Multi-resolution Storage and Search in Sensor Networks

Multi-resolution Storage and Search in Sensor Networks

Deepak Ganesan
Department of Computer Science,
University of Massachusetts, Amherst, MA 01003

Ben Greenstein, Deborah Estrin
Department of Computer Science,
University of California, Los Angeles, CA 90095,

John Heidemann
University of Southern California/Information Sciences Institute,
Marina Del Rey, CA 90292
and
Ramesh Govindan
Department of Computer Science,
University of Southern California, Los Angeles, CA 90089

Wireless sensor networks enable dense sensing of the environment, offering unprecedented opportunities for observing the physical world. This paper addresses two key challenges in wireless sensor networks: in-network storage and distributed search. The need for these techniques arises from the inability to provide persistent, centralized storage and querying in many sensor networks. Centralized storage requires multi-hop transmission of sensor data to internet gateways which can quickly drain battery-operated nodes.

Constructing a storage and search system that satisfies the requirements of data-rich, scientific applications is a daunting task for many reasons: (a) the data requirements are large compared to available storage and communication capacity of resource-constrained nodes, (b) user requirements are diverse and range from identification and collection of interesting event signatures to obtaining a deeper understanding of long-term trends and anomalies in the sensor events, and (c) many applications are in new domains where a priori information may not be available to reduce these requirements.

This paper describes a *lossy, gracefully degrading storage model*. We believe that such a model is necessary and sufficient for many scientific applications since it supports both progressive data collection for interesting events as well as long-term in-network storage for in-network querying and processing. Our system demonstrates the use of in-network wavelet-based summarization and progressive aging of summaries in support of long-term querying in storage and communication-constrained networks. We evaluate the performance of our linux implementation and show that it achieves: (a) low communication overhead for multi-resolution summarization, (b) highly efficient drill-down search over such summaries, and (c) efficient use of network storage capacity through load-balancing and progressive aging of summaries.

Categories and Subject Descriptors: C.2.7 [**COMPUTER-COMMUNICATION NETWORKS**]: Sensor Networks; H.3.4 [**INFORMATION STORAGE AND RETRIEVAL**]: Systems and Software—*Distributed Systems*

Additional Key Words and Phrases: Wireless Sensor networks, Data storage, Wavelet processing, Multi-resolution storage, Data aging, Drill-down query

1. INTRODUCTION

One of the key challenges in wireless sensor networks is the storage and querying of useful sensor data, broadly referred to as data management. The term *useful sensor data* is application-specific and has different meaning in different application scenarios. For instance, in a target tracking application, users are interested in detecting vehicles and tracking their movement. Here, useful sensor data comprises target detections (timestamp and location) as well as their tracks. In a structural monitoring application such as *Wisden* ([Xu et al. 2004]), scientists are interested in spatio-temporal analysis of sensor data, such as, vibrations measured at different points of a building in response to an excitation over a period of time. This requires access to vibration event data corresponding to periods when the building is excited. Data management captures the broad spectrum of how useful data is handled in a sensor network, and addresses three key questions:

—Where is data stored in a network? Is it stored locally at each sensor node (local storage), in a distributed manner within the network (distributed storage) or at the edge of the network at the base-station (centralized storage)?

—How are queries routed to stored data? Can we use attributes of the search to make it efficient?

—How can the network deal with storage limitations of individual sensor nodes?

Many schemes have been proposed for data management in sensor networks. The main ideas are summarized along the axes of communication required for data storage and communication required for query processing in Figure 1. The figure depicts a fundamental tradeoff between centralized storage and local storage. At one extreme is a conventional approach of centralized data management where all useful sensor data is transmitted from sensors to a central repository that has ample power and storage resources. This task is communication intensive (and hence energy intensive) but facilitates querying. Queries over this data are handled at the central location and incur no energy cost to the sensor network. Centralized storage is appropriate for low-data rate, small-scale sensor networks where there are infrequent events. As the useful data rate increases, and the network scale grows, centralized storage is not feasible in sensor networks due the power constraints on sensor nodes. Aggregate costs of transmitting the data to a base-station can quickly drain power on all nodes in the network. Nodes that are closer to the base-station are especially impacted due the need to relay data from many other nodes in the network. Thus, dealing with power constraints requires alternate approaches that involves storing sensor data within the sensor network and performing query processing on this distributed data store.

Many parameters are at work in determining which data management scheme best suits an application. Is the deployment short-term or long-term? What are the energy, storage and processing resources on sensor nodes? What kind of queries are posed on the data and with what frequency? How much useful sensor data is generated by the network? What in-network data processing is appropriate for the sensor data? Dealing with the wide range of requirements and constraints in sensor networks requires a family of solutions. These solutions are different from distributed storage systems in wide-area networks in two respects. First, energy and storage limitations introduce more stringent communication

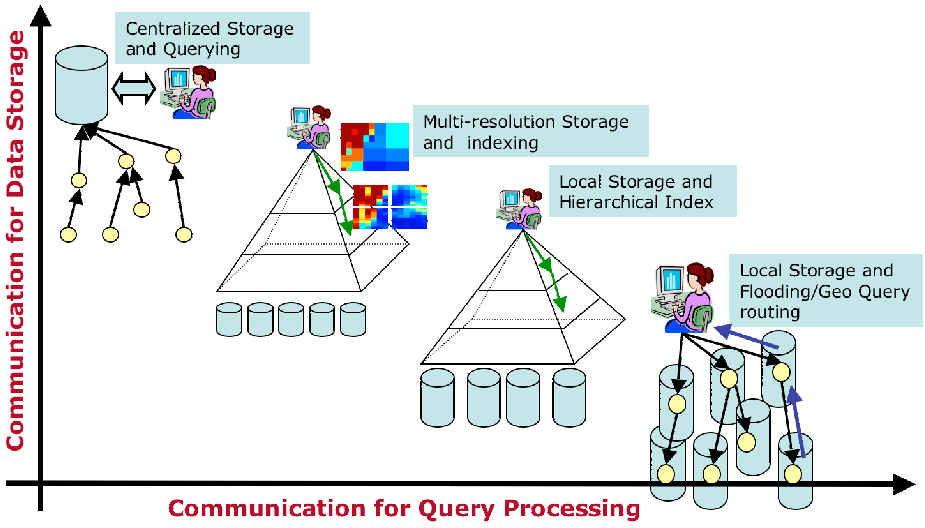

Fig. 1. Taxonomy of data storage solutions

constraints. Second, data in sensor networks exhibits spatio-temporal correlations, which can be exploited in the storage, processing and querying of data.

Our goal is to provide storage and search for raw sensor data in data-intensive scientific applications. We believe that long-term data storage or archival will complement event-driven sensor network development. Sensor network deployments involve two interacting phases. One phase involves enabling the scientific community to determine how events can be reliably detected from distributed data. The second phase involves using these events to trigger the sensor network such that the communication requirements can be significantly reduced.

Constructing a storage and search system that satisfies the requirements of data-rich, scientific applications is a daunting task as the storage requirements are large compared to available storage and communication capacity of resource-constrained nodes. Clearly, it is impossible for a sensor network to provide lossless, persistent storage and querying that a centralized wired system can provide. Our more modest goal, therefore, is to provide a *lossy, progressively degrading storage model*. We believe that such a model might be necessary and sufficient for many scientific applications for two reasons. First, a gracefully degrading storage model enables a *query and collect* approach for fresh data where users can precisely query recent data and selectively decide on important data snippets that can then be collected losslessly for persistent offline storage. Second, older data can be expected to be useful for identifying *long-term patterns*, and anomalous occurrences. Thus, older data can be stored more lossily, but with sufficient fidelity to satisfy such long-term queries.

How do we provide distributed, progressively degrading storage in a sensor network?

The key idea behind our system is spatio-temporal summarization: we construct multi-resolution summaries of sensor data and store them in the network in a spatially and hierarchically decomposed distributed storage structure optimized for efficient querying. Summaries are generated in a multi-resolution manner, corresponding to different spatial and temporal scales. Queries on such data are posed in a drill-down manner, *i.e* they are first processed on coarse, highly compressed summaries corresponding to larger spatio-temporal volumes, and the approximate result obtained is used to focus on regions in the network are most likely to contain result set data. Multi-resolution summaries are aged to provide a gracefully degrading storage model.

1.1 Contributions

Our main contributions are three-fold.

Novel Wavelet-based Distributed Data Summarization Systems: Data aggregation is a critical building block of sensor network systems. We developed a distributed wavelet-based summarization [Ganesan et al. 2003; Ganesan et al. 2002]) procedure that is very useful for sensor network data. Our summarization procedure is novel in two respects. From a distributed storage systems perspective, the use of data aggregation that exploits spatio-temporal correlation is novel. While there are many distributed storage systems designed for the internet, these deal with uncorrelated files or movies, and do not exploit correlations between them. Furthermore, the use of wavelets to deal with sensor data in a spatially distributed setting is an interesting application of wavelet transforms. We implement this codec on two different platforms, large sensor nodes (Ipaqs) and small sensor nodes (Motes).

Drill-down Query Processing: We empirically evaluate the use of drill-down query processing over multi-resolution data storage and show that high query accuracy (within 85%) at very low overhead (5% of the network) can be achieved. These results demonstrate the use of our system for query processing.

Optimization-based Graceful Data Aging: The key contribution of this work is building a data-aging framework that can deal with storage limitations in manner that provides a gracefully degrading query interface to users. We formalize the notion of storage in a distributed multi-resolution hierarchy, including data summarization and query processing. We construct an optimization procedure that determines the storage allocation at nodes in the network between different levels of resolution. This optimization procedure can be used to construct a training-data based strategy for graceful storage degradation. In the absence of training data, we propose a greedy scheme that can be used to determine the storage allocation.

The remainder of this paper is organized as follows. In Section 2, we survey research that relates to our work. Section 4 provides a overview of the architecture of DIMENSIONS, and describe the many usage models that it can support for sensor network applications. In Section 3, we outline the design goals of our system and we provide an architectural overview in Section 4. Section 5 provides a formal description of the aging problem. System implementation and performance results over a precipitation dataset are provided in Section 6. Section 7 describes one of the usage models of the system, multi-resolution data collection as part of a sensor network system for structural health monitoring. A mote implementation and performance over a structural vibration dataset is discussed as part of this work. Section 8 discusses some future extensions to our work. Finally, we conclude in section 9 with a discussion of how our system satisfies these goals.

2. RELATED WORK

Figure 1 shows four solutions for distributed storage and search. We discuss related work that address these different problems.

2.1 Centralized Storage and Search

The conventional approach to storing time series data is to have all sensing nodes feed all of their data to a central repository external to the sensing environment. In a network of n nodes, this cost is on the order of the diameter of the network for each piece of data sent, or $O(\sqrt{n})$. Queries over this data incur no additional cost to the network, because the data is already resident on storage external to the network. (Note, however, that queries initiated from within the network, perhaps by in-network actuation mechanisms, will incur $O(\sqrt{n})$.) Centralized storage may be appropriate for low-data rate, small-scale sensor networks. For instance, consider a target-tracking system that is detecting infrequent targets (once an hour), and generating event tuples, with the event type, timestamp and location of detection. The data is very small, and the event rate is low, hence centralized storage might be reasonable for a network of a hundred nodes transmitting data over 2-3 hops to a base-station. As the event or useful data rate increases, and the network scale increases, centralized storage is not always feasible in sensor networks due to the aggregate and bottleneck costs of transmitting all data towards a network gateway, potentially over multiple hops.

2.2 Local Storage and Geographical Search

At the lower right of the spectrum shown in Figure 1 is a fully local storage scheme where all useful sensor data is stored locally on each node and queries are routed to locations where the data is stored. Since data is stored locally at the sensing nodes that produced them, there is no communication cost involved in constructing the distributed store. However, in-network search and query processing can potentially incur high energy cost. Because data can reside anywhere in the network, a query that does not explicitly constrain the physical search space must be flooded to all storage nodes in the network, which costs $O(n)$, where n is the number of nodes in the network. Responses are sent back to the source of the query at a cost of $O(\sqrt{n})$ (since the network diameter is approximately $\sqrt{n}$). If only a few queries are issued during the lifetime of a network, answering these queries might involve little communication cost. A large number of flooded queries that each involves significant communication, however, can drain a network's energy reserves.

Many of the initial ideas pursued within the context of Directed Diffusion [Intanagonwiwat et al. 2000] followed such a paradigm. Sources publish the events that they detect, and sinks with interest in specific events can subscribe to these events. The Directed Diffusion substrate routes queries to specific locations if the query has geographic information embedded in it (eg: find temperature in the south-west quadrant), and if not, the query is flooded throughout the network.

There are three drawbacks of such a scheme. First, for queries that are not geographically scoped, search cost ($O(n)$) might be prohibitive for large networks with frequent queries. Second, queries that process spatio-temporal data (eg: edges) need to perform significant distributed data processing each time a query is posed which can be expensive. Third, these techniques need to be enhanced to deal with storage limitations on sensor nodes.

Application	*Sensors*	*Expected Data Rates*	*Data Requirements Per Year*
Building Health Monitoring [Kohler]	Accelerometer	30 minutes vibration data per day	8Gb
Micro-climate Monitoring	Temperature, Light, Precipitation, Pressure, Humidity	1 sample/minute/sensor	40Mb
Habitat Monitoring	Acoustic, Video	10 minutes of audio and 5 mins of video per day	1 Gb

Table I. Data Requirement estimates for Scientific Applications

2.3 Local storage with Distributed Indexing

Distributed indexing addresses search issues in the local storage mechanism described above. Recent research has seen a a growing body of work on data indexing schemes for sensor networks [Ratnasamy et al. 2002][Greenstein et al. 2003][Li et al. 2003]. These techniques differ in the aggregation mechanisms used, but are loosely based on the idea of geographic hashing and structured replication. One such indexing scheme is DCS[Ratnasamy et al. 2002], that provides a hash function for mapping from event name to location. DCS constructs a distributed storage structure that groups events together spatially by their named type. Names are considered to be arbitrary keys to the hash function and are the basic unit of categorization.

A node that detects an event stores the event at the mirror closest to its location. A search using structured replication would begin with the root, descend to its four children, descend to each of the children's four children, and so forth. DCS uses structured replication to register the existence of events at replicated rendezvous nodes. The communication cost to store a datum is $O(\sqrt{n})$; and the costs to send a query and retrieve data are each $O(\sqrt{n})$.

In traditional databases, a table is indexed in order to speed up common queries. Likewise, DCS indexes its data, but does so in a manner that is optimized for communication instead of latency. When a network administrator knows along which dimensions a query will be cast *a priori*, an indexing scheme can be employed that spatially organizes data so that a query need not visit more than a single site to be satisfied.

Distributed index of features in Sensornets (DIFS [Greenstein et al. 2003]) and Multi-dimensional Range Queries in Sensor Networks (DIM [Li et al. 2003]) extend the data-centric storage approach to provide spatially distributed hierarchies of indexes to data. In these two techniques, the storage atom is a high-level event that is described with attributes that each have associated numerical values.

3. DESIGN GOALS OF DATA STORAGE IN SENSOR NETWORKS

The storage requirements for sensing applications differ widely in their data-rates and storage needs. Table I shows a cross-section of applications and their data requirements. In this section, we describe the main design goals of a distributed storage and search for these applications.

Energy Efficient: Energy efficiency is critical to any system component designed for sensor networks, and a storage system is no exception. Energy efficiency generally depends on the amount of data communicated and other parameters such as efficiency of the duty-cycling scheme and the traffic patterns in the network. In our storage system, we will focus on limiting the amount of data communicated.

Long-term Storage: One of the major benefits of centralized storage over a distributed wireless sensor network storage is the ability to provide a reliable and persistent long-term storage capability. Scientists, in particular, are reluctant to let data be discarded for energy or other reasons, since this data could be valuable for their studies. To provide an effective service for such applications and users, we have to provide a service that can compete with centralized storage by providing a long-term persistent storage and data processing capability. The key criteria for persistent storage technology in sensor networks is power consumption and cost. The most attractive technology that fits both these criteria is Flash memory. In addition to low cost, the power requirements of flash memory are at least an order of magnitude less than communication.

Multi-Resolution Data Storage: A fundamental design goal is the ability to extract sensor data in a multi-resolution manner from a sensor network. Such a facility is very useful in storage designed for sensor data for multiple reasons: (a) it allows users to look at low-resolution data from a larger region cheaply, before deciding to obtain more detailed and potentially more expensive datasets, and (b) compressed low-resolution sensor data from large number of nodes can often be sufficient for spatio-temporal querying to obtain statistical estimates of a large body of data [Vitter et al. 1998].

Balanced, Distributed Data Storage: Design goals of distributed storage systems such as [Kubiatowicz et al. 2000][Rowston and Druschel 2001] of designing scalable, load-balanced, and robust systems, are especially important for resource constrained distributed sensor networks. We have as a goal that the system balances communication and computation load of querying and multi-resolution data extraction from the network. In addition, it should leverage distributed storage resources to provide a long-term data storage capability. Robustness is critical given individual vulnerability of sensor nodes. Our system shares design goals of sensor network protocols that compensate for vulnerability by exploiting redundancy in communication and sensing.

Robustness to Failure: An outdoor deployment of cheap sensor nodes is clearly susceptible to node and hardware failures due to the vagaries of the weather. The unpredictabilities of the environment are likely to reflect in a higher probability of nodes being permanently or temporarily disconnected and completely or partially losing their stored data. Thus, reliability schemes become an important design concern, and such schemes should be designed such that data is not lost even in the presence of a high failure probability.

Graceful Data Aging: To enable long-term storage given limited storage capacity, the system should to provide a mechanism to age data. This aging procedure should be utility-based *i.e.*, it should age data that is not useful to the user while retaining data that is potentially useful for querying. The aging procedure should be graceful and data quality should degrade slowly rather than abruptly over time to provide users with a more useful sensor network.

Exploiting Correlations in Sensor Data: Correlations in sensor data can be expected along multiple axes: temporal, spatial and between multiple sensor modalities. These

correlations can be exploited to reduce communication and storage requirements. The storage structure should be able to adapt to the correlation characteristics of sensor data.

4. ARCHITECTURE DESCRIPTION

We describe the architecture of our system in four parts: (a) the hierarchical processing that constructs lossy multi-resolution summaries, (b) the routing protocol that construct a distributed hierarchy to enable summarization at different layers, (c) the expected usage of these summaries through drill-down queries, and (d) the data aging scheme that determines how summaries should be discarded, given node storage requirements.

Summarization and data aging are periodic processes, that repeat every epoch. The choice of an epoch is application-specific, for instance, if users of a micro-climate monitoring network ([Hamilton]) would like to query data at the end of every week, then a reasonable epoch would be a week. In practice, an epoch should at least be long enough to provide enough time for local raw data to accumulate for efficient summarization.

4.1 Multi-Resolution Summarization

Our goal is to construct a system that can support a wide range of queries for patterns in data. Therefore, we use a summarization technique that is generally applicable, rather than optimizing for a specific query. Wavelets have well-understood properties for data compression and feature extraction and offer good data reduction while preserving dominant features in data for typical spatio-temporal datasets ([Rao and Bopardikar 1998][Vetterli and Kovacevic 1995]). As sensor networks mature and their applications become better defined, more specialized summaries can be added to the family of multi-resolution summaries maintained in such a framework.

Hierarchical construction (shown in Figure 2) using wavelets involves two components: temporal and spatial summarization of data.

Temporal Summarization: The first phase, *temporal summarization*, has low energy overhead since it involves only computation overhead at a single sensor node, and incurs no communication overhead. The first step towards constructing a multi-resolution hierarchy, therefore, consists of each node reducing the time-series data as much as possible by exploiting temporal redundancy in the signal and apriori knowledge about signal characteristics. By reducing the temporal data-stream to include only potentially interesting events, the communication overhead of spatial decomposition is reduced. In general, significant benefit can be expected from merely temporal processing at very little cost.

Consider the example of a multi-resolution hierarchy for building health monitoring (Table I). Such a hierarchy is constructed to enable querying and data extraction of time-series signals corresponding to interesting vibration events. The local signal processing involves two steps: (a) Each node performs simple real-time filtering to extract time-series that may represent interesting events. The filtering could be a simple amplitude thresholding i.e. events that cross a pre-determined signal-to-noise ratio (*SNR*) threshold. The thresholding yields short time-series sequences of building vibrations. (b) These time-series snippets are compressed using wavelet subband coding to yield a sequence that capture as much energy of the signal as possible, given communication, computation or error constraints.

Spatial Summarization: The *spatial summarization* phase constructs a hierarchical grid-based overlay over the node topology, and uses spatio-temporal wavelet compression to re-summarize data at each level. Figure 2 illustrates its construction: at each higher level of the hierarchy, summaries encompass larger spatial scales, but are compressed more, and

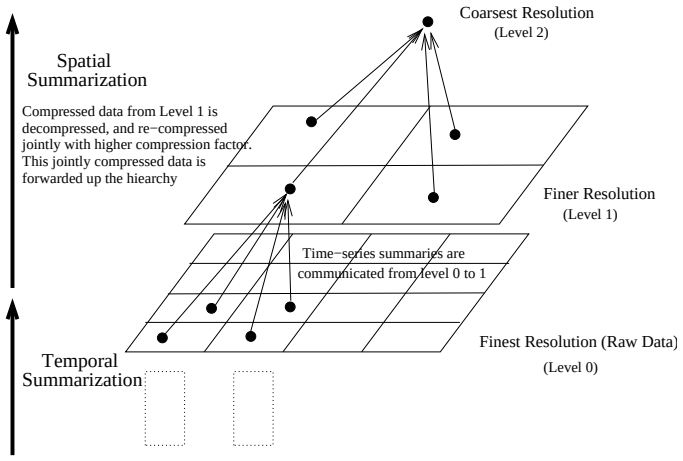

Fig. 2. Hierarchy Construction

are therefore more lossy. At the highest level (level 2), one or a few nodes have a very lossy summary for all data in the network.

Spatial summarization can be very useful in either highly over-deployed networks where there is a significant amount of spatial redundancy, or in large networks. In other instances, the reduction in total data size as a result of spatial summarization is less than that of temporal summarization. However, it still plays an important part in reducing sensor data sizes. In the instance of building health monitoring, spatial summarization may involve jointly processing correlated vibration data detected at proximate sensor nodes.

4.2 Distributed Quad Trees

While summaries at different spatial and temporal scales can be generated using wavelet processing, such hierarchical summarization will need to be enabled by appropriate routing and clustering techniques that aggregate data and stores it in a hierarchical manner. The routing framework that we use to enable the summarization of data at various spatio-temporal scales is called a Distributed Quad Tree (DQT), named after the quad-tree data structure that is frequently used in information retrieval [Wang et al. 1997].

Distributed Quad Tree (DQT) is loosely based on the notion of structured replication introduced in Data-Centric Storage (DCS [Ratnasamy et al. 2001]). Structured replication performs geographic hashes to specific locations and routes data to these locations instead of explicitly selecting clusterheads. Data is routed to the node closest to the hashed location using a variant of the GPSR protocol [Karp and Kung 2000]. This procedure is cheap from an energy perspective since it precludes the need for communication that is usually involved in a clusterhead election procedure.

DQT adds load-balancing to such a hashing and clustering scheme. Such a load balancing scheme is essential when individual storage capacity on nodes is not substantial. Clearly, a simple hierarchical arrangement as shown in Figure 2 distributes load quite unevenly. For instance, if no load-balancing were done, the highest level clusterhead (level 2) is responsible for all the coarsest resolution data. In a homogeneous network, a node that is elected to be the root has no more storage than any other node in the network, hence,

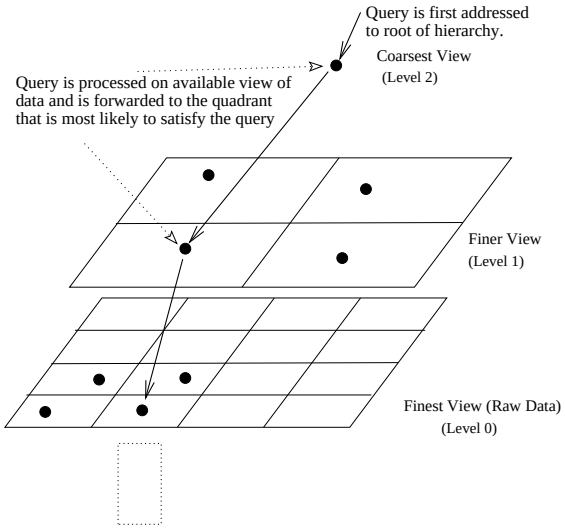

Fig. 3. Drill-down Querying

such a procedure leads to uneven storage distribution.

Our approach to deal with this problem is a simple probabilistic load-balancing mechanism, whereby each node assumes the role of a clusterhead at any level for a limited time frame. After each such time frame, a different node is probabilistically chosen to perform the role. As a result of such a load-balancing procedure, the responsibility of being a clusterhead is shared among different nodes. The performance of such a scheme depends on the node distribution, with uniform distribution of load in a regular setting. Our scheme is similar to load-balancing schemes proposed in the GAF [Xu et al. 2001] and ASCENT [Cerpa and Estrin 2002] protocols.

4.3 Drill-Down Querying

Drill-down queries on distributed wavelet summaries can dramatically reduce the cost of search. By restricting search to a small portion of a large data store, such queries can reduce processing overhead significantly. The term drill-down query is borrowed from the data mining literature, where drill-down queries have been used to process massive amounts of data. These queries operate by using a coarse summary as a hint to decide which finer summaries to process further.

Our use of drill-downs is in a distributed context. Queries are injected into the network at the highest level of the hierarchy, and processed on a coarse, highly compressed summary corresponding to a large spatio-temporal volume. The result of this processing is an approximate result that indicates which regions in the network are most likely to provide a more accurate response to the query. The query is forwarded to nodes that store summaries for these regions of the network, and processed on more detailed views of these sub-regions. This procedure continues until the query is routed to a few nodes at the lowest level of the hierarchy or until an accurate enough result is found at some interior node. This procedure is shown in Figure 3, where a drill-down query is forwarded over a logical hierarchy to the quadrants that are most likely to satisfy it.

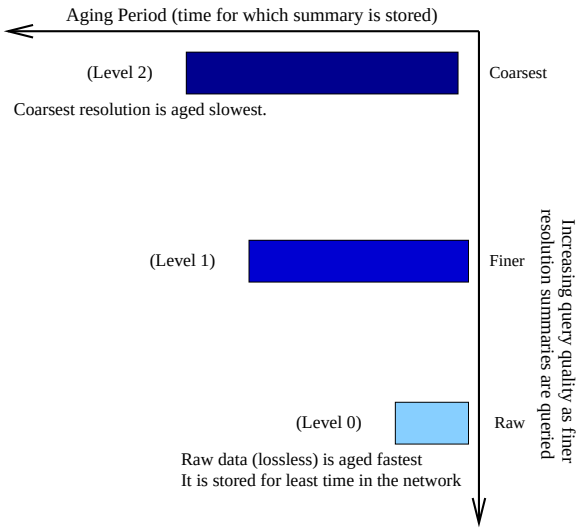

Fig. 4. Long term storage

4.4 Networked Data Aging

Hierarchical summarization and drill-down querying address challenges in *searching* for features in distributed sensor data. Providing a long-term storage and query processing capability requires storing summaries for long deployment periods. In storage-constrained networks (Table I), however, resources have to be allocated for storing new summaries by discarding older ones. The goal of networked data aging in our system is to discard summaries such that network storage resources are efficiently utilized, and graceful quality degradation over time is achieved. In other words, our work addresses the problem of apportioning the limited storage capacity in the network between different summaries.

We define the length of time for which a summary is stored in the network as the *age* of a summary. Each summary represents a view of a spatial area for an epoch, and its aging renders such a view unavailable for query processing. For instance, storing only the highest level (level 2) summary in Figure 4, provides a condensed data representation of the entire network and consequently low storage overhead compared to finer summaries, but may not offer sufficiently accurate query responses. Storing a level 1 summary (finer) in addition to the level 2 one, enables an additional level of drill-down, and offers better accuracy, but incurs more storage overhead. Figure 4 shows a typical instance of gracefully degrading storage, the coarsest summary being stored for the longest period of time, and subsequent lower level summaries being stored for progressively shorter time periods.

The networked data aging algorithms provide a storage partitioning between different summaries for each individual node such that the resultant global allocation matches user requirements. This algorithm weighs three factors: (a) the distributed storage resources in the network, (b) the storage requirements of different summaries, and (c) the incremental query benefit obtained by storing the summary.

4.5 Usage Models

A distributed multi-resolution storage infrastructure with system components as described above benefits search and retrieval of datasets that exhibit spatial correlation, and applications that use such data. In this section, we briefly describe some of the different applications that this system can be used for.

Long-term Storage: DIMENSIONS provides long-term storage to applications that are willing to sacrifice data fidelity for the ability to provide long-term storage. Long term storage is provided by exploiting the fact that thresholded wavelet coefficients lend themselves to good compression benefit [Rao and Bopardikar 1998][Vetterli and Kovacevic 1995]. The rationale in balancing the need to retain detailed datasets for multi-resolution data collection and to provide long-term storage is that if scientists were interested in detailed datasets, they would extract it within a reasonable interval (weeks). Long-term storage is primarily to enable querying for long-term spatio-temporal patterns, for which it is sufficient to store summaries that retain key features of data. Thus, the wavelet compression threshold is aged progressively, lending older data to progressively better compression, but retaining key features of data.

Querying for Spatio-Temporal features: The hierarchical organization of data can be used to search for spatio-temporal patterns efficiently by reducing the search space. Spatial features such as edges or discontinuities are important for many applications as well as systems components. Detecting edges is important for applications like geographic routing, localization and beacon placement. By progressively querying for the specific features, communication overhead of searching for features can be restricted to only a few nodes in the network. Temporal patterns can be efficiently queried by drilling down the wavelet hierarchy by eliminating branches whose wavelet coefficients do not partially match the pattern, thereby reducing the number of nodes queried.

Multi-Resolution Data Collection: Multi-resolution data extraction can proceed along the hierarchical organization by first extracting and analyzing low-resolution data from higher level clusterheads. This analysis can be used to obtain high-resolution data from sub-regions of the network if necessary. There are many circumstances under which such progressive data gathering might be useful. One instance is real-time monitoring of events of a bandwidth constrained network where all data cannot be extracted from the network in real-time. In such an instance, a compressed summary of the data can be communicated in real-time with the rest of the data being collected in non-real time when the bandwidth utilization is less (Wisden [Xu et al. 2004]).

Approximate Querying of Wavelet Coefficients: Summarized coefficients that result from wavelet decomposition have been found to be excellent for approximate querying [Chakrabarti et al. 2001][Vitter et al. 1998], and to obtain statistical estimates from large bodies of data. Often, good estimates for counting queries ([Hellerstein et al. 2003][Zhao et al. 2002]) can be obtained from higher level wavelet coefficients (range sum queries [Vitter et al. 1998]). coefficients at higher levels of the decomposition are often effective in capturing many interesting features in the original data. The hierarchy is aged progressively, with more compressed coefficients being stored for longer periods of time. Queries that mine patterns over long time-scales are executed on the compressed coefficients rather than the original dataset.

Network Monitoring: Network monitoring data is another class of datasets that exhibits high correlation. Consider wireless packet throughput data: throughput from a spe-

Symbol	*Parameter*
S	Local storage constraint
$f(t)$	User-specified aging function
$g(t)$	Provided step function
r_i	Size of each summary communicated from a level i clusterhead to a level $i+1$ clusterhead. Raw data is not communicated
R_i	Total data communicated in the network between level i and level $i+1$
s_i	Storage allocated to a level i clusterhead for storing summaries from level $i-1$
c_i	Compression ratio at level i
Age_i	Aging Parameter for level i, *i.e.,* duration in the past for which a level $i-1$ summary is available at a level i clusterhead
N	Number of nodes in the network
β	Resolution bias of the greedy algorithm

Table II. Parameters for the Aging Problem

cific transmitter to two receivers that are spatially proximate are closely correlated; similarly, throughput from two proximate transmitters to a specific receiver are closely correlated. DIMENSIONS serves two purposes for these datasets: (a) they can be used to extract aggregate statistics with low communication cost, and (b) discontinuities represent network hotspots, deep fades or effects of interference, which are important protocol parameters, and can be easily queried.

5. AGING PROBLEM

Having discussed the architectural components and usage models of our system, we now address the core question in distributed storage: How does each node age summaries such that that network can provide a long-term storage capability?

The storage allocation problem arises due to the finite storage capacity at each node. When a node's local storage is filled up, then, for each new summary that needs to be stored at a sensor node, a decision needs to be made about what data to discard in order to create storage space for the new summary. In this section, describe the criteria to consider while designing an aging strategy, formalize the aging problem and describe an efficient optimization-based solution.

5.1 Aging Problem Formulation

Consider a network with N nodes placed in a regular grid structure, over which a k-level ($k \leq \log_4 N$) multi-resolution hierarchy is constructed. The network is homogeneous, and each node samples sensor data at a rate of γ bytes per epoch, and has storage capacity, S, which is partitioned among summaries at different resolutions.

The goal of this analysis is to identify how the storage capacity of each node, S, can be partitioned among summaries at different resolutions. In order to derive the aging scheme, we use the fact that the system provides a load-balancing mechanism that distributes data approximately equally between nodes in the network. Therefore, in the analysis, we will assume perfect load-balancing, *i.e.,* each node has identical amount of storage allocated to data at each resolution. We also assume that the network is a perfectly dyadic grid, *i.e.,* a square with side 2^i, where i is an integer.

5.2 Communication overhead

The communication rate at level i is given by r_i, which determines the communication between a clusterhead at level i and a clusterhead at level $i+1$. Raw, uncompressed sensor data has rate γ, r_0 corresponds to temporally compressed data which is transmitted to a clusterhead at level 1 from a level 0 node. Similarly, r_i for $i \geq 1$ corresponds to communication overhead from level i to level $i+1$. The rate r_i depends on the compression ratio chosen for a summary at level i, c_i, defined as the ratio between size of compressed data transmitted from a clusterhead at level i to one at level $i+1$, and the total amount off raw data that the level i quadrant generates. The relation between rate, r_i, and the compression ratio that it corresponds to, c_i, is thus:

$$r_i = \frac{4^i \gamma}{c_i} \tag{1}$$

since there are 4^i nodes within each quadrant at level i, each generating γ bits, whose data is compressed by a factor c_i by the clusterhead for the quadrant.

To compute the total amount of data communicated in the entire network from level i to level $i+1$, R_i, we use the fact that there are $4^{\log_4 N - i}$ clusterheads at level i. Thus,

$$R_i = r_i 4^{\log_4 N - i} \tag{2}$$

In this paper, we will assume that the compression ratios, and therefore, the rates, have been appropriately chosen for the sensor data being studied. In practice, the relationship between lossy compression and query performance would need detailed study with the sensor dataset in question. Our goal, however, is to obtain appropriate aging parameters for a given choice of rates, r_i.

5.3 Query quality

Drill-down queries over this network can proceed hierarchically until summaries are available for the requested data. For instance, in the case of a 3-level hierarchy as shown in Figure 4, if only the coarsest summary is available, the query terminates at the root, if both the coarsest and finer summaries are available, it terminates at level 1, and so on. We define the query accuracy if a drill-down terminates at level i to be q_i. Thus, in the hierarchy in Figure 4, the query accuracy if only the coarsest resolution is queried is q_2, if the coarsest and finer resolutions are queried is q_1, and if all resolutions including raw data are queried is q_0. In practice, $q_0 \geq q_1 \geq ... \geq q_k$, *i.e.*, query quality increases with more drill-down levels since finer data is being queried.

5.4 Storage Overhead

The amount of storage required at any level is related to the total amount of data communicated to it from the lower level. For instance, a level 2 clusterhead receives summaries from four level 1 clusterheads, stores these summaries for future queries, and generates summaries of its own that are sent to level 3. We define s_i as the amount of data that a each node in the network allocates for summaries from level i. The per-epoch network-wide storage requirement for summaries from level i is R_i, which is the total amount of data communicated from level i to level $i+1$ in the network.

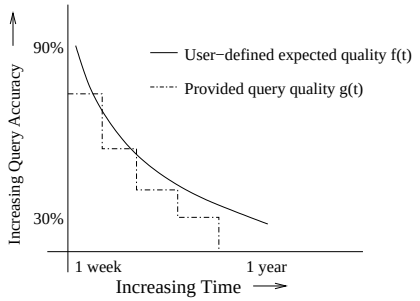

Fig. 5. Providing graceful quality degradation by aging summaries

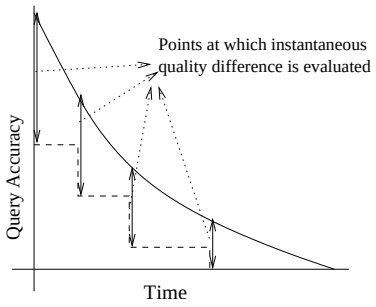

Fig. 6. Objective Function

5.5 Approximating user-specified aging function

Let $f(t)$ be a monotonically decreasing user-specified aging function which represents how much error a user is willing to accept as data ages in the network. Such a function can be provided by a domain expert who has an idea of the usage patterns of the sensor network deployment. The solid curve in Figure 5 is one instance of such a function, in which the user would like 90% query accuracy for data that is only a week old, and 30% accuracy for data that is over a year old, with a monotonically decreasing accuracy in between these two times.

We wish to approximate the user-defined aging function using a step function, $g(t)$, that represents query degradations due to summaries being aged. As shown in Figure 5, the steps correspond to time instants at which summaries of a certain resolution are aged from the network. We represent this age of each summary by Age_i.

Since each node allocates s_i data to level i summaries and there are N nodes in the network, the total networked storage allocated to data from level i is Ns_i. The total storage required for level i summaries is R_i, given by Equation 2. Assuming perfect load-balancing, the age of summaries generated at level i is:

$$Age_i = \frac{Ns_i}{R_i} = \frac{4^i s_i}{r_i} \; \forall i \geq 1 \tag{3}$$

Age of the raw data, Age_{raw} is a special case, since it is not communicated at all. If s_{raw} storage slots are allocated to each node for locally generated and stored raw data, the age of raw data, $Age_{raw} = \frac{s_{raw}}{\gamma}$.

The cost function that we choose is the *quality difference*, *q*diff (t), that represents the difference between the user-specified aging function and the achieved query accuracy at time t (shown in Figure 6). The objective is to $minimize$ the quality difference over all time. The *minimum error aging problem* can, thus, be defined as follows.

Find the ages of summaries, Age_i, at different resolutions such that the the maximum quality difference is minimized.

$$Min_{0 \leq t \leq T} \left(Max \left(q\text{diff} \right)(t) \right) \tag{4}$$

under constraints:

Drill-Down Constraint: Queries are spatio-temporal and strictly drill-down, *i.e.*, they

terminate at a level where no summary is available for the requested temporal duration of the query. In other words, it is not useful to retain a summary at a lower level in the hierarchy if a higher level summary is not present, since these cannot be used by drill-down queries.

$$Age_{i+1} \geq Age_i \; 0 \leq i \leq k \tag{5}$$

Storage Constraint: Each node has a finite amount of storage, which limits the size of summaries of each level that it can store. The number of summaries of each level maintained at a node ($\frac{s_i}{4r_i}$) is an integer variable, since a node cannot maintain a fractional number of summaries.

$$\Sigma_{0 \leq i \leq k} s_i \; \leq \; S$$
$$\frac{s_i}{4r_i} = integer\ variable$$

Additional Constraints: In formulating the above problem, we consider only drill-down queries and a network of homogeneous devices with identical storage limitations.

Our formulation can be extended to deal with different sensor network deployments and queries. For instance, queries may look at intermediate summaries directly, without drilling down. Previous research has proposed a tiered model for sensor deployments ([et al 2001]), where nodes at higher tiers have more storage and energy resources than nodes at lower tiers. Some of these constraints can be added in a straightforward manner to the above optimization problem.

For a monotonically decreasing user-specified aging function, *q*diff needs to be evaluated only at a few points (as shown in Figure 6). The points corresponds to the ages, Age_i, for each of the summaries in the network. As can be seen, the value of *q*diff at all other points is greater than or equal to the value at these points.

The minima of a maxima in Equation 4 can be easily linearized by introducing a new parameter μ

$$Min_{0 \leq i \leq n} \{\mu\} \tag{6}$$
$$q\text{diff}(Age_i) \leq \mu \; \forall \; i \tag{7}$$

The complexity of the resulting optimization procedure depends on the form of the user-specified aging function, $f(t)$. For instance, if $f(t)$ is a linear function of time, the optimization can be solved using a standard linear solver such as *lp_solve*.

5.6 Choosing an Aging Strategy

The constraint-optimization problem presented in Section 5.1 is straightforward to solve when all parameters are known. This brings up an important question: *How does one design an aging strategy with limited a priori information?*

Figure 7 shows different options that might be possible depending on the availability of prior datasets for the application. In traditional wired sensor networks, the entire dataset would be available centrally, and could potentially be used to construct an optimal aging strategy using the above-mentioned constraint-optimization procedure[1].

[1]The size of the dataset and the latency in estimating parameters using the entire dataset could preclude optimal aging even in a wired instance of the problem.

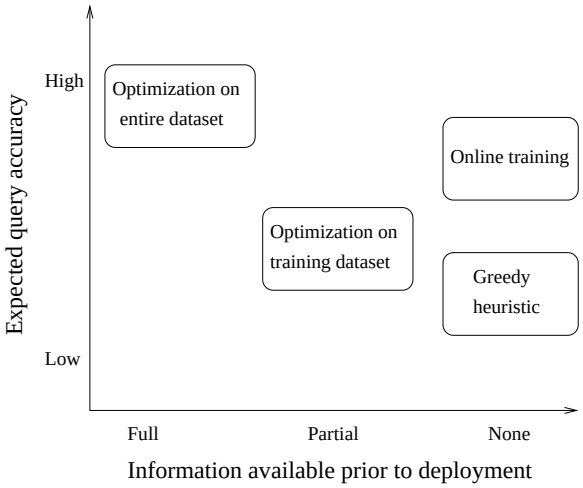

Fig. 7. Aging algorithms that operate on different levels of a priori information.

Distributed scenarios such as wireless sensor networks have to operate with less information due to the overhead of centralized data collection. In some scientific applications, a data gathering phase might precede full-fledged deployment (e.g.: James Reserve [Hamilton]), potentially providing training datasets. In other cases, there might be available data from previous wired deployments (e.g.: Seismic Monitoring [Kohler]). These datasets can be used to train sensor calibration, signal processing and in our case, aging, parameters prior to deployment. The usefulness of a training procedure depends greatly on how well the training set represents the raw data for the algorithm being evaluated. For instance, if the environment at deployment has deviated considerably from its state during the training period, these parameters will not be effective. Ultimately, a training procedure should be on-line to continuously adapt to operating conditions.

Systems, sometimes, have to be deployed without training data and with little prior knowledge of operating conditions. For instance, [Intanagonwiwat et al. 2000] describes sensor network deployments in remote locations such as forests. In the absence of training datasets, we will have to design data-independent heuristics to age summaries for long-term deployment.

We design algorithms for aging in two cases: with training data and without training data. For the case when prior datasets are available, we use the optimization problem to compute aging parameters, both for a baseline, omniscient scheme that uses full information, and for a training-based scheme that operates on a limited training set. For deployments where no prior data is available, we describe a greedy aging strategy.

5.6.1 *Omniscient Algorithm.* An omniscient scheme operates on the entire dataset, and thus, has full knowledge of the query error obtained by drilling down to different levels of the hierarchy. The scheme, then, computes the aging strategy by solving the optimization function, presented in Section 5.1, for each query type. The pseudo-code for such a scheme is shown in Algorithm 1. Omniscience comes at a cost that makes it impractical for deployment for two reasons: (a) it uses full global knowledge, which in a distributed sensor network is clearly impossible, and (b) it determines optimal aging

Level (i)	*Rate from level i to level $i+1$ (r_i)*	*Storage required per-epoch for data at level i ($4r_i$)*	*s_i (with greedy algorithm $\beta = 1$)*	Age_i
Raw	1024	1024	0	0
0 (finest)	64	256	256	4
1 (finer)	16	64	128	32
2 (coarsest)	8	32	128	256

Table III. Example of a greedy algorithm for a 16 node network

parameters for each query separately, whereas in practice, a choice of aging parameters would need to satisfy all possible queries together.

Algorithm 1 Omniscient Algorithm Pseudocode

for Each query in Q in set of QueryTypes **do**
  q_i = Query accuracy for Q obtained from entire dataset
  Solve constraint-optimization in Section 5.1
end for

5.6.2 *Training-based Algorithm.* The training scheme differs from the omniscient scheme in two ways: (a) data corresponding to a brief training period is chosen for determining aging parameters, rather than the entire dataset, and (b) a single choice of aging parameters is determined for all query types being studied.

Ideally, the choice of a training period should be such that the parameters extracted from training set is typical of the data that is sensed during system deployment. Often, however, practical limitations such as deployment conditions, storage, communication bandwidth and personnel limit the amount of training data.

Unlike the omniscient idealized algorithm, the training scheme cannot choose aging parameters per-query. In a practical deployment, a single allocation scheme would be required to perform well for a variety of queries. Therefore, the training scheme uses the error for different queries to compute a weighted cumulative error metric as shown in Algorithm 2. The cumulative error can be fed to the optimization function to evaluate aging parameters for different summaries.

Algorithm 2 Training Algorithm Pseudocode

Require: : Q: Set of all query types
Require: w_i: Weight for query type i
Require: E_i: Error for query type i from the training set
  /* Evaluate weighted cumulative error over all query types */
  $q_i = \frac{\sum_{i \in Q} w_i E_i}{|Q|}$
  Solve constraint-optimization in Section 5.1

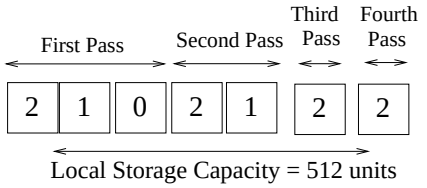

Fig. 8. Local Resource Allocation using the Greedy Algorithm. 4 coarsest, 2 finer, and 1 finest summaries are allocated

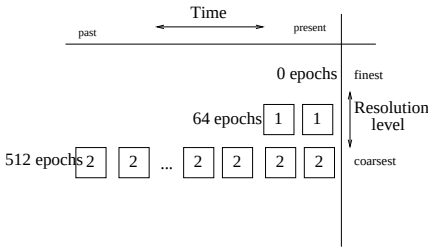

Fig. 9. Resource allocation where very long-term storage is achieved but low query accuracy is obtained old data.

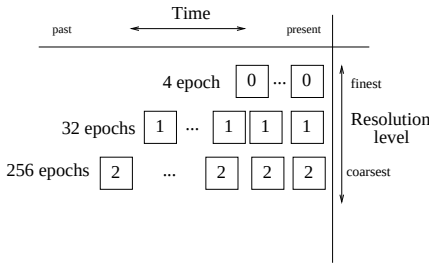

Fig. 10. Resource allocation scheme where medium-term storage is achieved with medium accuracy for queries over old data.

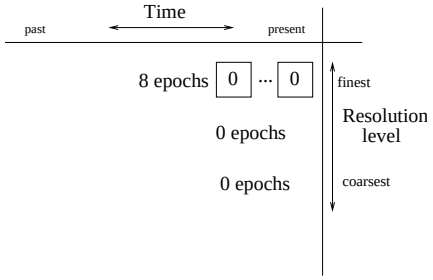

Fig. 11. Resource allocation scheme where short-term storage is achieved with high accuracy over old data.

5.6.3 *Greedy Algorithm.* We now describe a simple greedy procedure that can be used in the absence of prior datasets. The greedy procedure assigns weights to summaries according to a measure of expected importance of each resolution towards drill-down queries, represented by the parameter *r*esolution bias (β). Algorithm 3 shows the greedy allocation procedure: when available storage is larger than the size of the smallest summary, the scheme tries to allocate summaries starting with the coarsest one. The ratio of the coarsest summaries to summaries that are i levels finer are β^i. For instance, in a three-level hierarchy (Table III), a resolution bias of two means that for every coarse summary that is stored, two of finer, and four of the finest summaries are attempted to be allocated. The resolution bias parameter is used to control how gradually we would like the step function (in Figure 5) to decay.

The greedy allocation procedure specifies how the per-node parameters s_i are determined for each resolution level i. The networked age of each summary is determined from s_i using Equation 3. For instance, consider a greedy allocation with resolution bias of 1 in a 64-node network with parameters provided in Table III. There are three levels in such a hierarchy, with every node storing raw data, 16 clusterheads at level 1 storing summaries transmitted from 64 clusterheads at level 0, 4 at level 2 storing summaries from 16 level 1 clusterheads, and 1 clusterhead at level 3 storing summaries from 4 clusterheads at level

Algorithm 3 Greedy Algorithm Pseudocode

Require: N: number of nodes in the network
Require: k: number of levels
Require: r_i: the size of a summary at level i
Require: w_i: Weight for query type i
  while at least the smallest summary can fit into the remaining storage space **do**
    Assign Summaries starting from the coarsest
    for level $i = k$ down to 1 **do**
      if storage is available **then**
        allocate β^{k-i} summaries of level i
      end if
    end for
  end while

2. Consider an instance where the local storage capacity is 512 units and the sizes of each summary are as shown in Table III. The greedy allocation scheme allocates summaries starting with the coarsest level as shown in Figure 8. Note that storage is allocated in units of $4r_i$ since there are four clusterheads at level i sending data to each $i+1$ clusterhead. In the first pass, one of each summary except raw data is allocated, in the second, one coarsest and one finer summary is allocated, and in the third and fourth, one coarsest summary is allocated. Thus, a total of 128 bytes for coarsest, 128 bytes for finer, and 256 bytes for finest summary are allocated at each node. The age of summaries at various levels can be computed using the parameters provided in Table III on Equation 3. For instance, Age_0 is $\frac{4^0 s_0}{r_0} = \frac{256}{64} = 4$ epochs.

The resulting aging sequence is shown in Figure 10. The resulting allocation favors the coarsest summary more than the finer ones. Thus, the network supports long-term querying (256 epochs), but with higher error for queries that delve into older data. Raw data is aged very quickly, therefore, queries after four epochs will be unable to query raw data. Similarly, other allocations can be considered under the same resource limitations. Figure 9 shows an allocation that balances favors duration over detail, whereas the allocation in Figure 11 favors detail over duration.

6. EXPERIMENTAL EVALUATION

In this section, we describe the implementation of long-term storage and aging on a linux-based network emulation platform, Emstar ([Girod et al. 2004]). Since available dense wireless sensor network datasets lack sufficient temporal and spatial richness, we chose a geo-spatial precipitation dataset [Widmann and C.Bretherton] for our performance studies. The dataset provides daily precipitation for the Pacific Northwest from 1949 to 1994, with 50km resolution. It comprises a 15 x 12 grid of daily precipitation data for forty five years, where adjacent grid points are 50 kilometers apart.

Some features of the dataset were interesting from a system performance evaluation perspective. First, the dataset involved a reasonably long time-history, which would be useful for analyzing a data summarization algorithm. In addition, since precipitation is likely to have annual cycles, many years of data would provide us with sufficient temporal redundancy to test our data summarization techniques thoroughly. Second, the data had a reasonable spatial scale (15x12) which would enable us to explore both processing and

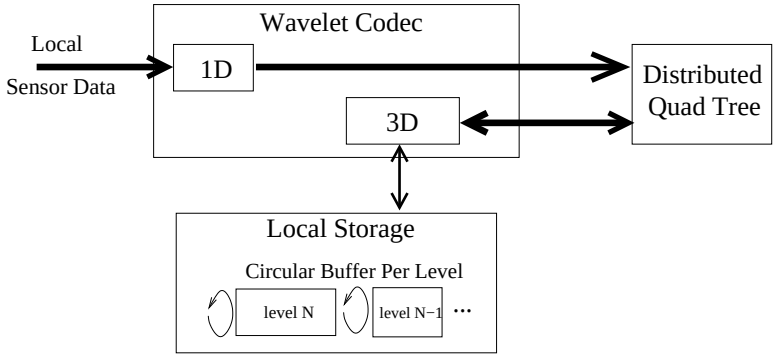

Fig. 12. Implementation Block Diagram

querying techniques over a spatial area. Finally, many different queries could be evaluated over the dataset due to its spatial and temporal richness. This includes temporal queries such as averages, maxima and minima, as well as spatial queries such as edges. Thus, while both the spatial and temporal sampling are much lower than what we would expect in a typical sensor network deployment, this dataset has edges and exhibits spatio-temporal correlations, both of which are useful to understand and evaluate our algorithms. In all our experiments, we replayed this dataset. While such geo-spatial datasets are readily available, further research has to be done to understand if such datasets are representative of sensor datasets such as at James Reserve ([Hamilton]).

6.1 Ipaq Wavelet Codec Implementation

The wavelet codec software is based on a high-performance transform-based image codec for gray-scale images (freeware written by Geoff Davis ([Davis]). We extended this coder to perform 3D wavelet decomposition to suit our application. For our experiments, we use a 9/7 wavelet filter, uniform quantizer, arithmetic coder and near-optimal bit allocator. The 9/7 filter is one of the best known for wavelet compression, and especially for images. While the suitability of different filters to sensor data requires further study, this gives us a reliable initial choice of wavelet filter.

Since the spatial scale of the dataset is 15x12, it is not feasible to use wavelet processing along the spatial axis. In practice, a grid of size, at least 32x32 would be required before spatial wavelet processing can be expected to be effective. For the given dataset, therefore, multi-resolution datasets were constructed by repeated temporal processing.

Communication overhead over a multi-resolution hierarchy is governed by the rates, r_i, that are determined as shown in Equation 1. We do not address the problem of finding the optimal r_i for a given dataset. Our objective is to choose a representative set of parameters that determine the communication overhead, *i.e.*, the compression ratios at each level, c_i, and the amount of data per epoch, γ, such that the rates, r_i increase slowly with the level of the hierarchy.

We select the parameters as follows:

—$\gamma = 3epochs * 365samples/epoch * 2bytes/sample = 2190bytes$. To construct summaries, we used an epoch of three years *i.e.*, the summary construction process repeats every three years. The choice of a large time-period was due to the temporal infrequency

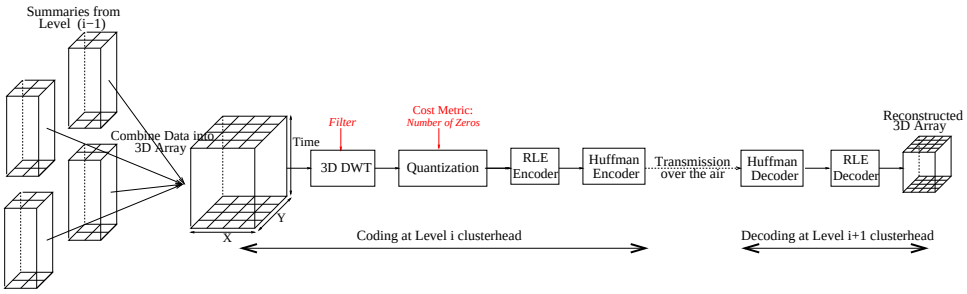

Fig. 13. Codec for the Ipaq

Hierarchy level (i)	*Num Cluster-heads (N_c)*	*Compression Ratio*	*Rate (r_i)*	*Total Data ($N_c r_i$)*
Raw	180	1	2190 (γ)	394.2K
0 to 1	180	5.97	367.1	66.08K
1 to 2	48	11.91	689.1	33.08K
2 to 3	12	23.46	1400.1	16.8K
3 to 4	4	50.9	2933.5	11.73K

Table IV. Communication Rate per Level

of samples. Each node in the network would have 1095 samples to process every three years, enough to offer reasonable temporal compression benefit. In a typical deployment, where nodes generate more data, the epoch would be much shorter.

—$c_0 : c_1 : c_2 : c_3 = 6 : 12 : 24 : 48$. Compression ratios should be chosen such that the exponential effect of aggregating data is mitigated. Our choice of compression parameters has two features that mitigate the increase in data, (a) temporal compression ratio of 6 means that approximately 367 bytes are communicated by each node at level 0, instead of 2190 bytes, and (b) the compression ratios increase by a factor of two instead of four (in Equation 1), thus, data implosion towards the root is less severe.

The total communication overhead for summaries at each level is shown in Table IV. The first row (Raw data) corresponds to uncommunicated data. The results from the codec were within 4% the input compression parameters. The standard deviation results from the fact that the dimensions of the grid are not perfectly dyadic (power of two) and therefore, some clusterheads aggregate more data than others.

6.2 Drill-down Query Performance

We use the summarized data constructed by wavelet compression to evaluate the performance of drill-down queries.

6.2.1 *Query Types.* Our implementation considers four types of queries (shown in Table V) that involve different extents of spatio-temporal processing that evaluate both advantages and limitations of wavelet compression. These queries can be classified into different categories corresponding to the spatial and temporal scales that they process as shown in Figure 14. The GlobalYearlyEdge and LocalYearlyMean queries explore features for which wavelet processing is typically well suited. The Max queries (GlobalDailyMax, GlobalYearlyMax) looks at the Max values at different temporal scales. The GlobalYear-

Type	*Query*
GlobalDailyMax	What is the maximum daily precipitation for year X?
GlobalYearlyMax	What is the maximum annual precipitation for year X?
LocalYearlyMean	What is the mean annual precipitation for year X at location Y?
GlobalYearlyEdge	Find nodes along the boundary between low and high precipitation areas for year X

Table V. Spatio-temporal queries posed on Precipitation Dataset

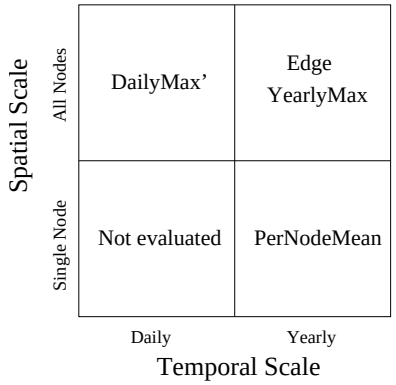

Fig. 14. Classification of Spatio-temporal Queries

lyMax query looks at the maximum yearly precipitation in the entire network, while the GlobalDailyMax queries for the daily global maximum.

Both the LocalYearlyMean query and the two Max queries are processed as regular drill-down queries. The query is processed on the coarsest summary to compute the quadrant to drill-down, and is forwarded to the clusterhead for the quadrant. The GlobalYearlyEdge query tries to find nodes in the network through which an edge passes, and involves a more complex drill-down sequence. This query is first processed by the highest-level clusterhead, which has a summary covering the spatial extent of the entire network. The clusterhead uses a standard canny edge detector to determine the edge in its stored summary, and fills a bitmap with the edge map. The query and the edge bitmap are then forwarded to all quadrants that the edge passes through. The cluster-heads for these quadrants run a canny detector on their data, and update the edge bitmap with a more exact version of the edge. The drill-down stops when no edge is visible, and the edge bitmap is passed back up, and combined to obtain the answer to the query.

6.2.2 *Query Performance.* We now evaluate the performance of drill-down queries over the multi-resolution dataset constructed as described above. Our goal in this section is to demonstrate the search features of the system and prove our claim that multi-resolution storage can be useful for a broad variety of queries.

To evaluate performance, each of the queries shown in Table V was posed over the dataset. For yearly queries (GlobalYearlyEdge and and GlobalYearlyMax), there were 45 instances each, since there are 45 years of data. For the GlobalDailyMax query, the results are averaged over 16801 instances (one for each day), and for GlobalYearlyMean, the queries were averaged over 8100 queries (180 nodes x 45 years).

The query accuracy for a drill-down query that terminates at level i (q_i) is measured as

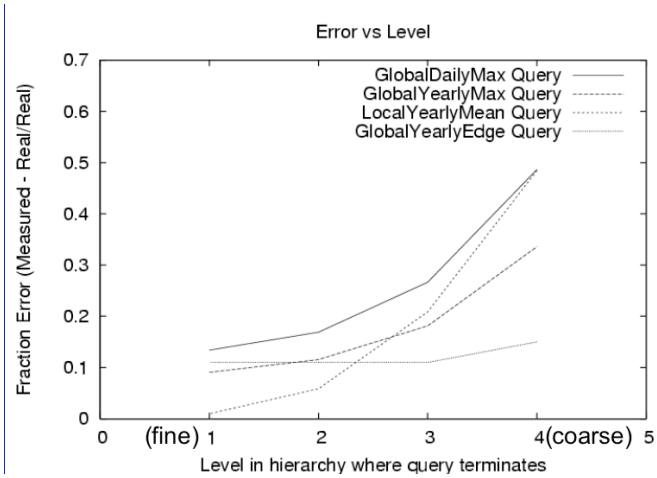

Fig. 15. Query error decreases as they drill-down to lower levels of hierarchy. Summaries at lower levels typically contribute less to reducing query error than higher level summaries.

the fraction error *i.e.*, the difference of the measured drill-down result and the real result over raw data over the real result (measured - real/real). Figure 15 shows the variation of query quality for queries defined in Table V for different levels of drill-down.

Performance for LocalYearlyMean, GlobalYearlyMax and LocalDailyMax queries are very similar, as shown in Figure 15. All of them have an error of 40-50% if only the coarsest (level 4) summaries are queries, but reduce rapidly when the drill-down proceeds to the lowest level. Even one or two levels of drill-down significantly improve error, for instance, querying level 3 in addition to level 4 reduces error to under 20%, and querying level 2 as well reduces error to less than 5%.

For the GlobalYearlyEdge query, we measure error as the fraction of nodes missed from the real edge. This query exhibits a different trend from other queries, with lower error by querying the coarsest level, and less benefit due to further drill-downs. Thus, in Figure 15, the error is 15% when only the coarsest (level 4) summaries are queried. The error reduces to 11% with an additional level of drilldown, however, further drill-downs do not improve the result. This trend is consistent with what one would expect for edge detection, the edge is more visible in a lower resolution (and consequently, higher level) view, and becomes more difficult to observe at lower levels of the hierarchy. In a larger network, with more levels, improvement might be observed using drill-down. Additionally, a more relaxed definition of query error can be considered, for instance, only nodes that are not nearest neighbors of the real edge are considered erroneous. The edge error is seen to be less than 2% with such a definition.

The communication overhead of in-network processing of these queries is extremely low as well. Even with false positives, the total query overhead of a GlobalYearlyEdge query is less than 10% of the network. Other drill-down queries such as GlobalYearlyMax and LocalDailyMax drill-down query only around 5% of the network. This performance results from hierarchical processing of queries, and for many queries that require a single answer (mean,max,min), the overhead is only $O(log_4 N)$ (one branch probed per level), *i.e.*, only around 5% of the network is queried for the result.

Level till which drilled down	GlobalYearlyMax		GlobalDailyMax		LocalYearlyMean		GlobalYearlyEdge		*Cumulative Training Error*
	Omniscient	*Training*	*Omniscient*	*Training*	*Omniscient*	*Training*	*Omniscient*	*Training*	
1	1.6%	1.2%	3.2%	6.6%	1.0%	1.0%	11.2%	7.5%	5.4%
2	5.5%	5.0%	7.2%	8.9%	5.9%	6.1%	11.2%	7.5%	9.2%
3	16.9%	12.2%	17.6%	12.9%	20.9%	21.0%	11.2%	7.5%	17.9%
4	38.6%	32.2%	40.8%	30.4%	48.4%	49.8%	15.6%	7.5%	39.9%

Table VI. Comparing the error in Omniscient (entire) Dataset vs Training (first 6 years) Dataset

These results demonstrate a key advantage of multi-resolution storage. While there is an initial overhead of communicating summaries, this overhead can be amortized over many queries posed by the users.

6.3 Aging Performance Evaluation

In this study, we consider linear aging functions of the form,

$$f(t) = 1 - \alpha t \tag{8}$$

The parameter α can be varied depending on the rate at which the user would like the aging function to decay. A large α would generate a rapidly decaying aging function.

As shown in the previous section, different summaries contribute differently to the overall query quality, with the top-level summary contributing maximum. For instance, in the case of the GlobalDailyMax query, query error reduces by 50% by storing only the level 4 summary. Adding an additional level of summaries decreases error by 15%, and so on till storing raw data results in 0% error. This trend motivates the aging problem, which allocates storage to different summaries based on their marginal benefit to query processing, and their storage utilization. In this section, we will look at the impact of aging summaries based on their relative importance. Since raw data adds little to the overall query result (Figure 15), we will assume that nodes store only summaries at various levels and not raw data.

The parameter, α, in Equation 8 is varied between 0.01 and 0.002, and determines whether the user would like a fast decay of query accuracy over time, or a slower decay.

We evaluate the three aging schemes, using the globally omniscient scheme as a baseline to compare the more practical training-based and greedy schemes. In this comparison, we increase the amount of local storage allocated to each node in the network from 0KB to 100KB, in steps of 4KB blocks. As with the previous section, our metric for error is *q*diff (Equation 4).

6.3.1 *Omniscient Strategy: Establishing a Lower Bound for Query Error.* The omniscient scheme uses the query error for each query on the entire dataset (Figure 15) to determine the optimal choice of aging parameters for each query type. As shown in Table VI, the error from the coarsest summaries ranges from 30% to 50% for different queries. As the local storage capacity increases, however, the optimal algorithm performs dramatically better, until 0% error is achieved when all levels can be drilled down. This behavior is also shown in Figure 16, which shows the performance of this scheme for the GlobalYearlyMax query on one instance of a user-specified linear aging function ($\alpha = 0.002$). In

α	*Omniscient*	*Training*	Greedy		
			Duration (β=0.5)	Balanced (β=1)	Detail (β=2)
0.01 (fast)	13.6%	14.8%	20.6%	13.7%	13.9%
0.0033	15.0%	15.9%	25.3%	16.0%	16.0%
0.002 (slow)	18.2%	19.2%	28.6%	20.0%	26.1%

Table VII. Comparison of between omniscient, training and greedy schemes. Training is within 1% of the omniscient scheme. The greedy algorithm shows significant variability to the choice of β, however, the *balanced* resolution bias performs within 2% of the omniscient scheme.

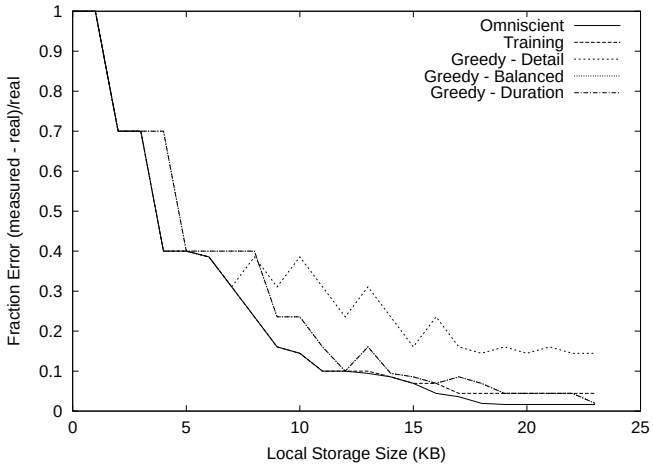

Fig. 16. Comparison of Omniscient, Training and Greedy strategies for GlobalYearlyMax query($\alpha = 0.002$)

networks composed of nodes with low local storage capacities, the error is high since only the coarsest summaries can be stored in the network .

6.3.2 *Evaluating Training using Limited Information.* In our evaluation, we use a training period of two epochs of data (10% of total deployment time) to predict the query accuracy for the entire dataset. Summaries are constructed over the training set, and all queries in Table V are posed over these summaries. Ideally, the error obtained from the training set would mirror error seen by the omniscient scheme.

How effectively does the the training dataset represent the entire dataset? Table VI shows that the predicted error from the training set is typically within 5% of the query quality seen by the omniscient scheme, but is almost 10% off in the worst case (GlobalDailyMax query over level 4 summaries). Also, in the case of the GlobalYearlyEdge query, the error seen from the training dataset is consistently off of the average result. Thus, the training set is moderately representative of the entire dataset.

To compute the cumulative error using Algorithm 2, we use equal weights for all queries. When usage statistics of sensor networks become available, application-specific weighting schemes can be used. This cumulative error can be fed to the optimization function to evaluate aging parameters for different summaries.

The first column in Table VII shows the difference between the performance of training and the optimal schemes. These results are aggregate results over a range of storage sizes

(0 - 100KB) and query types (shown in Table V). Training performs exceedingly well, and in fact is on average less than 1% worse than the optimal solution. These results are very encouraging since it suggests that even with moderately representative training datasets, very good performance can be observed.

Having seen the aggregate result, we look at a single run in Figure 16, that shows how the performance varies as the amount of storage allocated to a node is increased. Figure 16 shows the result of such a resource allocation for the GlobalYearlyMax query. As expected, increasing the storage size reduces error for all schemes. Notably, the training curve follows the omniscient storage allocation curve very closely (almost indistinguishably). Similar results were obtained for other queries as well.

6.3.3 *Greedy Algorithm.* We use three settings for resolution bias (β), a low resolution bias ($\beta = 0.5$), that favors $duration$ over detail, a medium bias ($\beta = 1$), that $balances$ both duration and detail, and a high bias ($\beta = 2$), that favors $detail$ over duration.

As seen in Table VII, varying the settings of resolution bias for the greedy heuristic significantly changes the performance of the greedy heuristic. When α is large, the user-specified aging function has a steep slope. In this case, a low resolution bias (*duration*) performs poorly since it prefers coarser summaries much more than finer ones. In contrast, when α is small and the user-specified aging function has a gradual slope, a high resolution bias (*detail*) performs exceedingly bad, since allocates more storage to finer summaries, thereby reducing duration. In both cases, the *balanced* summary assignment performs well, and has a worst-case performance of around 5% in comparison with the omniscient scheme, and 4% in comparison with the training scheme.

This result can be understood by looking at the relationship between resolution bias, β, and the slope of the user-specified aging function, α. A low value of resolution bias (*duration*) results in more storage being apportioned to coarser summaries, thus biasing towards very long duration, but low accuracy. The maximum user error (max(*q*diff))is observed for queries that look at recent data, where the user expects high accuracy, but the system can only provide coarse summaries. Thus, such an allocation performs well when the user-specified aging function requires very long duration storage (eg: $\alpha = 0.002$), but badly for short duration storage (eg: $alpha = 0.05$). In contrast, a higher value for resolution bias (*detail*) allocates significant storage to finer summaries. The error is low for queries on recent data, but the age of all summaries is limited as well. Queries on old data will result in a large max(*q*diff) because summaries will be unavailable in the network for old data. Thus, as we vary the resolution bias, β between these extremes, we get different results from the greedy algorithm. An ideal choice of β is seen to be $\beta = 1$ (*balanced*), which lies between these extremes, and results in more gradual aging of summaries.

This hypothesis also explains Figure 16. For a user-specified aging function that favors duration ($\alpha = 0.002$), the greedy algorithm with *detail* bias consistently has high error, whereas *balanced* and *duration* bias perform significantly better.

7. USAGE MODEL: PROGRESSIVE LOSSY DATA COLLECTION

Having described the DIMENSIONS system and its implementation for Linux-based nodes, we describe one of the usage models outlined in Section 4.5 - progressively lossy data collection. Many existing deployments for sensor networks have been deployed for data collection since they enable collection of previously unavailable fine-grained datasets for many different scientific disciplines. In this section, we describe how a subset of the archi-

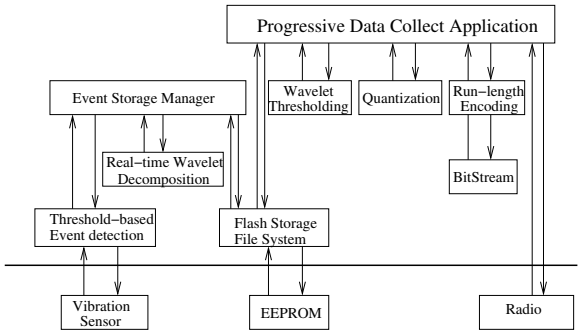

Fig. 17. Component Diagram of Mote implementation

tecture that we described can be used to enable scalable data collection in sensor networks. The implementation was done over motes as part of the *Wisden* [Xu et al. 2004] system.

Wisden is a wireless sensor network system for structural-response data acquisition. The system continuously collects structural response data from a multi-hop network of sensor nodes, and displays and stores the data at a base station. *Wisden* can be thought of as a first-generation wireless structural monitoring system; it incorporates some in-network processing, but later systems will move more processing into the network once the precise structural monitoring applications are better understood. In being essentially a data collection system, the system resembles other early sensor networks such as those being deployed for habitat monitoring [Hamilton ; Mainwaring et al. 2002].

While the architecture of *Wisden* is simple—a base station centrally collecting data—its design is a bit more challenging than that of other sensor networks built till date. Structural response data is generated at *higher data rates* than most sensing applications (typically, structures are sampled upwards of 100 Hz). The relatively low radio bandwidths, the high packet loss rates observed in many environments, and the resource constraints of existing sensor platforms add significant challenges to the system design.

To address the latency of data acquisition, we have designed and implemented a progressive storage and transmission strategy on the motes. This approach uses local storage on the motes as a in-network cache for raw data and transmits low-resolution compressed summaries of data in near-real time. The user can visualize this summarized event data and the raw data can be collected from the distributed caches when required. This on-demand data collection has to occur within the time window before which data in the cache is replaced by newly generated samples. As we show below, such an approach can compress vibration data by a factor of 20; when coupled with event detection, it can reduce the acquisition latency to less than in a minute in many cases.

7.1 Wavelet Codec Internals

We use an optimized implementation for the motes due to memory and computation constraints. For this reason, many design choices that we make are simpler than other progressive codecs such as JPEG2000. The component diagram of our implementation is shown in Figure 17. We now describe the individual system components in more detail.

Integer-Integer Wavelet decomposition: Our implementation uses the bi-orthogonal Cohen-Daubechies-Feauveau (2,2) (CDF(2,2)) integer wavelet lifting transform that relies

solely on integer addition and bit shifting operations. Wavelet lifting involves two steps: (a) a prediction step when the odd values of the time-series are predicted from the even values, and (b) an update step when the even values are updated to capture the error in the prediction step. The predict and update operations for the CDF(2,2) lifting transform are:

$$d_i \leftarrow d_i - \frac{1}{2}(s_i + s_{i+1})$$
$$s_i \leftarrow s_i - \frac{1}{4}(-d_{i-1} - d_i)$$

The choice of the above lifting transform over other kernels was based on two factors: computation overhead and compression performance. Using longer wavelet filters involves more computation overhead but does not provide significant compression improvement over the chosen filter, at least for the building vibration dataset that we studied ([Kang et al.]).

While the lifting transform itself is very efficient, normalization of coefficients at various subbands involves floating point operations. The normalization coefficients for the CDF(2,2) transform are:

$$n_H = \sqrt{2} \tag{9}$$
$$n_L = \frac{1}{\sqrt{2}}$$

where n_H is the higher frequency subband and n_L is the lower frequency subband. We perform the normalization operations during the wavelet thresholding step rather than during wavelet decomposition to be more computationally efficient.

The wavelet codec operates on buffers of length 2^n, where n is a positive integer. To avoid blocking artifacts at the buffer boundaries, we pad each buffer with a few samples at either end.

Quantization: Quantization involves representing a range of values in a signal by a single value. This reduces the number of symbols that are required to represent a signal, and hence makes the signal more compressible. We implemented a simple uniform quantizer that can be used to reduce the resolution of data depending on the range of the signal and the number of bits allocated to each sample.

Signal Thresholding: Thresholding is a technique used to modify the wavelet decomposed signal such that the resulting signal contains long sequences of zeros that can be efficiently compressed by an entropy coding scheme. We use a hard thresholding scheme in which if the absolute value of any wavelet falls below the threshold, it is set to zero. We maintain a probability density function (pdf) of the signal to facilitate the selection of an appropriate threshold. The user specifies what percentage of the signal need to be zeros in the lossy version, and the pdf can be used to determine the appropriate threshold.

The thresholds for different subbands are normalized using the coefficients shown in Equation 9. This operation needs to be done only once, hence, it reduces the computation requirements of normalizing the signal.

Run-length encoding: Wavelet decomposition, quantization and thresholding process the signal to make it more amenable for compression by an entropy coding scheme, but no compression has yet occurred. An entropy coding scheme is typically designed such that the symbols that occur most frequently use the least amount of bits. Run length coding

is the simplest of such schemes that exploits *runs* of a particular value in the signal. This scheme was preferred over other encoding schemes such as Huffman or Arithmetic coding due to its simplicity of implementation, a necessary requirement for Mote-based software.

BitStream: The run-length encoded signal is a series of symbols of different lengths depending on the number of bits used in quantization and the lengths of the special symbols used in the run length encoding process. A bitstream module is used to pack these variable length symbols into the data segment of a TinyOS message packet. Each time the data segment of a packet is filled up, it can be scheduled for transmission to the base station.

7.1.1 *Operation Description.* The progressive transmission operation involves three steps: event detection, local data storage, and progressive coding. An event detection scheme (discussed in detail in [Xu et al. 2004]) runs continually, and triggers when an event is detected. The event signal then undergoes wavelet decomposition, and the decomposed signal is written to the persistent external flash memory on the mote. Until this point, no compression has occurred, hence, the lossless event data is available on flash.

A separate periodic task reads the flash memory and compresses the signal using the five step process described in Section 7.1, after which it transmits the bitstream to the base-station. The choice of signal threshold and number of quantization bins is assumed to be determined by a priori analyis of training data to obtain maximum compression benefit within the specified error bounds.

A user at the base-station can analyze the low-resolution signal in real-time and request either the raw data or additional detail in the signal. Since the raw data is stored on flash, the difference between the previously transmitted resolution and the requested resolution is coded by the steps described in Section 7.1 and transmitted to the base-station. This progressive transmission procedure should be completed before the data on flash is over-written by future events.

This implementation is currently not integrated into the rest of our system due to the need for significant memory optimization.

	Computation Time	Memory Utilization
Wavelet Decomposition and Flash Storage	6.44ms	288bytes
Uniform Quantizer	0.32ms	7bytes
Run-length Encoder	6.30ms	20bytes

Table VIII. Performance of 128-sample 4-level transform

7.2 Performance Evaluation

We evaluate the performance of our system on three fronts: (a) the applicability of wavelet compression to structural vibration sensor data, (b) the computation and memory overhead of doing the wavelet compression in real-time on a mote, and (c) the compression gain by using our scheme, which translates to the latency of data acquisition. We used data from shaker table tests at CUREE-Kajima [Kang et al.].

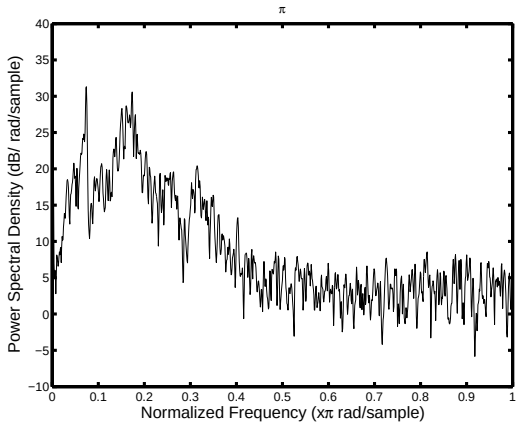

Fig. 18. Periodogram of the Power Spectral Density estimate of the structural vibration event. Energy is concentrated in the low-frequency bands, making the use of wavelet compression ideal.

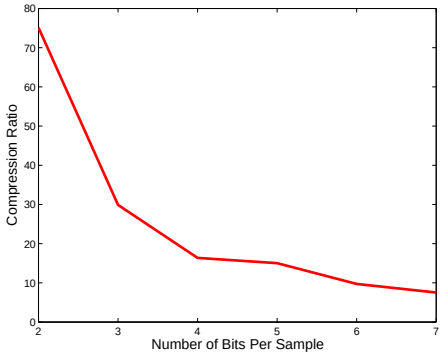

Fig. 19. Compression Ratios

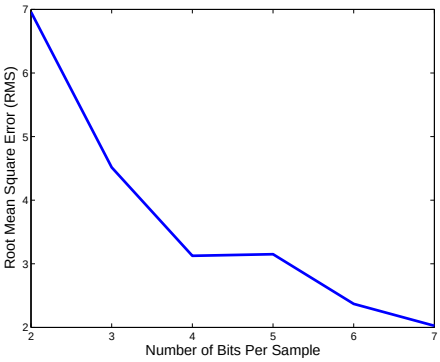

Fig. 20. Root Mean Square Error

7.2.1 *Applicability of Wavelet Processing.* Our progressive transmission strategy for vibration data uses wavelet compression. The applicability of wavelet techniques follows from characteristics of large structures, whose frequency response is usually focused in the low-frequency components [Vetterli and Kovacevic 1995]. Figure 18, which shows the power spectral density of a real vibration signal collected as part of the CUREE-KAJIMA project. It illustrates quite clearly that low-frequency components dominate the power spectral density, motivating the use of wavelet-based data processing.

7.2.2 *Computation and Memory Requirements.* Table VIII shows the computation and memory requirements of the core components of our compression system. The computation time is low, and enables us to perform the entire operation in real-time. We were able to perform sensor data sampling, 128 sample CDF(2,2) wavelet lifting transform as well as writing the decomposed buffer to the EEPROM for sampling rates upto 250Hz. As can be seen in Table VIII, the memory requirements are low as well, making it easier to integrate with other components of the system.

7.2.3 *Compression and Error Performance.* The compression and error results from using such a scheme are shown in Figure 19 and Figure 20 respectively. Both graphs use a threshold that sets 80% of the decomposed signal to zero. The trends of both graphs are as expected; as the number of quantization bits increases, both the compression ratio and the RMS error reduce. One sweet spot that emerges from these two graphs is a 4-bit quantization scheme. This choice gives us about 20-fold reduction in data size with very low RMS error of 3.1. The peak-to-peak signal to noise ratio (PSNR) for the above choice of parameters is 30dB.

These results are very promising are indicate that such an approach can be used to allow near real-time structural data acquisition from tens of sensors.

8. FUTURE RESEARCH PROBLEMS

Having discussed distributed data storage and aging, we point to some interesting potential research directions with distributed storage. There are many aspects of distributed storage that need to be addressed.

8.1 Building Long-term Data Archival Systems

Many current sensor network deployment are data collection-based ([Hamilton ; Mainwaring et al. 2002]) as they collect data for scientists to provide more datasets for analysis. As sensor network deployments become more commonplace, we believe that the need to optimize for lifetime of such networks will result in a shift towards distributed storage and querying systems.

Long-term data archival systems for sensor networks will need new tools. First, distributed storage and search techniques need to be adapted to heterogeneous sensor networks with different sensor modalities and nodes with different storage and processing constraints. Systems approaches for such heterogeneous networks will require careful splitting of processing and storage functionality. However, we believe that multi-resolution techniques are well-suited to such networks due to its ability to adapt the compression ratio to the resource-constraints on different devices. Second, much of the focus in the sensor network community has been about optimizing the power-constrained network rather than integrating it to a wide-area system. Eventually, sensor networks will form the edges of a larger wide-area network. Thus, an interesting question is how to build a wide-area querying system where data is archived at the edges.

8.2 Coping with Irregular Spatio-Temporal Sampling

A central issue that impacts our data storage system is the impact of irregular sampling on many different aspects of sensor network design. A large class sensor network deployments will have irregular spatial configurations for two fundamental reasons: (a) the phenonmena of interest are not uniformly distributed and the deployment of sensor resources will be variable in order to achieve denser sensing where there is greater spatial variability (e.g., on the edge of biological regions), and (b) terrain and other deployment practicalities bias deployment locations to where necessary power sources, communication or access can be achieved. For instance, in environmental monitoring networks such as that shown in Figure 8.2, node placement is irregular.

Irregular deployments impacts the design and performance of our system. Consider a 2-dimensional grid of sensor data where samples are taken in a non-uniform manner. A naive scheme would be to assume that the samples were regular and perform the wavelet

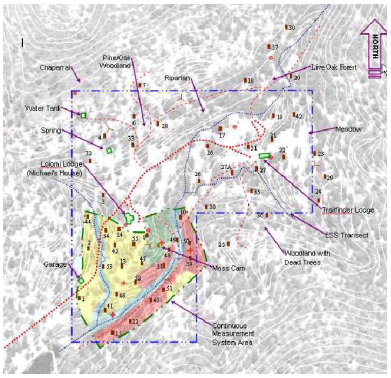

Fig. 21. Micro-climate monitoring sensor network deployment at James Reserve: Node placement is irregular, with the lower left being more densely deployed than the rest of the network

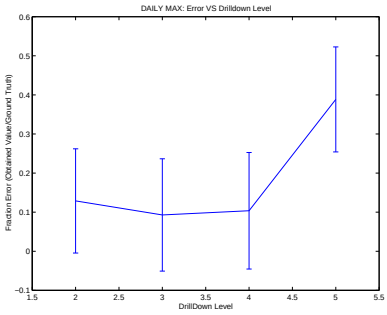

Fig. 22. Drill-down Daily MAX query performs quite well in an irregular setting.

compression accordingly. However, this creates numerous artifacts, and can distort the original data greatly since spatial correlation is not correctly captured.

Irregular spatial samples are routinely regularized in geo-spatial data processing since analysis of irregular datasets is significantly more complex than that of regularly spaced ones. This regularization procedure, called *resampling*, typically involves interpolation and can be used to deal with irregularity. The cheapest interpolation scheme for distributed sensor data is *nearest neighbor*, which assigns the value of a resampled grid point to the nearest known data sample. Such sampling can be done in a distributed and inexpensive manner by constructing the Voronoi cells corresponding to each sensor node. Ease of localized construction of voronoi cells [Ganeriwal et al. 2003; Meguerdichian et al. 2001] makes them particularly attractive as means to deal with irregularity. Higher degree polynomials can be used to improve the precision of the interpolation, especially if node distribution is highly skewed.

We combined nearest neighbor interpolation with DIMENSIONS to deal with highly irregular topologies. Figure 22 shows the performance of a GlobalDailyMax query over a highly irregular topology. While the performance trend is similar to the results in the regular topology case, the quality of the drill-down solution does not always improve with the level as one would expect. This can be attributed to artifacts in the interpolation scheme caused due to the large extent of data irregularity. We hope to improve these results with better interpolation and modeling.

9. CONCLUSIONS

As sensor networks start being deployed, the question of data storage and querying will become increasingly important. A closely related technological trend that demonstrates this importance is RFIDs. Data management in RFIDs is quickly becoming a critical problem as massive amounts of information being generated by these systems. Similarly, sensing the physical world makes it essential to deal with the large volumes of data generated by sensor networks. In-network storage and search in sensor networks is one of the main aspects of data management and poses considerable challenges. In-network storage is necessary for sensor networks because in power-limited systems, it is more efficient to store

data locally than to transmit to a central location. Significant research challenges emerge due to the need to optimize for resources, power, and the types of queries that are posed on the data.

Ideally, a search and storage system for sensor networks should have the following properties: (a) low communication overhead, (b) efficient search for a broad range of queries, and (c) long-term storage capability. In this paper, we present the design and evaluation of DIMENSIONS, a system that constructs multi-resolution summaries and progressively ages them to meet these goals. This system uses wavelet compression techniques to construct summaries at different spatial resolutions, that can be queried efficiently using drill-down techniques. We demonstrate the generality of our system by demonstrating the query accuracy for a variety of queries on a precipitation sensor dataset. Our proposal for progressive aging includes schemes that are applicable to a spectrum of application deployment conditions: a training algorithm where training sets can be obtained, and a greedy algorithm for others. A comparison shows that both the training and greedy scheme perform within 2% of an optimal scheme. While the training scheme performs better than the greedy scheme in practice, the latter performs within 1% of training for an appropriate choice of aging parameters.

REFERENCES

CERPA, A. AND ESTRIN, D. 2002. Ascent: Adaptive self-configuring sEnsor networks topologies. In *Proceedings of the IEEE Infocom*. IEEE, New York, NY.

CHAKRABARTI, K., GAROFALAKIS, M., RASTOGI, R., AND SHIM, K. 2001. Approximate query processing using wavelets. *VLDB Journal: Very Large Data Bases 10*, 2–3, 199–223.

DAVIS, G. Wavelet Image Compression Kit.

ET AL, A. C. 2001. Habitat monitoring: Application driver for wireless communications technology. In *Proceedings of the 2001 ACM SIGCOMM Workshop on Data Communications in Latin America and the Caribbean*.

GANERIWAL, S., HAN, C.-C., AND SRIVASTAVA, M. B. 2003. Going beyond nodal aggregates : Spatial average of a continuous physical process in sensor networks. In *Poster in Sensys 2003*. to appear.

GANESAN, D., ESTRIN, D., AND HEIDEMANN, J. 2002. Dimensions: Why do we need a new data handling architecture for sensor networks? In *First Workshop on Hot Topics in Networks (Hotnets-I)*. Vol. 1.

GANESAN, D., GREENSTEIN, B., PERELYUBSKIY, D., ESTRIN, D., AND HEIDEMANN, J. 2003. Multi-resolution storage in sensor networks. In *Proceedings of the First ACM Conference on Embedded Networked Sensor Systems (SenSys)*.

GIROD, L., STATHOPOULOS, T., RAMANATHAN, N., ELSON, J., ESTRIN, D., OSTERWEIL, E., AND SCHOELLHAMMER, T. 2004. A system for simulation, emulation, and deployment of heterogeneous sensor networks. In *Proceedings of the Second ACM Conference on Embedded Networked Sensor Systems*. Baltimore, MD.

GREENSTEIN, B., ESTRIN, D., GOVINDAN, R., RATNASAMY, S., AND SHENKER, S. 2003. Difs: A distributed index for features in sensor networks. *Elsevier Journal of Ad Hoc Networks*.

HAMILTON, M. James San Jacinto Mountains Reserve.

HELLERSTEIN, J., HONG, W., MADDEN, S., AND STANEK, K. 2003. Beyond average: Towards sophisticated sensing with queries. In *IPSN '03*. Vol. 1. Palo Alto, CA.

INTANAGONWIWAT, C., GOVINDAN, R., AND ESTRIN, D. 2000. Directed diffusion: A scalable and robust communication paradigm for sensor networks. In *Proceedings of the Sixth Annual International Conference on Mobile Computing and Networking*. ACM Press, Boston, MA, 56–67.

KANG, T. H., RHA, C., AND WALLACE, J. W. Seismic performance assessment of flat plate floor systems. CUREE-Kajima Joint Research Program.

KARP, B. AND KUNG, H. T. 2000. GPSR: greedy perimeter stateless routing for wireless networks. In *Proceedings of Mobicom*.

KOHLER, M. UCLA Factor Building.

KUBIATOWICZ, J., BINDEL, D., CHEN, Y., EATON, P., GEELS, D., GUMMADI, R., RHEA, S., WEATHERSPOON, H., WEIMER, W., WELLS, C., AND ZHAO, B. 2000. Oceanstore: An architecture for global-scale persistent storage. In *Proceedings of ACM ASPLOS*. ACM.

LI, X., KIM, Y.-J., GOVINDAN, R., AND HONG, W. 2003. Multi-dimensional range queries in sensor networks. In *Proceedings of the First ACM Conference on Embedded Networked Sensor Systems (SenSys)*. Vol. 1. to appear.

MAINWARING, A., POLASTRE, J., SZEWCZYK, R., CULLER, D., AND ANDERSON., J. 2002. Wireless sensor networks for habitat monitoring. In *ACM International Workshop on Wireless Sensor Networks and Applications*. Atlanta, GA.

MEGUERDICHIAN, S., KOUSHANFAR, F., POTKONJAK, M., AND SRIVASTAVA, M. 2001. Coverage Problems in Wireless Ad-hoc Sensor Networks. In *Proceedings of the IEEE Infocom*.

RAO, R. M. AND BOPARDIKAR, A. S. 1998. *Wavelet Transforms: Introduction to Theory and Applications*. Addison Wesley Publications.

RATNASAMY, S., FRANCIS, P., HANDLEY, M., KARP, R., AND SHENKER, S. 2001. A scalable content addressable network. In *Proceedings of the 2001 ACM SIGCOMM Conference*.

RATNASAMY, S., KARP, B., YIN, L., YU, F., ESTRIN, D., GOVINDAN, R., AND SHENKER, S. 2002. Ght - a geographic hash-table for data-centric storage. In *First ACM International Workshop on Wireless Sensor Networks and their Applications*.

ROWSTON, A. AND DRUSCHEL, P. 2001. Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility. In *18th ACM SOSP*. Vol. 1. Lake Louise, Canada.

VETTERLI, M. AND KOVACEVIC, J. 1995. *Wavelets and Subband coding*. Prentice Hall, New Jersey.

VITTER, J. S., WANG, M., AND IYER, B. 1998. Data cube approximation and histograms via wavelets,. In *Proceedings of CIKM'98*, D. Lomet, Ed. Washington D.C, 69–84.

WANG, W., YANG, J., AND MUNTZ, R. 1997. Sting: A statistical information grid approach to spatial data mining. In *Proceedings of the 23rd VLDB Conference*. Vol. 1. Athens, Greece.

WIDMANN, M. AND C.BRETHERTON. 50 km resolution daily preciptation for the Pacific Northwest, 1949-94, http://tao.atmos.washington.edu/data_sets/widmann/.

XU, N., RANGAWALA, S., CHINTALAPUDI, K., GANESAN, D., BROAD, A., GOVINDAN, R., AND ESTRIN, D. 2004. A wireless sensor network for structural monitoring. In *Proceedings of the Second ACM Conference on Embedded Networked Sensor Systems (SenSys)*.

XU, Y., HEIDEMANN, J., AND ESTRIN, D. 2001. Geography-informed energy conservation for ad hoc routing. In *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking (Mobicom)*. ACM, Rome, Italy, 70–84.

ZHAO, Y., GOVINDAN, R., AND ESTRIN, D. 2002. Residual energy scans for monitoring wireless sensor networks. In *Proceedings of the IEEE Wireless Communications and Networking Conference*.