in order to reduce expensive communication. We then describe how StonesDB enables this vision through a number of innovative features including energy-efficient use of flash memory, multi-resolution storage and aging, query processing, and intelligent caching.

1. INTRODUCTION

Wireless sensor networks has been an area of significant research in recent years. Sensors generate data that must be processed, filtered, interpreted, and archived in order to provide a useful infrastructure to users. Sensor deployments are often untethered, and their energy resources need to be optimized to ensure long lifetime. Consequently, an important research theme in sensor networks is energy-efficient data management.

Current industrial and scientific uses of wireless sensor networks can be classified broadly along two dimensions: queries on "live" data and queries on "historical" data. In live data querying, sensor samples are useful only within a small window of time after they have been acquired. Examples include event detection queries for detecting landslides [51] or other events, or ad-hoc queries on current data (e.g.: what is the temperature now?). Querying historical data is required for applications that need to mine sensor logs to detect unusual patterns, analyze historical trends, and develop better models of particular events. A common refrain from users who need access to historical data, often for scientific applications, is that "every bit of data is potentially important" and therefore cannot be discarded.

A large class of data management systems and techniques have been proposed for querying live data. TinyDB [32], Cougar [57] and Directed Diffusion [24] provide the functionality to push-down filters for continuous queries into the network such that data processing can be performed closer to where the data is sensed and only the end result needs to be communicated. Such push-down querying saves energy for communication, thereby increasing the lifetime of wireless sensors. Query-specific filters can be used both for event detection queries (e.g.: trigger when temperature > 120 F), and for pre-processing data at sensors and transmitting high-level aggregates rather than raw data (e.g.: temperature averages over 10 minutes). Another interesting class of data management techniques for live data querying is Acquisitional Query Processing (AQP). Based on query needs, AQP techniques intelligently determine which nodes to acquire data from, which attributes to sample, and when to sample. Both TinyDB as well as BBQ [14] are acquisitional in nature — in TinyDB, sensors can determine the order in which to acquire samples to answer queries with least energy cost, and in BBQ, the base-station uses a model to determine which nodes and attributes to query to minimize energy consumption as well as answer queries with the required error and confidence bounds.

In contrast to the wealth of research on data management techniques for live data querying, there has been little work on data management solutions for querying historical sensor data. There are two models for designing such historical data querying systems. The first model treats sensor data as a continuous stream that is losslessly aggregated within the network and then transmitted and archived *outside* the sensor network. Once the data is collected, they can be stored in a traditional database, and queried using standard techniques. Such networks have commonly been referred to as "dumb data collection" sensor networks [39] since very limited intelligence can be embedded within the network. Many practical deployments of wireless sensor networks for monitoring phenomena employ the data collection model. While such a model is easy to deploy, these deployments can be short-lived when high data rate sensors (e.g: camera, acoustic, or vibration sensors) are used, since the data communication requirements overwhelm the available energy resources.

A second model for querying historical data is to view the sensor network as a database that supports archival query processing, where queries are pushed inside the network, possibly all the way to the remote sensors that archive data locally. This architecture has the potential to be considerably more energy-efficient for querying archived data, since query processing is performed at the source

[*]This research is supported in part by NSF grants EEC-0313747, CNS-0626873, CNS-0546177, CNS-052072, and CNS-0325868.

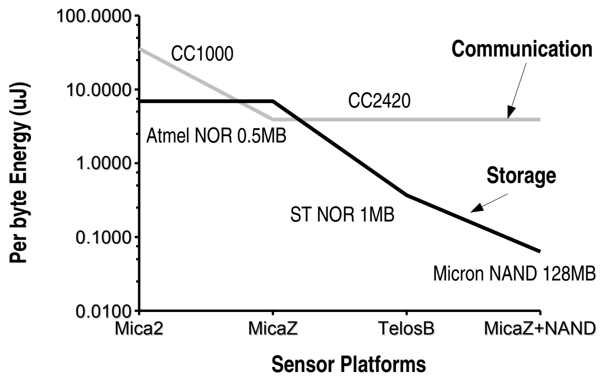

Figure 1: Energy cost of storage compared to that of communication for the popular Mica2, MicaZ and Telos sensor platforms; also included is the cost of storage using the UMass NAND flash adapter for the Mica2 / MicaZ platform. We notice that using NAND flash increases the difference between communication and storage costs to almost two orders of magnitude relative to storage.

and transmissions involve query results as opposed to data. However, the model has been considered impractical in real deployments for three reasons. First, it is commonly assumed the sensor devices have limited computational resources which preclude any complex query processing from being performed at remote sensors. Second, there exists a perception that storage capacities on sensors is limited (on the order of megabytes), thereby severely limiting the amount of data that can be archived locally (in contrast to archiving data outside the network, where storage is potentially limitless). Third, flash memories are considered to be less energy-efficient when compared to microcontrollers and low power radios used on sensor nodes, thereby reducing the energy benefits of local archival. Indeed, all three limitations are true of the popular Mica Mote platforms [12]. These nodes have a sub-10MHz processor, 4KB of RAM, and less than one megabyte of flash memory. In addition, the flash memories used on these devices are less energy-efficient than the low-power radios used on them, making transmitting data outside the network cheaper than local archival.

Technology Trends: Recent technology trends, however, make a compelling case for revisiting the argument for data collection as being the only practical solution for archival query processing. The emergence of new generation NAND flash memories have dramatically altered the capacities and energy efficiency of local flash storage. It is now possible to equip sensor devices with several gigabytes of low-power flash storage, and flash storage capacities continue to rise in accordance with Moore's law. Further, in a detailed measurement study of flash memories [34], we showed that equipping the MicaZ platform with NAND flash memory allows storage to be *two orders of magnitude cheaper* than communication and comparable in cost to computation. Figure 1 compares the per-byte energy cost of communication and storage for various sensor platforms and shows that the cost of storage has fallen logarithmically with the emergence of efficient NAND flash memories. This observation fundamentally alters the relative costs of communication versus computation and storage, making local archival far more attractive. Finally, a slew of new sensor platforms such as the iMote2 [11] and Yale XYZ [30] have become available that boast considerably greater processing capabilities than the Mica Motes at only slightly worse overall energy-efficiency and comparable prices. The iMote2 for instance is equipped with a 13 - 600 MHz PXA processor that is up to two orders of magnitude more capable than the 6MHz processor on a Mica Mote.

These trends challenge the conventional wisdom about how to architect a sensor network, and in particular, the role of storage in sensor networks. They make a compelling case for equipping sensor nodes with high-capacity energy-efficient local flash storage and redesigning algorithms to exploit cheap storage for reducing expensive communication, a tradeoff that has not been fully exploited in current system designs. In addition, they argue for designing systems that place far more query processing complexity at the sensors, since they now have the resources to perform more complex tasks.

In this paper, we present StonesDB, a novel sensor database architecture that emphasizes local data archival and query processing at embedded sensors. StonesDB[1] makes energy-efficiency its primary design goal. By exploiting flash-based in-network data storage, StonesDB represents a paradigm shift from many existing approaches that rely on streaming and long-term archival of data outside the sensor network. StonesDB is designed for performing rich query processing inside the network and supports both traditional queries as well as newer data mining style queries that are common in sensor data analysis. In addition, StonesDB is designed to exploit the hierarchical architecture of sensor networks—it places intelligence at sensor nodes, while fully exploiting the resource-rich nature of sensor proxies and gateways. Unlike existing work on in-network querying [24] that has focused on algorithms, indexing [21, 27, 42] and query forwarding [17, 44] approaches, our work focuses on the core infra-structural building blocks for designing a true in-network database.

In the remainder of this paper, we first articulate the numerous research challenges that arise in the design of StonesDB in Section 2. Section 3 presents an architectural overview of StonesDB. In Section 4, we present the local database layer of StonesDB that performs energy-efficient query processing, multi-resolution storage and data aging. In Section 5, we discuss the design of the distributed data management layer that unifies local storage and database capabilities at individual nodes into an online networked data store. We present systems closely related to StonesDB in Section 6 and then conclude with a brief status report of our ongoing implementation and directions for future work in Section 7.

2. RESEARCH CHALLENGES

This section outlines our high-level design goals and then discusses several challenges that arise in the design of StonesDB.

2.1 Design Goals

StonesDB assumes, and seeks to exploit, a two-tier architecture comprising battery-powered resource-constrained sensor nodes at the lower tier and resource-rich proxies at the higher tier. Although StonesDB is targeted towards a broad class of sensor applications, for the purposes of this paper, we choose monitoring using a camera sensor network as a representative example. Unlike simpler examples such as temperature monitoring, this application produces *rich* data in the form of images and features extracted from them, and requires handling of a broad set of queries on such image data, thereby stressing the limits of a resource-constrained sensor environment. In this application, low-power cameras sensors on tier-1 nodes monitor the environment by capturing high-resolution im-

[1]STONES is an acronym for STOrage-centric Networked Embedded Systems.

ages of their surroundings. These images are assumed to be stored locally and a low-resolution summary is sent to the proxy. Queries arriving at the proxy must be answered using a combination of data/index at the proxy and those at the sensor nodes. Our work seeks to address the following design goals:

- *Exploit local flash memory:* We wish to leverage the presence of cheap and energy-efficient flash memory as a storage substrate for StonesDB. Doing so trades storage for more expensive communication.
- *Optimize for energy-efficiency:* We seek to design a sensor database that is not only suitable for resource-constrained environments but also for highly optimized for energy-efficiency.
- *Exploit resource-rich proxies:* Despite the availability of more capable sensor platforms, they are still resource-poor when compared to proxies. Our design seeks to leverage the resources at the proxy, whenever possible, to reduce the burden on sensor nodes.
- *Support a rich set of queries:* Our design seeks to support a rich set of queries, including traditional SQL-style as well as data mining-style queries.
- *Support heterogeneity:* We seek to support multiple sensor platforms that are available today and our design exploits the resources available on each to make appropriate design choices. Thus, a StonesDB instantiation on the low-end Mica2 motes might be different from that on a more-capable iMote2.

The above design goals lend themselves to an architecture where (i) local flash-based storage is emphasized, (ii) energy-efficiency is a crucial design goal for all components such as storage, indexing and query processing, (iii) query processing is split between the proxy and the sensors, with some or all of the processing pushed to the remote sensors, (iv) different target sensor platforms result in different design choices depending on their capabilities. In the rest of this section, we discuss these research challenges in more detail.

2.2 Use of a Flash Memory Storage Substrate

As noted earlier, StonesDB exploits flash memories on sensor nodes for archiving data locally. Flash memories are vastly different from magnetic disks in their architecture, energy constraints, and read/write/erase characteristics, all of which fundamentally impact how an embedded sensor database is designed. Flash memories are page organized in comparison to the sector-based organization of magnetic disk drives. A key constraint of flash devices is that *writes are one-time* —once written, a memory location must be *reset or erased* before it may be written again. In addition, the unit of erase often spans multiple pages (termed as an erase block), thereby complicating storage management.

Table 1 shows both the device and system-level energy and latency costs involved with the read and write operations of a Toshiba 1Gb (128 MB) NAND flash[6] chip that we measured. Based on our measurements, we find that the energy cost of writing ($W(d)$) and reading ($R(d)$) d bytes of data to and from flash can be modeled as following:

$$W(d) = 24.54 + d \cdot 0.0962 \mu J \quad (1)$$
$$R(d) = 4.07 + d \cdot 0.105 \mu J \quad (2)$$

Comparison with Magnetic Disks: We find the energy cost associated with flash-based storage to be a linear function of the number of bytes written to or read from flash. However, magnetic disks have a constant power consumption associated with keeping the disk in motion (a couple of watts) which makes this an unsuitable storage medium for low-energy devices. Much like the seek overhead in magnetic disks, there is a fixed cost of accessing a page on flash, and then a per-byte overhead associated with each additional byte written to (or read from) the page. Accessing adjacent sectors on disk significantly reduces the seek overhead since the disk head does not need re-positioning. Unlike disks though, accessing adjacent pages on flash does not impact the fixed cost as this corresponds to the time during which an address is clocked in and the flash read or write operation is enabled. Once enabled, the cost of clocking data in/out of the flash chip is linearly dependent on the size of data being operated upon. Note that the cost of reading or writing n pages is n times the cost of reading or writing a single page since each page is addressed separately.

		Write	Read
NAND Flash Energy Cost	Fixed cost	13.2μJ	1.073μJ
	Cost per-byte	0.0202μJ	0.0322μJ
NAND Flash Latency	Fixed cost	238us	32us
	Cost per-byte	1.530us	1.761us
NAND Flash + CPU Energy Cost	Fixed cost	24.54μJ	4.07μJ
	Cost per-byte	0.0962μJ	0.105μJ
NAND Flash + CPU Latency	Fixed cost	274us	69us
	Cost per-byte	1.577us	1.759us

Table 1: Cost of NAND flash operations on the Mica2 sensor platform

Due to these differences, several design decisions made by traditional databases are not directly applicable to flash-based databases. For instance, disk-based storage systems often uses in-place updates to update a record or to overwrite an existing disk block. However, performing the same operation on flash would require reading the entire erase block, performing the modification, erasing the block and then writing it back. This *read-modify-erase-write* is a very energy-intensive operation, and therefore *in-place updates should be avoided whenever possible*.

The problem of avoiding in-place updates has been explored on multiple fronts in the database community. The use of shadow paging [29] avoids multiple writes within a page, though it results in serious performance drawbacks. Vagabond [36] builds a database on top of a log-structured [43] data store, using techniques such as delta-chains to perform updates on the database. There is also ongoing work to support flash-based databases in industry and research groups—examples include FUEL [25], DELite [55] Polyhedra Flashlite [18], Birdstep RDM [8] and eXtremeDB [37]. A number of flash-based sensor data storage systems have been built as well [13, 20, 54].

StonesDB differs from all of these efforts by focusing on energy-efficient use of the flash memory storage substrate for a sensor database. This necessitates the re-design of a multitude of database components – organization of data and indices on flash, buffering strategies, and data structures that store and update data.

2.3 Optimize for Energy-efficiency

A traditional database system chooses its index structures and query processing strategies to optimize for response time, whereas a sensor database needs to optimize for low *energy consumption*. This change has significant implications on the design of a sensor database. For instance, traditional databases almost always construct indices on data to improve query processing performance, since sequential scan of data incurs high latency. However, in a sensor database, there is a tradeoff between the cost of index construction and the benefit offered by it. To illustrate, consider a B+ tree construction on a stream of sensor readings. A standard B+ tree construction algorithm for magnetic disks [41] would build the tree dynamically as readings are appended to the archived stream. However, since flash memory pages can not be overwritten without an erase operation, insertions into the B+ tree are very expensive. This cost is worth-while only if the number of reads due to queries is sufficiently high. For data that is infrequently queried, it may be better to skip the index construction altogether and use linear scans of the data instead. Even when an index is desirable, the index structure should be organized on flash so that index updates incur as few erase operations as possible.

Thus, in an energy-optimized sensor database, the query workload including the types of queries and the frequency of their execution will dictate the relative cost-benefit tradeoff of index construction as well as the types of indices that are maintained. In addition, traditional algorithms for index construction over magnetic disks need to be re-designed to ensure energy-efficiency implementations in a flash-based database system.

2.4 Handle Finite Storage

In traditional databases, incremental growth over the system lifetime is handled by periodic system upgrades; database administrators deal with storage capacity exhaustion either by increasing storage capacity or moving data to tertiary storage. In contrast, wireless sensor networks are often designed to operate for many years without human intervention. When these sensors are used to store rich sensor data such as images or acoustic streams, the storage capacity may be insufficient to store all data losslessly throughout the lifetime of the sensor. In such instances, sensors will need to deal with storage capacity exhaustion by intelligently “aging out” part of the archived data to make room for new data. There has been little research on performing data aging in a database. The closest work is the vacuum cleaning technique proposed in Postgres [49] to “vacuum” old data from secondary to tertiary storage.

Data aging in sensor database raises a number of challenges. First, rather than naively aging the oldest data, strategies that age out the least recently used or the least valuable data must be designed. Second, rather than simply discarding data (and losing it completely), it may be better to generate a coarser grained representation of the data and age out the raw data; data can be progressively aged as its use or value diminishes. Data aging reduces query precision since queries on coarser grained data can be answered with less precision than that on the original data. While aging and multi-resolution storage have been considered in Dimensions [19], the tradeoff with query precision hasn’t been addressed. Thus, an important goal of StonesDB is to resolve the tension between long-term storage and loss of query precision.

2.5 Support Rich Querying Capability

StonesDB is designed to support a wide range of queries over a storage-centric sensor network, that is, over the *historical data* stored across the sensor network. These queries can be broadly classified into two families, discussed as follows.

The first family of queries consists of SQL-style queries that can involve equality and range predicates over value and/or time, and additionally a variety of spatial and temporal aggregates. There has been considerable work on in-network processing of queries in this family. The prior work, however, mostly focuses on *live data* or *recent data* that is specified by a small sliding window. In this context, energy-efficient query processing strategies have been proposed to handle equality and range predicates [5, 14, 53], simple aggregates [4, 31, 47, 53] including min, max, count, and avg, complex aggregates [22, 46] including median and top-k, and user-defined aggregates such as contour maps and target tracks [23].

An equally important family of queries that have received considerably less attention are data mining queries to perform signal processing, pattern identification, time-series and spatio-temporal data analysis. These queries usually involve processing a large amount of data. Support for these types of queries is necessary for post-facto analysis of historical data across the spectrum of sensor network applications. We now discuss the new query types that we seek to support in StonesDB.

- ***Time-series Analysis Queries***: Time-series analysis queries are typically interested in detecting trends or anomalies in archived streams. Such queries can specify ascending, descending, spike, or non-continuous change patterns. This class of queries are particularly useful in anomaly detection applications such as fire monitoring and machine failure monitoring. An example query would be “determine the parameters of an ARIMA model that best captures last 5 days of temperature data”.

- ***Similarity Search Queries***: In this class of queries, a user is interested in determining whether data similar to a given pattern has been observed in archived data. Similarity queries are important for event detection applications such as habitat monitoring, earthquake monitoring, and camera surveillance. An example query would be “was a vehicle with license number $24VK02$ detected last week” in a vehicle monitoring application using camera sensors. Similarly, in a habitat monitoring network with acoustic sensors, a query could be “was a bird matching acoustic signature S detected in the last month”.

- ***Classification Queries***: Target classification queries are related to similarity search queries, but go further in requiring StonesDB to classify a given signal into a type of event. For instance in [28], acoustic and seismic signals of vehicles are used to determine which was the most likely vehicle that was observed. Such classification queries use techniques such as maximum likelihood, support-vector machines or nearest neighbor to determine the best match corresponding to observed data. An example query is “determine the types of vehicles that were detected last week”.

- ***Signal Processing Queries***: Many operations on sensor data involve signal signal processing tasks such as FFT, wavelet transform, and filtering. For instance, in a structural monitoring application of buildings, a typical query is “find the mode of vibration of the building” [52]. Such a query typically involves using an FFT or spectrogram on the raw time-series data of building vibrations to extract the frequency components of the signal followed by determining the mode.

Our goal is to offer *energy-efficient support for both families of queries over historical data across the sensor network.* Toward this goal, several challenges need to be addressed. Handling such a broad spectrum of queries requires a query language that is rich enough for them. A sufficient language may need to integrate SQL extensions proposed for sequence databases [45] and new constructs for specifying non-traditional data mining queries. In addition, processing the variety of queries over large amounts of data across the sensor network poses significant challenges in the design of a local sensor database, e.g., access methods and query processing techniques, as well as a distributed database, e.g., query planning and optimization across the network. Finally, depending on the application, these queries may be executed one-time or periodically (a simple form of continuous queries). Efficient support for a mixture of these execution modes is another issue to address.

2.6 Support Heterogeneous Sensor Platforms

A plethora of embedded sensor platforms are available today ranging from very constrained, low-power sensors to more powerful, PDA-class platforms. At the lowest end of sensor platforms are highly constrained *mote-class* devices like Mica2 [12] and Telos [38] equipped with 4-10KB of RAM and 8 bit processors. Some examples of intermediate-class sensor platforms include the Intel iMote2 [11] and Yale's XYZ [30] with 32 bit processors and many megabytes of RAM. At the high end of the spectrum are larger *micro-servers* platforms such as Stargates [48] that have more powerful radios, processors and more RAM. The flash memory storage substrate on these devices can differ as well since both NAND and NOR flash memories may be used. A typical sensor deployment could comprise one or more platforms discussed here.

A key goal of StonesDB is to be configurable to a heterogeneous set of sensor platforms while providing functionality proportional to the resources available on a platform. Such configurability will often necessitate significant changes to the underlying design. For instance, consider the design of StonesDB for the Mote platform and the iMote2 platform. These two platforms differ widely in their memory capabilities — the Mote has under 10K of RAM whereas the iMote2 has 32MB of RAM. A typical flash memory block is of size 32-64KB, which is too large to buffer on the Mote but small in comparison to the memory on the iMote2. Thus, the low-level storage sub-system for the Mote platform needs to be severely restricted in how it organizes data across erase blocks, whereas the iMote2 can use a considerably more sophisticated design.

2.7 Support Distributed Architectures

Our discussion thus far has focused on query processing at a single sensor node. However, sensor deployments are often distributed, consisting of multiple remote sensor nodes that are wirelessly connected to sensor proxies or gateways. Queries on archived data first arrive at the proxy and are then forwarded to one or more sensor nodes for processing.

A number of challenges in distributed sensor data management have been addressed in prior work including handling packet-losses [31], handling uncertain data [14, 15], in-network data aggregation [10], optimal data gathering tours [35], and others. In this paper, we focus on a problem that is specific to a storage-centric sensor network architecture — *proxy caching*. Since proxies are typically far more resource-rich than the remote sensor nodes, an energy-efficient sensor data management system should leverage the proxy resources whenever possible to reduce resource consumption at battery-powered sensor nodes. One possible technique for doing so is to maintain a proxy cache containing data and query results. Caching any retrieved data as well as query results at a proxy has a number of benefits. If a subsequent query can be answered using cached values, it can substantially reduce query response times while saving precious sensor resources. Proxy caching also improves overall data availability and persistence, since it effectively replicates archived data at the proxy. However, proxy caching comes at high cost since sensors expend energy in communicating data to the proxy. Thus, an important goal of StonesDB is to balance the energy cost of caching with the benefits obtained with it.

3. ARCHITECTURE

StonesDB employs a two layer database stack that maps onto the two tier architecture of a sensor network as shown in Figure 2. As indicated earlier, our network architecture comprises of a lower tier of battery-powered resource-constrained sensors nodes (*e.g.*, Motes, iMotes) and an upper tier of resource-rich proxies. While sensor nodes are assumed to be equipped with large flash storage, their computation and communication capabilities are more constrained. For instance, Mote-class devices have 4KB RAM and a 10MHz processor while an iMotes have 32MB RAM and more capable Intel PXA processor. For our representative camera sensor network application, such nodes are assumed to be equipped with low-power imaging sensors (e.g. Cyclops [61]) for image capture and image processing. The proxy tier is assumed to be tethered and resource-rich.

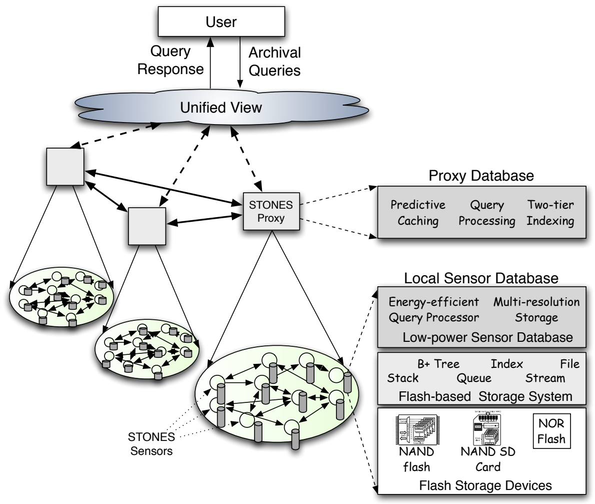

Figure 2: StonesDB architecture for a two-tier sensor network comprising proxies and sensors. The sensor tier runs an ultra low-power database stack that performs flash-optimized energy-efficient archival and query processing and multi-resolution data aging. The proxy tier performs intelligent caching and indexing, and determines how to handle queries with minimal energy cost.

System Architecture: The two-layer StonesDB stack comprises of a local database that runs on each sensor node and a distributed data management layer that runs on the proxy and interacts with the local database layer. The local database in StonesDB has three key components, (a) a query engine that generates energy-efficient query plans for executing queries and presents query results with confidence values, (b) data summarization and aging algorithms that enable multi-resolution summaries of data for efficient query processing and for space-saving storage of old data given flash memory constraints, (c) an energy-efficient storage substrate that offers partitioned storage and indexing to facilitate query processing and to simplify aging of data. The instantiations of these three components depends on the capabilities of the nodes. For instance, a more resource-constrained Mica2 Mote will run a minimalist version that supports a simple declarative query interface and a storage substrate that supports simple data aging techniques [33]. A more capable node such as the iMote will support richer set of queries and more sophisticated storage and indexing techniques.

The distributed data management layer at the proxy comprises of two key components. First, it employs a cache that contains summaries of data observed at lower-tier nodes (e.g., low-resolution images). Any data fetched from the sensors for query processing is also stored in the cache. Second, it employs a query processing engine that determines how to handle each incoming query. Queries can be processed locally by using cached data or fetching more data from the nodes, or they can pushed to the sensor nodes after some initial processing.

System Operation: We describe the operation of StonesDB using the example of a camera sensor network. In a storage-centric camera sensor network, the camera sensor nodes store high fidelity raw images and transmit metadata to the proxy. The metadata can include a low resolution image of frames where motion was detected, features extracted from images such as the number of objects or size of objects, coordinates describing the field of view, average luminance, and motion values, in addition to basic information such as time and sensor location. Depending on the application, this metadata may be two or three orders of magnitude smaller than the data itself, for instance if the metadata consists of features extracted from image.

We now consider how ad-hoc queries can be handled in such a tiered storage-centric camera sensor network. A user can pose queries over such a network using a declarative querying interface, perhaps with a confidence bound that specifies the desired quality of response. Consider a search query on a camera sensor network where the user is searching for regions of the network where a particular type of object (say a face) is detected. Here, we assume that the specific object that the user is looking for may not be known in advance, hence, the data stored by the network is used to search for new patterns in a post-facto manner.

The query is first routed to the sensor proxy which attempts to answer the query using the summaries and metadata that it has obtained from the sensors. If metadata includes a low-resolution image, the proxy can process the query on the summary to get an approximate answer for the query. If the query can be satisfied with the data cached at the proxy, it provides an immediate response to the query. If the quality of the response is not sufficient to meet the query needs or if the data relevant to the query is not present in the cache, the proxy determines the subset of sensors that are most likely to answer the query and can forward the query to these sensors.

The proxy is presented with a number of options in terms of how to query the sensor tier. One option is to pull relevant data from appropriate sensors and perform the query processing at the proxy. This option might be preferable if the query involves considerable image processing that requires more resources than is available at the sensors, for instance, if the lower tier comprises resource-constrained Motes. A second option is to push the entire query to the relevant sensors and let the local databases at the sensors handle the query on locally stored data. The proxy can then merge results from the databases at individual sensors to and provide a response to the user. This places greater querying complexity at the sensor tier and might only be feasible when more powerful sensor nodes are used. However, since computation and storage are far less expensive than communication, such in-network querying is considerably more energy-efficient. A third option is for the proxy to partially process the query on the summaries that it has stored, and send the query as well as partial results obtained by executing the query on the proxy cache to the sensor. This can potentially reduce the computation requirement and storage accesses at the sensor since the local database will only need to refine the results of the partial query.

4. LOCAL DATABASE

An important service in a data management stack is an ultra low-power embedded sensor database that enables users to query the archived sensor data. More specifically, the sensor database offers a service where (1) readings obtained from a specific sensor over its entire lifetime are viewed as an archived stream arranged in order of time, (2) high-level declarative queries can be posed against one or multiple archived streams, and (3) query answers are presented with confidence values. The architecture of a local database, shown in Figure 3, was sketched in the previous section. We now discuss key issues in the design of these components.

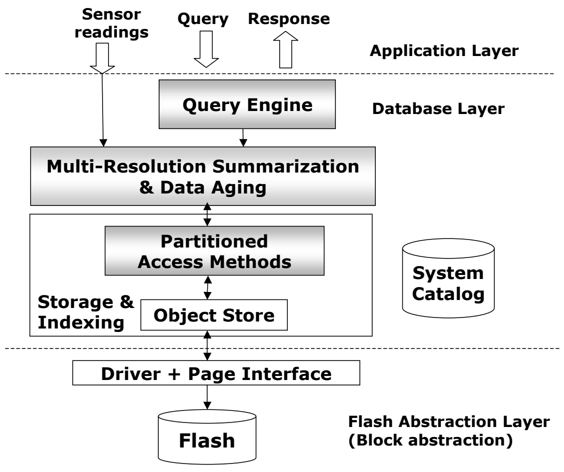

Figure 3: Architecture of the local database

4.1 Storage and Indexing

The storage and indexing module stores sensor data onto flash and provides a set of access methods for uses in query processing. As shown in Figure 3, this module has two components: the lower component, *Object Store*, offers implementations of basic data structures on flash, including streams, queues, stacks, and simple search trees [33]. The discussion below focuses on the upper component, *Partitioned Access Methods*, that builds on the object store to provide access methods for query processing.

In our design, readings obtained from a specific sensor can be stored and accessed in a number of ways.

- **Stream**. In the basic approach, data is organized as a contiguous sequence of readings in order of sensing time, and can be accessed only through a sequence scan. Thus, an archived stream in our system is analogous to a heap file in traditional database systems.

- **Index**: In the second approach, index structures are built on top the stream, based on one or multiple attributes of the readings. Such indices will provide efficient support for equality and range predicates on the matching attributes, eliminating the need to scan the entire stream. Standard indices, such as hash indices, B-trees, and R-trees, will be equally useful in our system.

- **Summary**: Efficient handling of non-traditional data mining queries requires a different strategy. Typical queries in this family, such as time series analysis and signal processing queries, require a sequential scan of the data of interest. Standard indices for equality or range predicates are not so helpful to them. Processing of certain queries, however, can be made more efficient by exploiting a lower-resolution summary of the data— the coarser grained representation can be first scanned quickly to eliminate subsets that are not of interest and identify parts that are likely to match; the full resolution versions of only these subsets are then scanned for further processing. Summary-based access methods as such can substantially reduce the energy costs of full scans.

Costs and Benefits of Access Methods. A fundamental difference between a sensor database and a traditional database is that in the former system, auxiliary access methods such as indices and summaries consume energy in their construction, so their benefits in query processing come at a cost. Since the tradeoff between benefits and costs of access methods is a significant issue, it challenges the conventional wisdom which indicates it is almost always beneficial to create indices for read-only query workloads. Consider a H-level B+ tree stored on flash memory. The complete cost for a single B+ tree insertion is H page reads and page writes, *i.e.* $H(C_r + C_w)$, where C_r and C_w are the per page read cost and write cost, respectively. Alternately, if index construction is avoided altogether, and a sequential scan of the stream is needed to process each query, which costs C_r/R_{page}, where R_{page} is the number of readings stored in one page. A back-of-the-envelope calculation using flash read and write costs [34] and a B+ tree of depth two reveals that performing a sequential scan is 340 times more energy efficient than building a B+ tree for it! Thus, the benefit of an index offsets the high construction cost only when the data is accessed very frequently. Otherwise it is more energy efficient to execute the query by resorting to sequential scans.

For this reason, StonesDB supports lazy index construction. By default, index construction is disabled; it is triggered dynamically when it is deemed beneficial. This decision can be made based on the knowledge of the current query workload, e.g., the presence of periodic queries that need to repeatedly scan overlapping regions of the stream, or the presence of multiple queries that overlap in their search ranges. It can also be made based on the statistics collected, e.g., the frequency of scanning the data in the recent past. Additionally, indices can be independently maintained for different partitions of the stream, which is explained more below.

Energy Efficient Construction and Maintenance of Access Methods. Another issue pertaining to the storage system in a sensor database is how to efficiently construct and maintain access methods over flash memory, once a decision is made to built them. As stated in Section 2.2, read, write as well as erase operations on flash memory consume energy; hence, all these operations need to be minimized to achieve an energy-optimized database. The storage subsystem of StonesDB achieves energy-efficiency using two techniques.

Partitioned Access Methods. The first technique that StonesDB uses to optimize energy is to create partitioned access methods. More specifically, each stream in the flash memory store is organized into temporal segments, called partitions; auxiliary access methods such as indices and summaries are built for each partition. Under this approach, data and its indices are bundled together in partitions; if the data needs to be deleted, its indices can be easily located and pruned together with the data. Otherwise, one has to search a large index created for the entire stream to prune index entries pointing to the deleted data. The storage system can choose to create *logical* or *physical* partitions. Logical partitions are simply defined by the temporal ranges that they cover, and linked with the relevant data and indices. In this scheme, however, deleting data and indices may incur separate erase operations, if they belong to different erase blocks. Physical partitions can provide an additional benefit. In this scheme, the flash memory store is physically organized into partitions that are aligned with flash erase block boundaries. Each partition is a container comprising data and associated indices and is completely self-contained; if deleted, the data and its index are both discarded in one erase operation.

Figure 4 shows how partitioned access methods are performed in StonesDB. For Stream 1, based on the application specification or internal decision, the database creates two indices for this stream, e.g., a B+Tree and an R-Tree. Streams are broken into physical partitions, with only the most recent partition under construction, e.g., the partition labeled as "Data1" in this example. A new set of indices are created for every new partition that is written. Each partition co-locates a segment of the stream together with its associated indices so that they can be pruned together later.

Write-Once Indexing. The presence of partitions enables the second technique that StonesDB uses for energy optimization, which we call write-once indexing. Our goal here is to ensure that indices generated on the data stream are written only once to flash and are not updated once written. This design principle aims to eliminate read, write and erase overhead that is incurred when a flash memory index is updated as described in Section 2.3. Clearly, if one wants to create a B+tree for an entire stream, it will be very challenging (if possible) to avoid updates of existing index pages. Partitioning data into smaller temporal ranges and creating a separate index per partition raises the possibility of devising algorithms to achieve write-once indexing.

A simple idea is that given a reasonable amount of memory, we might be able to hold the entire B+tree in memory during its construction for a partition and write it once to flash. The smaller the partition is, the more likely the index fits in memory. A small value for the partition size, however, may result in poor performance in query processing: given a range query, we may have to search in many partitions overlapping with the query-specified search range, which results in higher accumulated overhead in traversing those indices. How to find an appropriate partition size to strike the balance between the index construction cost and query processing cost is a research issue that we will explore. In memory-constrained environments, e.g., platforms whose available memory is smaller than or close to the size of an erase block, holding an index for a partition is impossible. In such cases, more advanced techniques are needed to ensure no (or very few) updates of existing index entries.

4.2 Summarization and Aging

The second main component of StonesDB addresses the summarization and aging of data. Before we delve into details, it is worth noting the difference between these two: while summarization can be used during aging, it is applicable in a broader set of contexts, as explained below.

Multi-Resolution Summarization. StonesDB provides the option of storing coarse-grained versions of data with varying granularity. This scheme, which we refer to as multi-resolution storage, serves three purposes: (1) coarse-grained summaries may allow more efficient processing of certain types of queries, as discussed above; (2) such summaries can be transmitted to proxies to facilitate query optimization there, as noted in Section 3; and (3) they can be used to retain useful information about old sensor data while aging the raw data to create space for new data.

Figure 4 shows an example of how the multi-resolution summarization component handles a stream of images from a camera sensor. The multi-resolution component takes the raw stream as input and generates two summary streams — a wavelet-based summary stream that captures key features of the images, and a sub-sampled stream that retains every tenth image.

We focus our discussion below on summarization for aging. We utilize multi-resolution storage to capture the key characteristics of the data stream using coarse-grained summaries that use substantially less storage space. These summaries can be used to respond to queries on older data that has been deleted, though with lesser confidence. Since summaries occupy less storage, they can be re-

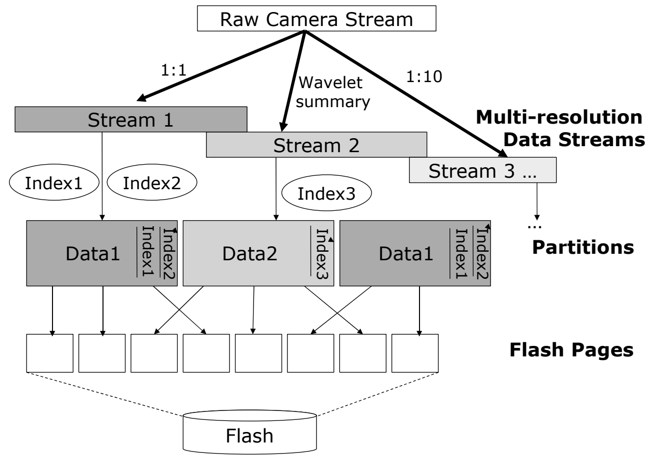

Figure 4: Data Organization in the Database

tained for significantly longer periods than raw data.

A key research contribution of our work in addressing the question of *what summaries to construct*. The summarization algorithm selected depends on the types of queries that are posed on the data, user input such as precision requirements of queries over old data, and the type of sensor data being stored. For instance, weather data has temporal patterns, hence the summary might be a model that captures the patterns in the data, and major deviations from this pattern. In other instances such as habitat monitoring, the summary can be histograms of events that have been observed in the network. Techniques to generate summaries from raw data are another issue. One approach that we will explore is to leverage a wealth of research from the data mining community, including non-sampling based approaches such as wavelet summaries [9] and histograms [40]; and sampling-based techniques such as AQUA [7], and adapt and compare them for energy-efficient implementation over flash.

Data Aging. As the flash starts filling up, some data needs to be discarded to make room for future data requiring some partitions to be *aged*. When the database determines that a partition worth of data needs to be discarded, the aging component looks through all the partitions, assigning each partition an *erase age*. The erase age of a partition does not depend on the write timestamp of the partition alone – other factors need to be taken into account such as the importance of the data in the partition (*e.g.* some partitions may hold certain event of interest) and the importance of the stream itself (*e.g.* a summary stream is more important than a raw data stream). Once all partitions have been assigned an erase age, the "least valuable" partition can be discarded.

The primary challenge in aging sensor data is determining *what to age*. StonesDB determines the erase age of data based on application aging policies and priorities. In our current design, the application provides its utility specification for queries – for example, an application can specify that it needs raw data for 30 days and then summaries that provides 95% confidence response for 90 days. This specification can be used to determine the erase age of different partitions and age data accordingly.

4.3 Query Engine

The third component of StonesDB is the query engine. As argued earlier, unlike traditional query engines that are optimized for response time, the StonesDB query engine is optimized for energy efficiency. This involves determining the most energy-efficient plan for answering a query. Overall, query processing and optimization in a sensor database is a remarkable challenge that we have just started to explore. We next highlight two research issues.

Query Optimization: A Simplified View. The conceptual model for the query optimizer is rather intuitive: first identify a set of feasible plans, and choose the most energy efficient one from them. Typical sensor applications are unlikely to issue complex queries such as those in decision support systems. A traditional query plan for a sensor database query may include a relatively simple tree of operators, for example, involving a join, an aggregate function or a pattern detection operator on top of the necessary access methods. This view is sufficient in the absence of data aging. What remains to be done is to build a cost model that captures both flash read cost and CPU cost for available implementations of each operator in the query plan. Such a cost model can be derived from the measurements given in Table 1 and the expected numbers of read operations from flash and those of CPU computation. However, when data aging also needs to be supported, the query processing strategy changes and we discuss the implications below.

Probabilistic Query Processing. Data aging poses a unique challenge for query processing and optimization. Recall that due to finite storage, some older data that a query requires may have been aged and summarized. In such cases, no single access method may be able to retrieve all data for answering the query. Therefore, the query needs to be split into a number of subqueries, for each of which there is at least one access method that covers the required data. Extra operations may be required to merge the results of the subqueries and derive the final result. A first related issue is that sophisticated metadata needs to be maintained so that the query optimizer is able to find a covering set of access methods. A second issue relates to the merging of results of subqueries, especially in scenarios where some results are generated from raw data while others are from summaries. A third issue concerns the possibility of splitting a query into subqueries. It is well-known in the literature that aggregation functions such as *medium* and *percentile* cannot be divided into subqueries with guaranteed correct answers. The last two issues imply that the query answers will be probabilistic in nature. While probabilistic query processing has recently gained research attention [3, 2, 1], how to perform it in a energy-constrained environment remains a new world to explore.

5. DISTRIBUTED DATA MANAGEMENT

The *distributed data management layer* in StonesDB unifies local storage and database capabilities at individual sensor nodes into a networked sensor database. StonesDB provides the user with the abstraction of a centralized database over a distributed sensor network, and transparently determines what data to cache at the proxy, how to execute queries on cached data, and how to efficiently query the sensor tier. In this section, we provide a brief glimpse of to key questions that we are addressing that relates to distributed querying of storage-centric sensor networks:

- *What summaries to cache at a proxy to efficiently locate sensors that have data relevant to a particular query?*
- *How should a query plan be split between the proxy and the sensors?*

Of these two problems, the first bears similarity to caching techniques used in traditional distributed databases and web caching, however, there are differences due to the energy-constrained nature of the network and sensor data characteristics. To address the second problem, we identify unique opportunities in the context of storage-centric sensor networks, which, to the best of our knowledge, have not been explored in other work in the literature.

5.1 Querying the Proxy Cache

Proxy caching in storage-centric sensor networks differs in a number of ways from traditional caching techniques in databases and networked systems. First, while traditional caches store a frequently accessed subset of the data, the sensor proxies cache *summaries* of the data to enable efficient search of remotely stored sensor data. Second, traditional caching techniques are designed to optimize performance objectives such as latency and bandwidth, whereas proxy caching in sensor networks needs to minimize the total energy used to transmit summaries from sensors to the proxy, to forward queries from the proxy to sensors, to execute the queries on locally stored data at the sensors, and to transmit the results of the query back to the proxy. In this section, we discuss two problems in proxy caching in storage-centric sensor networks: *what summaries to cache* and *what resolution of summaries to cache*.

What summaries to cache: The decision of what summaries to cache depends on the types of queries posed on the data. For instance, certain queries will simply retrieve a subset of the archived data (e.g., retrieve all tuples where temperature exceeds 80 F) whereas others will compute a function over a set of observations (e.g., max temperature over the past day). StonesDB seeks to provide a family of summaries suitable for sensor data caching. For example, spatio-temporal data models that have been proposed for acquisitional query processing [14], and in our recent work on model-driven push [26] can be adapted to be used for proxy caching and data retrieval. Here, a statistical model of sensor data (e.g.: ARIMA model) is maintained at the proxy and raw data is stored at the sensors. The proxy uses the cached data to answer queries on past data, but if the query cannot be answered with the required confidence interval, the query is pushed to the sensors which can process the query on locally archived data. Besides statistical models, the data that is cached at the proxy could be just lossy summaries of archived data at a sensor such as a low-resolution summary or metadata of images that were observed in a camera sensor network (e.g.: TSAR [16]).

What resolution of summaries to cache: This question is relevant to search-based sensor networks where the sensors transmit low-resolution summaries of the data that they sense to the proxy. More precise summaries of data at the proxy can enable the proxy to answer a greater fraction of queries on its own without needing to forward queries to the sensors. However, transmitting more precise summaries incurs higher energy overhead. One of the tradeoffs that such a storage-centric query execution framework presents is balancing the energy cost of updates with the overhead of false positives. Transmission of more coarse-grained image summaries to the proxy incurs less energy overhead but increases the fraction of false positives. Alternately, transmission of more fine-grained summaries incurs greater energy overhead but reduces false positives and hence query overhead. StonesDB seeks to adaptively balance the cost of transmitting summaries together with the cost of false positives to minimize energy consumption [16].

5.2 Querying the Sensor Tier

Upon a cache miss, the proxy queries the appropriate sensors provide an accurate query response while simultaneously minimizing the energy cost incurred by sensors for communication. An interesting problem that is unique to storage-centric query processing is splitting the query processing between the proxy and the remote sensors. Several possibilities arise for such split query processing. First, since the proxy has a coarse-grain summary of the data, it can use this summary and/or an index of the data to prune the set of sensor nodes that need to be queried (instead of flooding the query). For instance, if a query to a camera sensor network asks for all instances of a truck seen over the past hour, the low-resolution summary can be used to eliminate all nodes that didn't see any objects in this time period. Assuming the low-resolution image summaries only indicate whether an object was seen but not its type, only those nodes that detected an object in this time period need to be queried.

A more interesting possibility is to partially process the query at the proxy and refine the result at the sensor node; this reduces the query processing burden at the sensor nodes and saves energy. Consider a query that requests the number of trucks seen by a camera sensor over the past hour. Suppose that the proxy cache contains data from the node for the first half hour. This data can be employed to partially process the query at the proxy and determine the number of trucks seen for the first half hour. The sensor node can refine this partial result by processing the query for the remaining period. Another possibility is to use the coarse-grain summaries to process a query and produce a result with a certain confidence bound; if the confidence bound does not satisfy the query error tolerance, the partial results can be sent to the sensor node for further refinement until a result with sufficient confidence is produced. These examples of split query processing illustrate how a resources at the proxy can be leveraged to reduce the amount of data retrieved and processed at the sensor node, thereby yielding an energy-efficient design.

6. RELATED WORK

Related work can broadly be classified into the following categories – work in the sensor network domain towards treating the sensor network as a database and work in the database community towards building databases on resource-constrained platforms. This section discusses only *complete* systems related to our work—Cougar, TinyDB and BBQ fall in the former category and we discuss these first, while PicoDBMS and DELite fall in the latter category. Other related work has been discussed in the relevant sections where it has been referenced.

The approach adopted in Cougar [57] is to treat the sensor network as a distributed database where data collection is performed using declarative queries, allowing the user to focus on the data itself, rather than data collection. Given a user query, a central query optimizer generates an efficient query plan aimed at minimizing resource usage within the network. The sensors sense data and then transmit data matching some criteria to the base-station. The amount of data transferred is further minimized by doing some level of in-network processing on the data as it is being transferred to the base.

TinyDB [32] uses an acquisitional query processing approach where data is requested from sensors depending on the specific query posed to the network. It allows user queries to be posed using a database query language that allows both data collection and aggregation. Given a user query, TinyDB generates data filters that it then distributes and pushes onto individual sensor nodes. The sensors send data matching the filter, which is then aggregated within the network on its way to the base-station. Both these techniques minimize resource usage and hence are energy-efficient for the sensor network.

BBQ [14] improves over TinyDB by constructing data models of the sensed data using statistical modeling techniques. The data model and live sensor data are both used to respond to queries. While this approach does introduce approximations with probabilistic confidences, it allows significant energy and time savings over the TinyDB approach. Queries that require low confidence bounds can be answered at the base station itself with the help of the data model, while queries with high confidence requirements might require acquisition of some data from the sensor.

A multitude of techniques have been explored to generate data summaries which could be used to generate data models and approximate responses to queries at the base station. Non-sampling based approaches include the use of wavelet summaries [9] and histograms [40], while examples of other sampling based approaches include AQUA [7]. [60] uses small sketches to approximate data aggregation within a sensor network.

PicoDBMS [56] is a complete database platform targeted at smart-card platforms. Like sensor platforms, smart-card platforms are highly resource constrained, however, unlike sensors node, they are not energy constrained, since they depend on external energy sources such as that of the card reader. The PicoDBMS design is specifically targeted at minimizing the write operations to EEPROM as these are time-consuming and reduce the performance of the system. The paper proposes a novel storage model that indexes data while it is being written, reducing write costs. PicoDBMS also handles complex query plans with minimal RAM consumption while supporting select, project, join and aggregate queries. A key difference is that, unlike sensor networks, energy optimizations are not a major design goal in PicoDBMS.

The DELite [55] project aims at constructing a relational database on PDA-class platforms, which have significantly more available resources (processor, memory and available power) in comparison to the smart-card or sensor platforms. The goal of the project is to support complex local queries as well as efficient database synchronization with a central database. The project uses flash memory for storage and focuses on constructing efficient query execution plans to efficiently execute complex queries. [58] and [59] aim at constructing efficient B-trees and R-trees respectively on NAND flash storage media. Both these approaches are also targeted at PDA-class platforms as they employ out-of-place data modification techniques that require a substantial amount of memory, which is unavailable on typical sensor platforms.

7. CURRENT STATUS AND CONCLUSIONS

In this paper, we argued that new technology developments in flash memories and sensor platforms have enabled energy-efficient storage and rich query processing on senor nodes and argue for revisiting the sensor network as a database architecture. We presented *StonesDB*, a sensor network database architecture that we are designing to exploit these trends. Our recent research has addressed several issues that arise in the design of StonesDB, although much remains to be done. We recently developed *Capsule* [33], a flash-based object store that provides energy-efficient implementations of objects such as linked lists, arrays, streams and trees. We are currently enhancing Capsule to handle the needs of StonesDB, such as multi-resolution summarization, aging and partitioned indexing. We have also investigated hierarchical data management in sensor networks in the context of *TSAR* [16] and *PRESTO* [26]. TSAR envisions separation of data from meta-data by emphasizing local archival at the sensors and distributed indexing at the proxies. At the proxy tier, TSAR employs a novel multi-resolution ordered distributed index structure, the Interval Skip Graph, for efficiently supporting spatio-temporal and value queries. PRESTO implements an initial version of our proxy cache—the cache is used to answer queries while error tolerances can be met, else the queries are forwarded to sensors. PRESTO also proposed a novel model-driven push-based data acquisition technique where the proxy builds a model of sensor data and transmits it to the sensor. The sensor checks the model against ground-truth and transmits only the deviations. Our ongoing work focuses on the challenges that arise in the design of the local database, including support for rich, energy-efficient query processing, multi-resolution storage and aging. We are also designing a distributed layer that splits query processing between the proxy and remote sensors.

8. REFERENCES

[1] N. Dalvi and D. Suciu Efficient Query Evaluation on Probabilistic Databases. In *VLDB 2004*.

[2] O. Benjelloun, A. D. Sarma, A. Halevy, and J. Widom. ULDBs: Databases with Uncertainty and Lineage. In *VLDB 2006*.

[3] J. Widom Trio: A System for Integrated Management of Data, Accuracy, and Lineage. In *CIDR 2005*.

[4] J. Considine, F. Li, J. Kollios, and J.W. Byers Approximate Aggregation Techniques for Sensor Databases. In *ICDE 2004*, pgs 449-460.

[5] S. Madden, M.J. Franklin, J.M. Hellerstein, and W. Hong The Design of an Acquisitional Query Processor For Sensor Networks. In *SIGMOD 2003*, pgs 491–502.

[6] Toshiba America Electronic Components, Inc. (TAEC), `www.toshiba.com/taec`. *Datasheet: TC58DVG02A1FT00*, Jan 2003.

[7] S. Acharya, P. B. Gibbons, V. Poosala, and S. Ramaswamy. The Aqua approximate query answering system. In *SIGMOD 1999*, pgs 574–576.

[8] Birdstep. Birdstep RDM Mobile 4.0. `http://www.birdstep.com/database/rdmm/v4.0/whats_new.php3`.

[9] K. Chakrabarti, M. Garofalakis, R. Rastogi, and K. Shim. Approximate query processing using wavelets. *VLDB Journal*, 10(2–3):199–223, 2001.

[10] J. Considine, F. Li, G. Kollios, and J. Byers. Approximate aggregation techniques for sensor databases. In *ICDE*, April 2004.

[11] Intel Corporation. Intel Mote 2. `http://www.intel.com/research/downloads/imote_overview.pdf`.

[12] Micaz sensor platform. `www.xbow.com/Products/Wireless_Sensor_Networks.htm`.

[13] H. Dai, M. Neufeld, and R. Han. ELF: An efficient log-structured flash file system for micro sensor nodes. In *SenSys 2004*, pages 176–187, New York, NY.

[14] A. Deshpande, C. Guestrin, S. Madden, J. M. Hellerstein, and W. Hong. Model-driven data acquisition in sensor networks. In *VLDB*, 2004.

[15] A. Deshpande and S. Madden. Mauvedb: supporting model-based user views in database systems. In *SIGMOD 2006*, pgs 73–84, New York.

[16] P. Desnoyers, D. Ganesan, and P. Shenoy. TSAR: A two tier storage architecture using interval skip graphs. In *SenSys 2005*, pgs 39–50, San Francisco, CA.

[17] C. T. Ee, S. Ratnasamy, and S. Shenker. Practical data-centric storage. In *NSDI*, May 2006.

[18] ENEA. Polyhedra flashlite. `http://www.enea.com/templates/Extension____9122.aspx`.

[19] D. Ganesan, B. Greenstein, D. Perelyubskiy, D. Estrin, J. Heidemann, and R. Govindan. Multi-resolution storage in sensor networks. *ACM TOS*, Aug 05.

[20] D. Gay. Design of Matchbox, the simple filing system for

motes. in TinyOS 1.x distribution, `www.tinyos.net`, August 21 2003. Version 1.0.

[21] B. Greenstein, D. Estrin, R. Govindan, S. Ratnasamy, and S. Shenker. DIFS: A distributed index for features in sensor networks. *Elsevier Journal of Ad-Hoc Networks*, 2003.

[22] M. B. Greenwald and S. Khanna. Power-conserving computation of order-statistics over sensor networks. In *ACM PODS*, 2004.

[23] J. Hellerstein, W. Hong, S. Madden, and K. Stanek. Beyond average: Towards sophisticated sensing with queries. In *IPSN*, Palo Alto, CA, 2003.

[24] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks. In *ACM/IEEE Mobicom*, pages 56–67, Boston, MA. Aug, 2000.

[25] ITTIA. Fuel data sheet. `http://www.ittia.com/library/datasheets/fuel_datasheet.pdf`.

[26] M. Li, D. Ganesan, and P. Shenoy. PRESTO: Feedback-driven data management in sensor networks. In *NSDI*, May 2006.

[27] X. Li, Y.-J. Kim, R. Govindan, and W. Hong. Multi-dimensional range queries in sensor networks. In *SenSys*, 2003.

[28] D. Li, K. Wong, Y. Hu, and A. Sayeed. Detection, classification and tracking of targets. *IEEE Signal Processing Magazine*, 2002.

[29] R. A. Lorie. Physical integrity in a large segmented database. *ACM TODS*, 2(1):91–104, 1977.

[30] D. Lymberopoulos and A. Savvides. XYZ: A motion-enabled, power aware sensor node platform for distributed sensor network applications. In *IPSN SPOTS*, April 2005.

[31] S. Madden, M. Franklin, J. Hellerstein, and W. Hong. TAG: a tiny aggregation service for ad-hoc sensor networks. In *OSDI*, Boston, MA, 2002.

[32] S. Madden, M. Franklin, J. Hellerstein, and W. Hong. Tinydb: An acqusitional query processing system for sensor networks. *ACM TODS*, 2005.

[33] G. Mathur, P. Desnoyers, D. Ganesan, and P. Shenoy. Capsule: An energy-optimized object storage system for memory-constrained sensor devices. To appear in *SenSys*, November 2006.

[34] G. Mathur, P. Desnoyers, D. Ganesan, and P. Shenoy. Ultra-low power data storage for sensor networks. In *IPSN-SPOTS*, Nashville, TN, April 2006.

[35] A. Meliou, D. Chu, C. Guestrin, and W. Hong. Data gathering tours in sensor networks. In *IPSN*, 2006.

[36] K. Nrvag. Vagabond: The design and analysis of a temporal object DBMS. PhD thesis – Norwegian University of Science and Technology, 2000.

[37] McObject. eXtremeDB. `http://www.mcobject.com/pdfs/standard_datasheet.pdf`.

[38] J. Polastre, R. Szewczyk, and D. Culler. Telos: Enabling ultra-low power wireless research. In *IPSN-SPOTS*, April 2005.

[39] J. Paradiso, G. Borriello, L. Girod, and R. Han. Applications: Beyond Dumb Data Collection (Panel). In *EmNets*, 2006.

[40] V. Poosala. *Histogram-based estimation techniques in database systems*. PhD thesis, Madison, WI, USA, 1997.

[41] R. Ramakrishnan and J. Gehrke. *Database Mgmt Systems*. McGraw Hill, 2003.

[42] S. Ratnasamy, D. Estrin, R. Govindan, B. Karp, L. Y. S. Shenker, and F. Yu. Data-centric storage in sensornets. In *ACM HotNets*, 2001.

[43] M. Rosenblum and J. K. Ousterhout. The design and implementation of a log-structured file system. *ACM TOCS*, 10(1):26–52, February 1992.

[44] N. Sadagopan, B. Krishnamachari, and A. Helmy. Active query forwarding in sensor networks (acquire). In *SNPA*, 2003.

[45] P. Seshadri, M. Livny, and R. Ramakrishnan. The design and implementation of a sequence database system. In *VLDB*, pages 99–110, 1996.

[46] A. Silberstein, R. Braynard, C. Ellis, K. Munagala, and J. Yang. A sampling-based approach to optimizing top-k queries in sensor networks. In *ICDE*, 2006.

[47] A. Silberstein, K. Munagala, and J. Yang. Energy-efficient monitoring of extreme values in sensor networks. In *ACM SIGMOD*, 2006.

[48] Stargate platform. `http://platformx.sourceforge.net/`.

[49] M. Stonebraker. The Postgres storage system. In *VLDB*, England, Sept 1987.

[50] N. Trigoni, Y. Yao, A. Demers, J. Gehrke, and R. Rajaraman. Hybrid push-pull query processing for sensor networks. In *Workshop on Sensor Networks as part of the GI-Conference Informatik*, Berlin, Germany, Sept 2004.

[51] A. Terzis, A. Anandarajah, K. Moore, and I.-J. Wang. Slip surface localization in wireless sensor networks for landslide prediction. In *IPSN 2006*, pages 109–116, New York, NY, USA, 2006. ACM Press.

[52] N. Xu, S. Rangawala, K. Chintalapudi, D. Ganesan, A. Broad, R. Govindan, and D. Estrin. A wireless sensor network for structural monitoring. In *SenSys*, 2004.

[53] Y. Yao and J. Gehrke. Query processing for sensor networks. *CIDR*, Jan 2003.

[54] D. Zeinalipour-Yazti, S. Lin, V. Kalogeraki, D. Gunopulos, and W. Najjar. MicroHash: An efficient index structure for flash-based sensor devices. In *FAST*, December 2005.

[55] R. Sen and K. Ramamritham. Efficient Data Management on Lightweight Computing Devices. In *ICDE*, Tokyo, Japan, April 2005.

[56] P. Pucheral, L. Bouganim, P. Valduriez and C. Bobineau. PicoDBMS: Scaling down Database techniques for the Smartcard. In *VLDB*, Secaucus NJ, 2001.

[57] Y. Yao and J. E. Gehrke. The Cougar Approach to In-Network Query Processing in Sensor Networks. In *SIGMOD Record*, Vol 31 Number 3, Sept 2002.

[58] C. Wu, L. Chang and T. Kuo. An Efficient B-Tree Layer for Flash-Memory Storage Systems. In *RTCSA*, Tainan, Taiwan, 2003.

[59] C. Wu, L. Chang and T. Kuo. An Efficient R-Tree Implementation over Flash-memory Storage Systems. In *ACM GIS*, 2003.

[60] J. Considine, F. Li, G. Kollios and J. Byers. Approximate Aggregation Techniques for Sensor Databases. In *IEEE ICDE*, 2004.

[61] M. Rahimi, R. Baer, O. I. Iroezi, J. C. Garcia, J. Warrior, D. Estrin, and M. Srivastava. Cyclops: In Situ Image Sensing and Interpretation in Wireless Sensor Networks. In *SenSys 2005*, New York, NY, 2005.