An Evaluation of Multi-resolution Storage for Sensor Networks

Deepak Ganesan [†] Ben Greenstein [†] Denis Perelyubskiy [†]
Deborah Estrin [†] John Heidemann [†‡]

[†] Department of Computer Science, UCLA, Los Angeles, CA 90095
[†‡] USC/ISI, 4676 Admiralty Way, Marina Del Rey, CA 90292
{deepak,ben,denisp,destrin}@lecs.cs.ucla.edu, johnh@isi.edu

ABSTRACT

Wireless sensor networks enable dense sensing of the environment, offering unprecedented opportunities for observing the physical world. Centralized data collection and analysis adversely impact sensor node lifetime. Previous sensor network research has, therefore, focused on in network aggregation and query processing, but has done so for applications where the features of interest are known a priori. When features are not known a priori, as is the case with many scientific applications in dense sensor arrays, efficient support for multi-resolution storage and iterative, drill-down queries is essential.

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.

1. INTRODUCTION

Research in sensor networks has been targeted at numerous scientific applications, including micro-climate and habitat monitoring [1, 2, 3], earthquake and building health monitoring ([4]) and others. Such networks are primarily intended for long-term deployment, to obtain data about previously unobservable phenomena for detailed analysis by experts in the field. Data analysis in such applications often involves complex signal manipulation, including modeling, searching for new patterns or trends, looking for correlation structures etc. For instance, researchers interested in building health monitoring hope to be able to correlate changing vibration patterns of buildings to data about small earthquakes. Conventional approaches to such monitoring have involved wired and sparsely deployed networks that transfer all data from sensors to a central data repository for persistent storage.

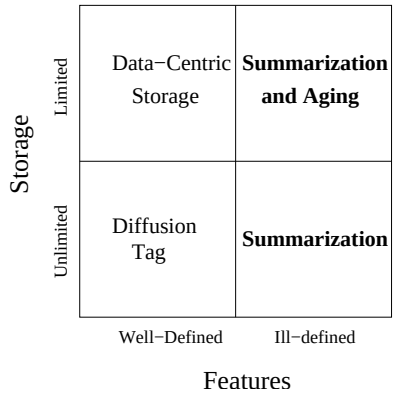


Figure 1: Feature Extraction in Sensor Networks

Advances in wireless sensor network technologies have opened up unprecedented opportunities for dense sensing of such phenomena. However, the goals of providing a non-invasive, in situ, dense, long-term deployment of sensing infrastructure has necessitated that sensor nodes be cheap, wireless, battery-powered, and consume very little power. An unfortunate consequence of the limited resources of such nodes is that they are highly communication constrained, severely limiting deployment lifetime if raw data were transmitted to a central location (Table 1,[5]). For instance, as shown in Table 1, a micro-climate monitoring network composed of motes can be expected to last at most a couple of months even with optimized communication schedules. Storage requirements adds an additional dimension to optimize in such systems, since the storage capacity of low-end sensor nodes (motes[6], smartdust[7]) will be limited by cost and form factor. Storage requirements of such systems are exacerbated by the fact that they are often envisaged to be long-lived, and operate unattended for many years. Non-volatile storage prices and form factor will no doubt reduce, yet, for long-lived sensor nodes, the disparity between the sensor data sizes, and the sizes of storage that they can be equipped with will continue to generate the need to optimize for storage.

Existing techniques offer schemes for applications where the features of interest and aggregation operators are well-defined (Figure 1). For instance, a Diffusion query suggested in [9] tracks the movement of a bird with known signature. Such an event detec-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profi t or commercial advantage and that copies bear this notice and the full citation on the fi rst page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specifi c permission and/or a fee.
SenSys'03, November 5–7, 2003, Los Angeles, California, USA.
Copyright 2003 ACM 1-58113-707-9/03/0011 ...$5.00.

Application	*Sensors*	*Expected Data Rates*	*Data Requirements/Year*	*Expected Lifetime using Centralized Data Collection*		*Expected Time to Storage Limit if all Raw data were stored locally at each node*	
				Mote	*MK-2*[8]	*Mote*	*MK-2*
Building Health Monitoring [4]	Accelerometer	30minutes seismic events per day per sensor	8Gb/year	few weeks	few months	few days	1 year
Micro-climate Monitoring	Temperature, Light, Precipitation, Pressure, Humidity	1 sample/minute/sensor	40Mb/year	few months	1 year	1 months	25 years
Habitat Monitoring	Acoustic, Video	10 minutes of audio and 5 mins of video per day	1 Gb/year	few weeks	few months	few days	8 years

Table 1: Data Requirement estimates for Scientific Applications

tion scheme can be augmented with in-network event storage using Data-Centric Storage (DCS [10]) such that related detections are stored at predefined locations in the network. TAG [11] provides SQL-like semantics to define aggregates on a data collection tree, so that operators at junctions can construct streaming data aggregates such as histograms. While these techniques are all three important to current and future sensing systems, they are not sufficient for all applications. Diffusion and DCS require that queries and associated processing be clearly defined a priori. While DCS provides a storage framework, it is more concerned with placement of named data at known locations, such that routing overhead is low for queries. Storage is much less of a concern, since event detections themselves are highly summarized representations of raw sensor data, and therefore, not likely to be storage intensive. TAG is a stream aggregation model, and relies on the knowledge of aggregate operators such that these can be placed at junctions of a data gathering tree. When details about queries and therefore the aggregation operators, are not known a priori, such a stream model is inappropriate.

Unlike querying frameworks that operate on known events, or proposals for a continuous streaming data aggregation service, our goal is to provide storage and search for raw sensor data in data-intensive scientific applications. Constructing a storage and search system that satisfies the requirements of data-rich, scientific applications is a daunting task; the storage requirements are massive, and little a priori information can be exploited to reduce these requirements. 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. A promising approach was introduced in [12], where multi-resolution summarization using wavelets, and drill-down querying was proposed. 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. Wavelet-based summarization can potentially benefit query processing in three ways: (a) it produces a compact representation of data that highlights interesting features such as long-term trends, edges and significant anomalies and is, therefore, useful for general purpose query processing; (b) once generated, many spatio-temporal queries can be satisfied with negligible communication overhead by drill-down querying; and (c) aging (discarding) summaries selectively gracefully degrades query performance over time in storage-constrained networks.

In this paper, we address the performance issues of this scheme. Specifically, given that storage is constrained, and communication is expensive, can we intelligently store and lookup summaries so as to maximize query accuracy? Our contribution in this paper is twofold:

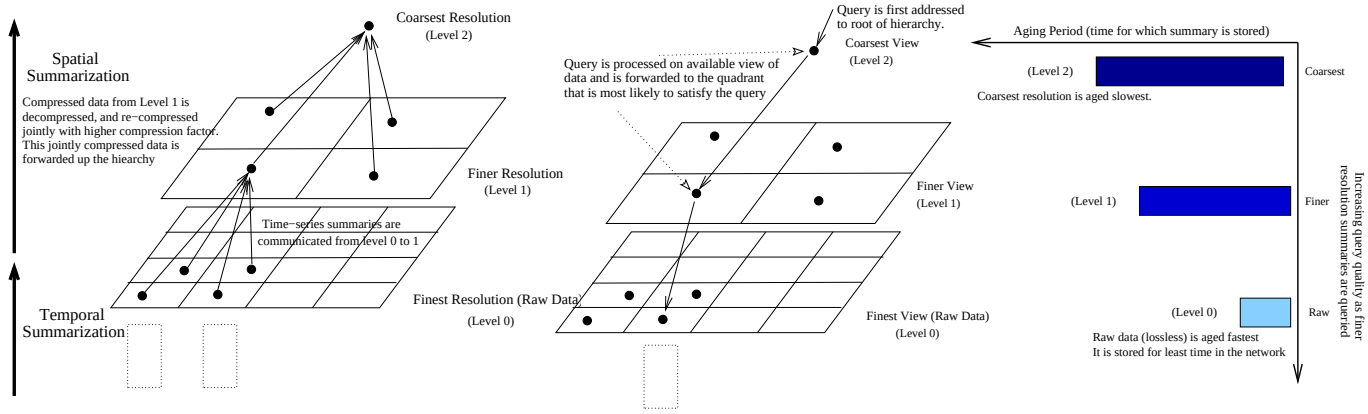
- We show that wavelet-based hierarchical summarization provides accurate responses to a broad spectrum of spatio-temporal queries with low communication overhead. Although a preliminary description of multi-resolution summarization was presented in ([12]), in this paper, we present comprehensive query surveys on an iPAQ-based implementation.
- Second, we show that in a storage-constrained environment, graceful query degradation over time is achieved by networked *aging* of summaries, such that more useful summaries are retained longer. To our knowledge, no one has examined data aging in the context of highly distributed sensor systems.

We present the design and implementation of DIMENSIONS on a linux platform. To evaluate the performance of our system, we consider queries posed on a geo-spatial dataset ([13]), which provides precipitation data from a medium-scale sparse sensor network.

2. DIMENSIONS ARCHITECTURE

We describe the architecture of our system in three parts: (a) the hierarchical wavelet processing that constructs lossy multi-resolution summaries, (b) the expected usage of these summaries through drill-down queries, and (c) the data aging scheme that determines how summaries should be discarded, given node storage requirements.

This architectural description makes some assumptions, such as grid placement of nodes, and a homogeneous network with nodes of similar capability. Such assumptions are not necessarily characteristic of sensor networks, on the contrary, irregular deployments



(a) Hierarchy Construction

(b) Drill-down Querying

(c) Long term storage

Figure 2: Temporal and Spatial summarization

may be frequently seen for reasons described in [14]. We discuss these assumptions in more detail in a discussion section at the end of this paper (Section 8).

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 ([1]) 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.

2.1 Multi-Resolution Summarization

Our goal is to construct a system 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 ([15, 16]). 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(a)) using wavelets involves two phases. The first phase, *temporal summarization*, is cheap since it involves only computation at a single sensor, and incurs no communication overhead. This step consists of each node compressing the time-series data by exploiting temporal redundancy in the signal. Significant benefit can be expected from merely temporal processing since a lot of sensor data is locally generated at each node.

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

2.2 Drill-Down Querying

Drill-down queries on distributed wavelet summaries can dramatically reduce the cost of search. The term 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. By restricting search to a small portion of a large data store, such queries can reduce processing overhead significantly.

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 2(b), where a drill-down query is forwarded to the quadrants that are most likely to satisfy it.

Such hierarchical query processing offers multiple benefits. First, when the target query result is sparsely distributed in a large network, query results can be obtained in a few steps($O(\log_4 n)$, where n is the number of nodes in the network). This benefit is provided by other approaches that construct quad-trees ([17]), however, these techniques can only answer a limited set of queries. Second, since the data is processed and stored once, the up front cost of storage can be amortized over multiple queries.

2.3 Networked Data Management

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 1), however, resources have to be allocated for storing new summaries by discarding older ones. The goal of networked data management in our system is to discard summaries such that network storage resources are efficiently utilized, and graceful quality degradation over time is achieved.

Two challenges need to be addressed to achieve long-term gracefully degrading storage. The first, simpler challenge, is to balance storage load among nodes in the network. Clearly, a simple hierarchical arrangement as shown in Figure 2(a) distributes load quite unevenly. For instance, the highest level clusterhead (level 2) is solely 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, 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. We address this topic further in Section 8.

Once we have such a load-balancing mechanism that ensures utilization of network storage capacity , we can ask the second, more challenging question. How long should a summary be stored in the network? In other words, how should we apportion 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 2(c), 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 2(c) 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. While the figure shows a global assignment of summaries for ease of explanation, the final goal of networked data management is to provide a storage partitioning between different summaries for each individual node such that the resultant global allocation matches user requirements. Thus, an algorithm to determine the age of summaries in the network has to weigh 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.

In this paper, we propose two storage allocation schemes: (a) an offline progressive data aging algorithm that operates on training data to determine system parameters for online operation, and (b) an online greedy algorithm.

3. 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.

As described earlier, 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.

Symbol	*Parameter*
S	Local storage constraint
$f(t)$	User-specifi ed 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 $i-1$
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 2: Parameters for the Aging Problem

3.1 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 at increasing spatial scales of processing. 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.

3.2 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 generages summaries of its own that are sent to level 3. We define s_i as the amount of data that a level $i+1$ clusterhead allocates for summaries from level i. Since four clusterheads at level i send r_i data to level $i+1$, the amount of storage-per-epoch for a clusterhead at level $i+1$ to store level i summaries is $4r_i$. The corresponding

storage requirement over the entire network is R_i, which is the total amount of data communicated from level i to level $i+1$.

3.3 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 2(c), 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 2(c), 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.

3.4 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 3 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 3, 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. The age of summaries generated at level i depends on two parameters: (a) the amount of storage apportioned at each level $i+1$ clusterhead for summaries from level i, s_i, and (b) the amount of data communicated from level i to level $i+1$, R_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 raw data, the age of raw data, $Age_{raw} = \frac{s_{raw}}{\gamma}$.

The cost function that we choose is the *instantaneous 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 4). The *minimum error aging problem* can, then, be defined as follows. Find the ages of summaries, Age_i, at different resolutions such that the the maximum instantaneous quality difference is minimized.

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

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.

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

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.

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

Additional Constraints: In formulating the above problem, we consider only drill-down queries and a network of homogeneous devices with identical storage limitations. These constraints might not always be true. For instance, queries may look at intermediate summaries directly, without drilling down. Previous research has proposed a tiered model for sensor deployments ([18]), where nodes at higher tiers have more storage and energy resources than nodes at lower tiers. We do not consider such constraints in this work, although some of these are straightforward extensions of our problem.

Solving the Constraint 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 4). 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

$$Min_{0 \leq i \leq n} \{\mu\} \tag{5}$$
$$q\text{diff}(Age_i) \leq \mu \; \forall \; i \tag{6}$$

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

3.5 Desirable Parameters

While the emphasis in this paper is on solving the aging problem, there are many parameters in the above formulation that impact the performance of our system. We briefly describe some of the key parameters and desirable values for these parameters before addressing the aging problem in more detail.

- *Data implosion:* A significant concern with any data aggregating tree is the implosion of data closer to the root. In our multi-resolution hierarchy, the extent of implosion is determined by the rate, r_i, which depends on the compression ratio (c_i) as shown in Equation 1. Since the numerator in Equation 1 grows exponentially with the level, slow growth in the compression ratio (c_i) will cause data implosion. An exponentially growing compression ratio can alleviate this concern.
- *Progressively improving query response:* The choice of compression rations, c_i also depends on the query quality (q_i)

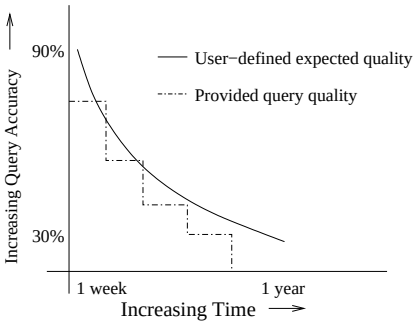


Figure 3: Providing graceful quality degradation by aging summaries

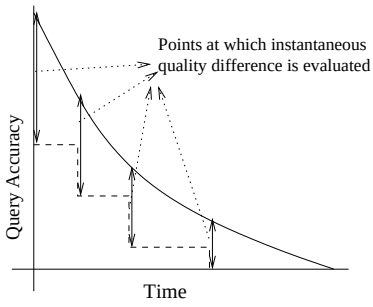


Figure 4: Objective Function

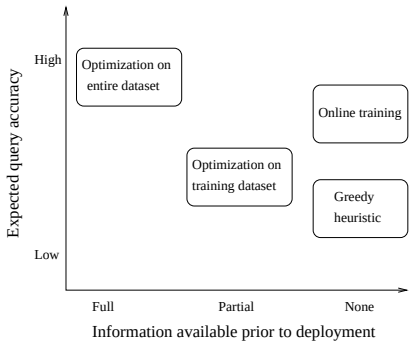


Figure 5: Aging algorithms that operate on different levels of a priori information.

provided by such a summary. For instance, consider an assignment of compression ratios where query quality does not change significantly between on levels. Such an assignment is not useful, since drill-downs cannot improve query quality. An effective choice of compression ratios would be one where query quality progressively increases as a query drills down the hierarchy. Such a choice makes it more effective to progressively age summaries as shown in Figure 2(c).

- *Load-balancing rate:* A parameter that we ignore in the above treatment is the rate of load-balancing, *i.e.*,, how frequently must the system choose an alternate clusterhead to store data. This load-balancing rate depends on the local storage allocated to each level, s_i. For instance, if s_i is four epochs, then the load-balancing rate for a level i clusterhead should be at least once in four epochs. If not, recent data that is merely 4 epochs old is aged out of the system.

4. CHOOSING AN AGING STRATEGY

The constraint-optimization problem presented in Section 3 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 5 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].

Distributed scenarios such as wireless sensor networks have to operate with less information due to the overhead of centralized data collection (Table 1). In some scientific applications, a data gathering phase might precede full-fledged deployment (e.g.: James Reserve [1]), potentially providing training datasets. In other cases, there might be available data from previous wired deployments (e.g.: Seismic Monitoring [4]). 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, [19] 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.

The intent of this study is to see how to 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.

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

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 3;

Algorithm 1: Omniscient Algorithm Pseudocode

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

[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.

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	8
2 (coarsest)	8	32	128	64

Table 3: Example of a greedy algorithm for a 16 node network

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.

Data : w_i: Weight for query type i;
E_i: Error for query type i from the training set
Result : Assignment of number of summaries of each level to store locally
/* Evaluate weighted cumulative error over all query types */
$q_i = \frac{\sum_{query\ types} w_i E_i}{Number\ of\ query\ types}$;
Solve constraint-optimization in Section 3;

Algorithm 2: Training Algorithm Pseudocode

Greedy Algorithm. We now describe a simple greedy procedure that can be used in the absence of prior datasets. This procedure assigns weights to summaries according to a measure of expected importance of each resolution towards drill-down queries, represented by the parameter *resolution bias (β)*. Algorithm 3 shows this 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 3), 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. This parameter is used to control how gradually we would like the step function (in Figure 3) to decay.

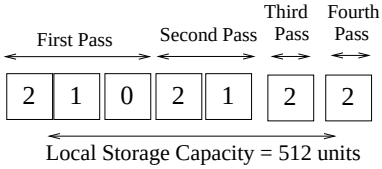


Figure 6: Local Resource Allocation using the Greedy Algorithm ($\beta = 1$). 4 coarsest, 2 finer, and 1 finest summaries are allocated

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, which translates the local storage time period to networked storage time period.

Data : S: local storage capacity;
N: number of nodes in the network;
k: number of levels;
r_i: the size of a summary at level i
Result : Assignment of number of summaries of each level to store locally
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;

Algorithm 3: Greedy Algorithm Pseudocode

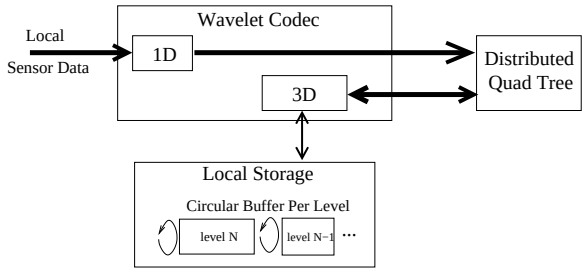
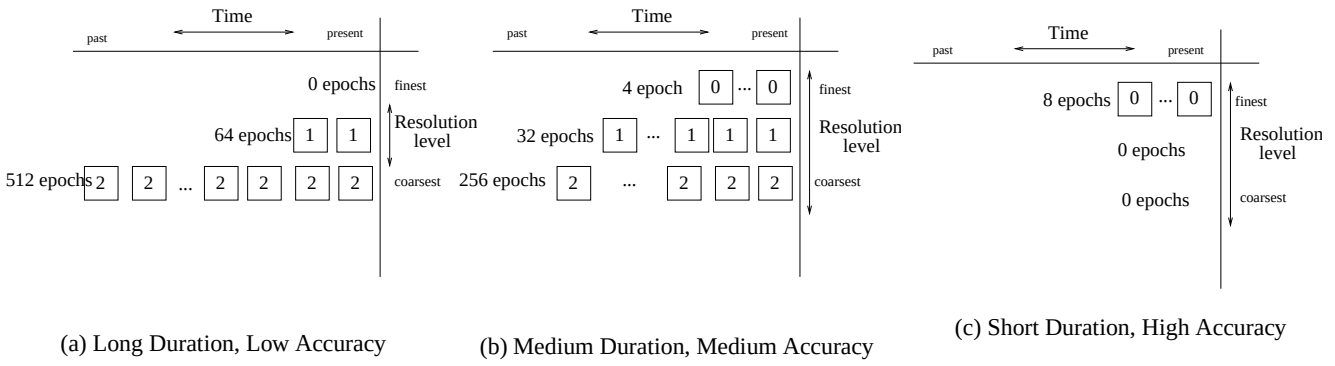


Figure 8: Implementation Block Diagram

For instance, consider a greedy allocation with resolution bias of 1 in a 64-node network with parameters provided in Table 3. 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 2. Consider an instance where the local storage capacity is 512 units and the sizes of each summary are as shown in Table 3. The greedy allocation scheme allocates summaries starting with the coarsest level as shown in Figure 4. 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 3 on Equation 3. For instance, Age_0 is $\frac{4^0 s_0}{r_0} = \frac{256}{64} = 4$ epochs.
This aging sequence is shown in Figure 7(b). 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 7(a) shows an allocation that balances favors duration over detail, whereas the allocation in Figure 7(c) favors detail over duration.

5. SYSTEM IMPLEMENTATION

In this section, we describe the implementation of DIMENSIONS on a linux-based network emulation platform ([20]). There are three major components to our system (shown in Figure 8): the wavelet transform coding used to construct the summaries, the local storage implementation that allocates storage to summaries from different levels, and the distributed quad-tree, which provides a system abstraction to support hierarchical storage and drill-down search.



(a) Long Duration, Low Accuracy

(b) Medium Duration, Medium Accuracy

(c) Short Duration, High Accuracy

Figure 7: Different global resource allocations that can result from a local allocation procedure.

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 4: Spatio-temporal queries posed on Precipitation Dataset

Wavelet Codec. The wavelet codec software is based on a high-performance transform-based image codec for gray-scale images (freeware written by Geoff Davis ([21]). 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.

Local Storage. Local storage is implemented as circular buffers over berkeleydb (standard in glibc). Each summary is indexed by the level, quadrant, and epoch to which it corresponds. Every query on a summary involves first decompressing the summary, and then processing the query on the summary.
The parameters of the local storage are determined using an aging strategy, that partitions the local storage between different summaries.

Drill-down Queries. Our implementation considers four types of queries (shown in Table 4) that involve different extents of spatio-temporal processing that evaluate both advantages and limitations of wavelet compression. 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 GlobalYearlyMax query looks at the maximum yearly precipitation in the entire network, while the GlobalDailyMax queries for the daily global maximum. Wavelet processing does not preserve maxima very well in practice.
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 cluster-head, which has a summary covering the spatial extent of the entire network. The cluster-head 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.

Aging function. For our current implementation, we make two relaxing assumptions for the aging function ($f(t)$) that make the constraint-optimization problem described in Section 3 easier to solve, (a) $f(t)$ is monotonically decreasing, thus *q*diff needs to be evaluated only at a few points, and (b) $f(t)$ is linear, thus standard linear optimization tools such as *lp_solve* can be used. Both of these restrictions are not fundamental, and can be changed depending on the problem being studied. Our choice merely represents a possible choice of aging functions rather than an exhaustive list.
In this study, we consider linear aging functions of the form,

$$f(t) = 1 - \alpha t \tag{7}$$

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.

Distributed Quad Tree. The local storage partitioning discussed in the previous section allocates some portion of the storage at each node to summaries for each level. The Distributed Quad Tree (DQT) describes the routing and cluster-head selection scheme that assigns summaries to each node in the network such that the allocated networked storage is appropriately utilized.
The DQT is a routing overlay that decomposes the network space hierarchically into quadrants, and assigns a rendezvous node to each quadrant in the manner suggested by GHT's structured replication ([22]). By using a geographical hash function, a globally shared convention for selecting nodes, the construction of the tree overlay requires no setup communication, eliminating the overhead

of clusterhead election and global consensus. In this sense, it is a good choice for energy constrained sensor networks.
GHT specifies a geographic hashing scheme that translates a globally known constant and a bounding rectangle into geographic coordinates. It relies on an underlying geographically informed routing protocol that understands the semantics "route to the node that is geographically closest to location (x,y)" to select and deliver messages to rendezvous nodes. The extension of GPSR that is presented in [22] is such a protocol.
Our implementation differs from the above in two ways. First, in our current implementation, we used a link-state geographic routing protocol enhanced to support GPSR semantics. Since the population of link-state routing tables require the dissemination of each node's link state to every other, it may provide adequate routing for relatively small networks, but it is not a scalable solution for the long run. We therefore plan to port the GPSR and its GHT enhancement to our architecture. Second, DQT supports a time varying version of the geographic hash so that the particular nodes chosen for rendezvous in the DQT overlay change at a specified frequency.

6. EXPERIMENTAL EVALUATION

Since available dense wireless sensor network datasets lack sufficient temporal and spatial richness , we use a geo-spatial precipitation dataset [13] for our current performance studies. This dataset provides a 15 x 12 grid of daily precipitation data for forty five years, where adjacent grid points are 50 kilometers apart. Both the spatial and temporal sampling are much lower than what we would expect in a typical sensor network deployment (Table 1). While the data sizes and scale would be expected to be larger in practice, this dataset has edges and exhibits spatio-temporal correlations, both of which are useful to understand and evaluate our algorithms. 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 ([1]). In all our experiments, we replayed this dataset.
Since the spatial scale of the dataset is low, it is not feasible to use wavelet processing along the spatial axis. In practice, a grid of size, at least 30x30 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.

6.1 Communication Overhead

Communication overhead over a multi-resolution hierarchy is governed by the rates, r_i, that are determined as shown in Equation 1. The problem of selecting optimal rates, r_i, for a particular dataset is outside the scope of this paper. 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 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 (Table 1), where nodes generate more data, the epoch would be much shorter.

Hierarchy level (i)	*Num Clusterheads (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 5: Communication Rate per Level

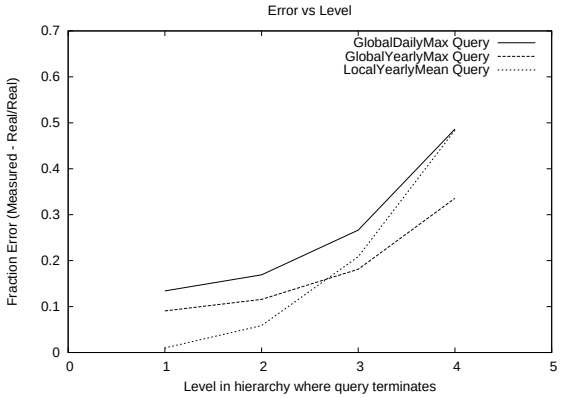


Figure 9: Query error decreases as they drill-down to lower levels of hierarchy. Summaries at lower levels typically contribute less to reducing query error than higher level summaries.

- $c_0 : c_1 : c_2 : c_3 = 6 : 12 : 24 : 48$. As described in Section 3.5, 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 5. 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 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 4 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 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 6.2 shows the variation of query quality for queries defined in Table 4 for different levels of drill-down.

Performance for LocalYearlyMean, GlobalYearlyMax and LocalDailyMax queries are very similar, as shown in Figure 6.2. All of them have an error of 40-50% if only the coarsest (level 4) summaries are queries, but reduce rapidly to almost 0% 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 6.2, 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.

These results demonstrate a key advantage of multi-resolution storage. While there is an initial overhead of communicating summaries (described in Section 6.1), this overhead can be amortized over many queries posed by the users.

6.3 Performance of Aging Strategies

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 6.2), we will assume that nodes store only summaries at various levels and not raw data.

The parameter, α, in Equation 7 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).

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 7: 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.

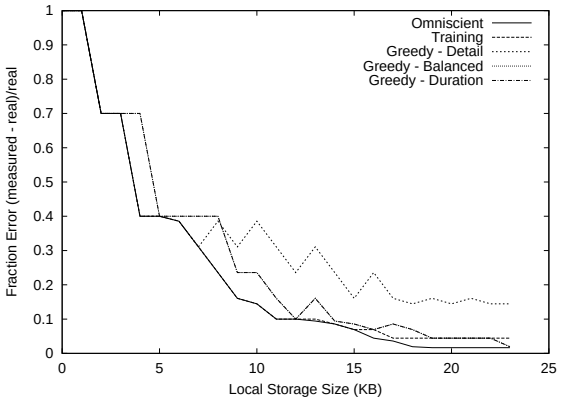


Figure 10: Comparison of Omniscient, Training and Greedy strategies for GlobalYearlyMax query($\alpha = 0.002$)

Omniscient Strategy: Establishing a Lower Bound for Query Error

The omniscient scheme uses the query error for each query on the entire dataset (Figure 6.2) to determine the optimal choice of aging parameters for each query type. As shown in Table 6, 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 10, 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 networks composed of nodes with low local storage capacities, the error is high since only the coarsest summaries can be stored in the network .

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 4 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

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 6: Comparing the error in Omniscient (entire) Dataset vs Training (first 6 years) Dataset

dataset? Table 6 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 7 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 4). 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 10, that shows how the performance varies as the amount of storage allocated to a node is increased. Figure 10 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.

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 7, 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 10. 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.

Scheme	*Storage per node*		*Communication per node*	
	Avg	StdDev	Avg	StdDev
Fixed Hierarchy	0.045	0.022	0.0154	0.014
Load-balanced Hierarchy	0.015	0.004	0.015	0.006

Table 8: Comparison of Load-balanced hierarchy to fixed hierarchy

6.4 Performance of the Distributed Quad-Tree

To quantify the benefit provided by our load-balanced DQT implementation, we compare the multi-hop communication and storage overhead of propagating data up the tree using rotating hash nodes against having nodes fixed at the centers of their covering rectangles. The study was done in the Emstar simulation framework ([20]), on a 8x8 grid topology, where each node has at most 8 neighbors. Each node generates the amount of data provided in Table 5, and picks a different hash location periodically as described in Section 5. Root nodes rotate at a rate of once every 10 epochs. A tree node i levels below the root rotates at a frequency 2^{-i} times that of the root.

Table 8 shows that that a load-balanced hierarchy reduces storage used per node by a factor of three, while having similar communication requirements as a fixed hierarchy. In addition, the standard deviations for both communication and storage reduce approximately by a factor of 2, showing that the scheme balances these parameters among nodes well. Communication load for the load-balanced scheme is not entirely balanced (standard deviation is non-zero) because of limitations of a finite grid, and nodes in the middle of the grid (or at the center of any ad-hoc network), being subjected to more multi-hop communication overhead than other nodes. In an longer simulation on a larger scale network, we would expect the communication load to be more uniformly balanced in the network. Similarly, in the case of storage overhead, the hash function is probabilistic, and there is a finite probability of the same node being

selected more (or less) than the expected number of times. In an irregular network, storage balancing would depend on the spatial density of node deployment as well, a problem that we are pursuing as part of future work.

How does this impact aging parameters of the system? Consider an example where a node were to store summaries for a certain level for 8 epochs. If the frequency of rotating clusterheads were chosen to be once every 8 epochs, there is a finite probability that the same node is hashed to twice within the aging period, thus, resulting in the loss of summaries. Such a situation can be mitigated by choosing the rotation period to be faster than the number of summaries stored at each node. For instance, if the frequency of rotation is once every 2 epochs, then, better storage balancing can be achieved. In practice, since nodes will store many summaries, faster rotation will also balance communication load better.

7. RELATED WORK

We briefly describe the rationale behind our design choice to use wavelets, and proceed to describe a broad spectrum of related work.

Survey of Data Compression and Representation Techniques. Techniques for data and signal compression abound. Typical lossless data compression techniques include Huffman, arithmetic encodings, the Lempel-Ziv coder, etc. These techniques reduce data by a factor of two to ten in practice, and rarely provide the compression ratios of hundreds that we would like in our system. Lossy schemes, on the other hand, can be tuned to compress data to fit communication requirements in sensor networks.

Wavelets decompose the data recursively into various time and frequency scales, and provides a good representation of both time and frequency content in a signal. This flexibility has been the primary reason behind the popularity of wavelets in signal and image compression, time-series data mining, and approximate querying [23, 24]. Wavelet compression tends to preserve interesting features such as edges and succinctly captures long-term behavior, and consequently provides a data representation ideal for spatio-temporal querying. Its benefit for edge and boundary detection stems from two properties: (a) subband coding preserves sharp changes at various scales, thus providing a good representation of edges and (b) multi-scale techniques are useful to detect edges that may be visible at different spatio-temporal scales.

The easily distributable nature of wavelet transforms and the compact support is also exploited in [25] to address the sensor broadcast problem. While the reasons for using wavelets are similar, this work addresses a different problem from ours.

Distributed Data Storage. Distributed databases and data storage have been extensively studied in the context of the wide-area Internet. Aging of web-pages is particularly important for web-cache performance and has, therefore, been explored by researchers (eg: [26]). While similar in motivation, these systems differ from ours in many significant respects: (a) web pages are not known to exhibit spatial correlations, (b) they are designed for a much more resource-rich infrastructure, and (c) bandwidth, while limited, is a non-depletable resource unlike energy.

Data Centric Storage (DCS [10]) is a system that extends primitives used in the peer-to-peer Content Addressable Networks system to provide a geographic hash table (GHT [22])-based high-level event storage in sensor networks. DCS assumes that an event description exists a priori, and therefore, event detections can computed in a distributed manner within the network. The system is responsible for storing these detections (called observations) in a distributed framework for easy access. DIMENSIONS is designed for data mining, and does not assume a priori knowledge of event signatures. Thus, it involves mining massive spatio-temporal datasets, which is not the focus of DCS. The distributed storage aspects of our system have similarities to the techniques used in DCS, and we extend this scheme in our DQT implementation.

Data Mining. Geographic Information Systems (GIS) deal with data that exhibits spatio-temporal correlations, but the processing is centralized, and algorithms are driven by the need to reduce search cost, typically by optimizing disk access latency. Some of these approaches ([23, 24, 27]) propose the construction of wavelet synopses, for fast processing of range-sum queries. Many of the techniques proposed for approximate querying and data mining have informed the coding techniques that we choose in our system. An interesting example of a distributed infrastructure for wide-area data mining of sensor data is Astrolabe [28], which proposes a hierarchical data mining approach that is similar in motivation and design to DIMENSIONS. Astrolabe hierarchically aggregates data, and uses a drill-down approach. There are three key differences between Astrolabe and our work: (a) Astrolabe is a generic platform, and does not specify an aggregation function to use, while our system is built upon being able to do multi-resolution wavelet processing in a distributed framework. (b) Astrolabe is designed for the wide-area Internet and therefore assumes less stringent bandwidth constraints. It uses bimodal multicast as the communication framework, which ensures very high reliability, but incurs high communication overhead. (c) The storage constraints are not as stringent in a wired network and are, therefore, not addressed.

Sensor Network Databases. In recent years, many database mechanisms have been extended to sensor network data querying. Systems such as Diffusion [9], Cougar [29] and TinyDB [11] use in-network processing techniques. Diffusion provides a general framework for routing and in-network processing. It's query mechanism is simply to flood an interest to all network nodes. Its efficiency comes in the way it is designed to process matching data in a hop-by-hop fashion on the return path, potentially reducing redundancy while transforming the data from raw time series information into a more compact and semantically richer representation. When multiple data sources respond to a query, Diffusion exploits the property that data generated from spatially proximal nodes is likely to be correlated. While Diffusion takes a dynamic approach to in-network processing, where data can be combined anywhere along the routing path, Cougar proposes a static model, where a centrally computed query plan decides where to place joins within the network. Finally, TinyDB provides a platform for resource-constrained devices, and provides a programming interface to create new queries, and inject them into the network. The above approaches operate under the assumption that event description is known a priori, and that queries are explicitly defined for these event descriptions. Our system is designed to look for and find patterns in data.

An extension to TinyDB, [30], suggests the use of wavelets for constructing histogram summaries of sensor data. Histograms lose information of the temporal and spatial instant when a data sample was collected. Thus, while such an approach can be efficient when the goal is to gather a histogram of sensor data, it would be ineffective for our objective *i.e.*, to look for events at different granularities, and multi-resolution data collection.

Quantization. The aging problem that we discuss is similar in spirit to non-uniform quantization that is ubiquitous in signal processing. Given a finite number of bits to represent each sample, the goal of non-uniform quantization is to construct a step function such that an error metric is minimized. The aging problem that we discuss is an instance of quantization applied in a distributed framework, and to a specific dataset.

8. DISCUSSION

Our work is preliminary, and there are many aspects that need to be considered such as the impact of irregularity, considering correlation statistics in storage overhead, and finer progressive aging.

Impact of irregularities. Irregularities in spatio-temporal sampling of data impacts numerous aspects of our design. Along the spatial axis, our architectural description currently assumes that nodes in the network are arranged in a grid, or otherwise uniformly deployed. Along the temporal axis, data from different areas in the network would need to be synchronized before our techniques can be used. While techniques such as [31] and [32] can be used to create a globally synchronized timebase, such a mechanism might not always be available or cost-effective. Two aspects of our system would need to be modified in the event of irregular spatio-temporal deployment:

- The wavelet processing approach that we use currently operates on regularly spaced datasets. We are exploring the use of interpolated wavelet processing ([33]) to relax this assumption.
- The simple load-balancing mechanism that we use currently works well on uniformly deployed networks, but clearly needs to be extended in an irregular spaced deployment. As we describe in [14], skewed storage load distribution would result in the irregular case.

Data Correlation Statistics. In this work, we make the assumption that summaries at the same level are of equal size.
We impose two restrictions on the creation and storage of summaries in our current work that might change in future versions. First, we assume that all summaries at the same level are of equal size. While this constraint has the advantage of making communication balanced between different parts of the network, this may not be ideal for query accuracy, since it ignores the actual variations in the data in different regions of the network. In future work, we will consider error-based configurations, that look at correlation statistics to determine the amount of data to be transmitted at each level.

Finer Progressive Aging. We assume that summaries are not progressively degraded by nodes that store them. This restriction can be relaxed, and can result in more graceful quality degradation by degrading in smaller steps. Such a mechanism can be introduced quite easily into the aging strategies that we discuss in this paper.
We do not discuss how to determine the ideal hierarchy depth for a particular application. A hierarchy with a large number of levels might not be useful since the high-level summaries might be too coarse for useful query processing. For medium scale networks, such as the one that we consider, the number of levels ($\log_4 N$) is presumably low (less than 5). For a much larger network, this question would need to be addressed.

9. CONCLUSION

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 studying the query accuracy for a variety of queries on a wide-area precipitation sensor dataset.
A significant contribution of this work is extending our system to storage-constrained large-scale sensor networks. We use a combination of progressive aging of summaries, and load-sharing by cluster-rotation to achieve long-term query processing under such constraints. 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. We demonstrate the load-sharing properties of our system in a sensor network emulator. In our future work, our primary focus will be deploying our system in a real sensor network deployment scenario.

Acknowledgements

This work was made possible with support from the NSF STC program (award number 0120778). We would like to specially thank Stefano Soatto and members of the UCLA LECS laboratory for very useful insight on different aspects of our system. We would also like to thank our Sensys shepherd, Sergio Servetto, and anonymous reviewers for very useful suggestions to improve our work.

10. REFERENCES

[1] Michael Hamilton. James San Jacinto Mountains Reserve.
[2] P. Juang, H. Oki, Y. Wang, M. Martonosi, L. Peh, and D. Rubenstein. Energy-efficient computing for wildlife tracking: Design tradeoffs and early experiences with zebranet. In *ASPLOS*, San Jose, CA, October 2002.
[3] Alan Mainwaring, Joseph Polastre, Robert Szewczyk, David Culler, and John Anderson. Wireless sensor networks for habitat monitoring. In *ACM International Workshop on Wireless Sensor Networks and Applications*, Atlanta, GA, September 2002.
[4] Monica Kohler. UCLA Factor Building.
[5] A.A. Abidi, G.J. Pottie, and W.J. Kaiser. Power-conscious design of wireless circuits and systems. *Proceedings of the IEEE*, 88(10):1528–45, October 2000.
[6] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler, and Kristofer Pister. System architecture directions for networked sensors. In *Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-IX)*, pages 93–104, Cambridge, MA, USA, November 2000. ACM.
[7] J.M. Kahn, R.H. Katz, and K.S.J. Pister. Next century challenges: mobile networking for Smart Dust. In

Proceedings of the fifth annual ACM/IEEE international conference on Mobile computing and networking, pages 271–278, 1999.

[8] Mani Srivastava (UCLA) Andreas Savvides (UCLA). Medusa MK-2 Node.

[9] John Heidemann, Fabio Silva, Chalermek Intanagonwiwat, Ramesh Govindan, Deborah Estrin, and Deepak Ganesan. Building efficient wireless sensor networks with low-level naming. In *Proceedings of the Symposium on Operating Systems Principles*, pages 146–159, Chateau Lake Louise, Banff, Alberta, Canada, October 2001. ACM.

[10] S. Ratnasamy, D. Estrin, R. Govindan, B. Karp, L. Yin S. Shenker, and F. Yu. Data-centric storage in sensornets. In *ACM First Workshop on Hot Topics in Networks*, 2001.

[11] Samuel Madden, Michael Franklin, Joseph Hellerstein, and Wei Hong. Tag: a tiny aggregation service for ad-hoc sensor networks. In *OSDI*, volume 1, Boston, MA, 2002.

[12] Deepak Ganesan, Deborah Estrin, and John Heidemann. Dimensions: Why do we need a new data handling architecture for sensor networks? In *First Workshop on Hot Topics in Networks (Hotnets-I)*, volume 1, October 2002.

[13] M. Widmann and C.Bretherton. 50 km resolution daily preciptation for the Pacific Northwest, 1949-94, http://tao.atmos.washington.edu/data_sets/widmann/.

[14] Deepak Ganesan, Sylvia Ratnasamy, Hanbiao Wang, and Deborah Estrin. Coping with irregular spatio-temporal sampling in sensor networks. Technical report, Univ. of CA, Los Angeles, Dept. of Computer Science, 2003. CENS Technical Report 019.

[15] R. M. Rao and A. S. Bopardikar. *Wavelet Transforms: Introduction to Theory and Applications*. Addison Wesley Publications, 1998.

[16] M. Vetterli and J. Kovacevic. *Wavelets and Subband coding*. Prentice Hall, New Jersey, 1995.

[17] Xin Li, Young-Jin Kim, Ramesh Govindan, and Wei Hong. Multi-dimensional range queries in sensor networks. In *Proceedings of the First ACM Conference on Embedded Networked Sensor Systems (SenSys).*, volume 1, 2003. to appear.

[18] Alberto Cerpa et al. 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*, April 2001.

[19] Chalermek Intanagonwiwat, Ramesh Govindan, and Deborah Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks. In *Proceedings of the Sixth Annual International Conference on Mobile Computing and Networking*, pages 56–67, Boston, MA, August 2000. ACM Press.

[20] Jeremy Elson et al. EmStar: An Environment for Developing Wireless Embedded Systems Software. Technical report, Univ. of CA, Los Angeles, Dept. of Computer Science, 2003. CENS Technical Report 009.

[21] Geoff Davis. Wavelet Image Compression Kit.

[22] S. Ratnasamy, B. Karp, L. Yin, F. Yu, D. Estrin, R. Govindan, and S. Shenker. Ght - a geographic hash-table for data-centric storage. In *First ACM International Workshop on Wireless Sensor Networks and their Applications*, 2002.

[23] J. S. Vitter, M. Wang, and B. Iyer. Data cube approximation and histograms via wavelets,. In D. Lomet, editor, *Proceedings of CIKM'98*, pages 69–84, Washington D.C, November 1998.

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

[25] S. D. Servetto. Sensing lena—massively distributed compression of sensor images. In *Proceedings of the IEEE International Conference on Image Processing (ICIP)*, 2003.

[26] Edith Cohen and Haim Kaplan. Aging through cascaded caches: Performance issues in the distribution of web content. In *ACM Sigcomm*, 2001.

[27] Phillip B. Gibbons and Yossi Matias. Synopsis data structures for massive data sets. *DIMACS: Series in Discrete Mathematics and Theoretical Computer Science: Special Issue on External Memory Algorithms and Visualization*, A, 1999.

[28] Robbert van Renesse, Kenneth Birman, and Werner Vogels. Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. In *ACM Transactions on Computer Systems (TOCS)*, September 2001.

[29] Philippe Bonnet, Johannes Gehrke, Tobias Mayr, and Praveen Seshadri. Query processing in a device database system. Technical Report TR99-1775, Cornell University, October 1999.

[30] Joseph Hellerstein, Wei Hong, Samuel Madden, and Kyle Stanek. Beyond average: Towards sophisticated sensing with queries. In *IPSN '03*, volume 1, Palo Alto, CA, 2003.

[31] Jeremy Elson, Lewis Girod, and Deborah Estrin. Fine-grained network time synchronization using reference broadcasts. In *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation*, Boston, MA, 2002.

[32] Richard Karp, Jeremy Elson, Deborah Estrin, and Scott Shenker. Optimal and global time synchronization in sensornets. Technical report, Univ. of CA, Los Angeles, Dept. of Computer Science, 2003. CENS Technical Report 012.

[33] I. Daubechies, I. Guskov, P. Schröder, and W. Sweldens. Wavelets on irregular point sets. *Phil. Trans. R. Soc. Lond. A*, 357(1760):2397–2413, 1999.