Sharing and Exploring Sensor Streams over Geocentric Interfaces

Liqian Luo, Aman Kansal, Suman Nath, and Feng Zhao
Microsoft Research, One Microsoft Way, Redmond, WA 98052
{liqian, kansal, sumann, zhao}@microsoft.com

ABSTRACT

We present SenseWeb, an open and scalable infrastructure for sharing and geocentric exploration of sensor data streams. SenseWeb allows sensor owners to share data streams across multiple applications and users, thus amortizing sensor deployment costs effectively. It also provides mechanisms to transparently index and cache data, to process spatio-temporal queries on real-time and historic data, and to aggregate and present results on a geocentric web interface. In this paper, we present the architecture of SenseWeb, its techniques to enable global sharing of heterogeneous sensors, and its map-based front-end for spatio-temporal data exploration. We enable interactive geocentric data exploration in the map-based front-end using techniques for rapidly changing map overlaid visualizations of numerous data streams. We also demonstrate flexibility and scalability of the architecture by evaluating a deployed prototype of SenseWeb, which has been publicly available since March 2008.

Categories and Subject Descriptors

C.4 [**Computer Systems Organization**]: Performance of Systems; C.2.4 [**Computer Systems Organization**]: Computer Communication Networks—*Distributed Systems*

General Terms

Algorithms, performance, scalability, measurement

Keywords

Sensor networks, peer produced, geocentric interface

1. INTRODUCTION

We present *SenseWeb*, an open infrastructure for sharing and geocentric exploration of sensor data streams. It incorporates diverse sensor networks, such as sensor motes, webcams, and weather stations, deployed by independent entities and enables applications to use them as a single larger system with significantly enhanced capabilities than any of the individual components. Its web-based front-end, called *SensorMap*, enables users such as environmental scientists to visualize the sensor data steams on an interactive map, providing efficient data exploration for discovery of spatial and temporal correlations among the streams.

The sharing approach, sometimes referred to as peer production [7], has been found successful in various systems such as the Wikipedia, Linux, or Web 2.0 applications where multiple contributors each build a small component with some individual utility, but the system as a whole has much greater utility and enables a larger number of applications than those supported by the individual components.

The current generation of sensor network deployments are typically monolithic, concentrated in a single area, dedicated to a single set of applications, and maintained by a single team [19, 20]. While occasionally sharing their data, monolithic deployments are not designed to make their resources re-usable by other systems or concurrently used by multiple entities. The peer production approach can provide several unique advantages over these deployments in terms of providing larger spatio-temporal coverage, sharing resources opportunistically, and combining multiple sensing modalities.

In such a peer-produced, large-scale sensor-sharing system, techniques for efficient data exploration are necessary to help users to locate and identify particular sensor streams of interest. Traditional information sharing systems, such as the Internet, serve content that is human created, therefore directly indexable and searchable using keywords or tags. However, sensor streams consist of numerical or binary data automatically collected by sensors, which are therefore harder to explore or search using text-based approaches. The goal of sensor data exploration also differs from that of Internet exploration. Unlike text based content, where users wish to find individual relevant websites, users exploring sensor data are not typically interested in finding a single sensor stream but in finding an environmental phenomenon observed by multiple sensors. Such environmental phenomena are usually composed of multiple snapshot observations across multiple senors dispersed in the space. In order to facilitate the discovery of such spatio-temporal phenomena, SenseWeb adopts a geocentric approach. Its data exploration front-end, called SensorMap, presents sensor streams visually on a map-based interface.

Using SensorMap, users can zoom in, zoom out or pan the map to locate sensor streams in different geographical regions. Users can also select any sensors of interest and view their temporal distributions and correlations in time-

ACM GIS '08, November 5-7, 2008. Irvine, CA, USA

series charts. A third major feature of SensorMap is online contour generation. A user can select any type of numerical sensors and request for a contour map to view their spatial distribution. As the user traverses to different geographical regions using the map-based interface, contour maps of the sensor streams in view are dynamically generated on the fly, allowing the user to quickly explore their spatial correlations.

Large-scale sensor sharing and exploration imposes many unique research challenges, of which we consider two in this paper. First, embedded sensors are heterogeneous in terms of resource capabilities, mobility, network connectivities, and administrative boundaries. Such heterogeneity should be hidden from sensing applications as much as possible by a simple and homogeneous abstraction, to help them seamlessly take advantage of multiple relevant deployments. An open and scalable architecture is preferred for flexible use across multiple application domains.

Secondly, the system should serve temporal and spatial visualization requests in real-time to enable interactive exploration of a large collection of evolving sensor data. Particularly, spatial visualizations based on contour maps require online interpolation of values at unobserved locations as sensor readings are sparsely dispersed. Such interpolation often incurs large computational cost, and therefore incurs intolerable delays even for a small set of sensors. Addressing these two requirements becomes extremely challenging when the infrastructure must be scalable to a large collection of sensors, be extensible to support new sensor types, and be highly responsive to visualization requests.

By tackling the two challenges, the paper contributes in the following ways:

- We provide an open and scalable back-end architecture that enables sharing of heterogeneous sensors. It proposes the use of remote sensor gateways to host sensor data streams, thereby federating a distributed, Internet-scale sensing system. Moreover, it provides a tree-based type system, enabling customization of sophisticated sensor types.

- We present techniques[1] for interactive geocentric data exploration in a map-based front-end. It provides snapshot, temporal, and spatial views of sensor streams, overlaid on top of the map. For scalability, it caches computationally expensive visualizations derived from sensor data, and efficiently reuses the relevant portions based on overlapping regions among queries. The proposed techniques leverage the spatio-temporal localities in data exploration to enable rapidly changing spatial visualizations of numerous streams.

- We demonstrate the effectiveness of our techniques by evaluating them with a prototype SenseWeb deployment. The techniques were applied to an existing prototype in March 2008. We use a real-world workload from Live Search Maps and other appropriate data to illustrate the efficacy of our approach.

Our current prototype (available at `http://atom.research.microsoft.com/sensewebv3/sensormap`) already incorporates a large heterogeneous collection of sensors including wireless motes, weather stations, traffic sensors, rain meters, and web cameras. Several groups of environmental scientists from different institutions including EPFL Switzerland, NTHU Taiwan, and NTU Singapore, have been using the prototype system to share their sensor deployments. We report our experience with this public deployment in Section 5.

The rest of the paper is organized as follows. In Section 2, we discuss the design challenges. Section 3 presents the architectural design to enable sharing of heterogeneous sensors. Section 4 describes the geocentric interface for data exploration, focusing on the underlying algorithms to support scalable contour visualizations. Evaluations of our proposed techniques as well as the deployed prototype are presented in Section 5. Section 6 summarizes related work and Section 7 concludes the paper.

2. DESIGN CHALLENGES

SenseWeb aims to address the unique requirements and constraints posed by its constituent sensors, data, and target applications. This section summarizes the major challenges.

Heterogeneity: Unlike the sensors in monolithic systems, components of a shared sensing infrastructure may be highly heterogeneous along several dimensions:
Sensor Heterogeneity. The types of sensors may range from wireless motes, mobile phones, network cameras, pollution sensors, weather stations, to even RSS feeds. It is difficult to exhaustively list the types of sensors at design time.
Data Heterogeneity. Sensor heterogeneity leads to variations in the structure of data collected by sensors. It can be scalar, image, video, or more complex structures. For instance, a weather station sensor may generate both a binary image and an array of scalar measurements such as temperature, humidity, wind speed and direction.
Application Heterogeneity. Developers who wish to use a shared sensing substrate in their applications may have distinct domain specific needs. They are usually interested in different sensor types, geographical locations, and spatio-temporal resolutions. Moreover, data processing methods such as data cleaning, aggregation, and interpolation vary across different applications.

Scalability: A shared system becomes more and more useful as the number of participants grows, creating the community effect. This introduces significant challenges for scalability. As the number of sensors and applications grows, the demand for resources increases. A naïve centralized solution that collects and processes all the sensor data streams at a single server will not be efficient. The scalability challenge manifests itself in two key aspects.

First, the large data volume continuously generated by shared sensors imposes challenges in data collection and storage. Obviously, it is not efficient to transfer all the data sets to a central storage system, which may then become a bottleneck for communication and processing. Therefore, mechanisms that distribute the tasks of data collection, storage, and query processing are required.

Second, in typical applications, raw data streams traverse through multiple layers of processing (e.g., data cleaning, gap filling, aggregation and transformation). It is not scalable to exhaustively pre-compute the multiple layers due to diverse application and user requirements. For example, an environmental scientist may request per minute readings of temperature for every square kilometer, while other users

[1]Our focus is to to efficiently apply existing spatial interpolation methods in a distinct context rather than proposing new ones.

may only be interested in hourly average for a certain zip code. On the other hand, if data processing is always on demand, end users may experience intolerable response delays. Hence, a balance between pre-computation and on-demand computation needs to be found to make the system scalable but highly responsive.

In addition to heterogeneity and scalability, there are challenges in security, data verifiability, trust, data provenance, and sharing incentives [4, 6, 15] that have not been touched upon in this work.

3. GLOBAL SENSOR SHARING

This section focuses on the back-end infrastructure of SenseWeb that enables contributors to share their sensor deployments, and application developers to access the shared sensor data streams.

Dedicated sensor network deployments that are not shared are typically monolithic, operate on uniform hardware platforms, collect data via a common network stack, serve a fixed set of applications, and are maintained by a single team. After globally interconnecting and sharing such monolithic deployments, the resulting system becomes extremely heterogeneous in both hardware and software. Exposing such heterogeneity to application programmers complicates development, and is therefore undesirable. On the other hand, restricting the types of sensors and data to be shared is not an appropriate solution either since it constrains the system to a small subset of contributors. Therefore, the sharing system should be designed such that (i) it provides contributors the flexibility to share a wide spectrum of sensors, and (ii) it exposes simple abstractions for developers to isolate the complexity of heterogeneous sensors.

The SenseWeb system is designed to tackle the aforementioned challenges by providing an open and extensible architecture that exposes uniform interfaces to access sensors and their data streams. Conceptually, a sensor or sensor network is shared by adding its description to the SenseWeb index and applications can then discover such sensors based on location, type, or other characteristics. Sensor data may be accessed with specified time windows, into the past or future, at available sampling rates. This section describes the architectural design of the sharing system and discusses the details of sensor management.

3.1 Architectural Design

Figure 1 depicts the architecture of SenseWeb, showing the sensing substrate at the bottom, the sensing applications at the top along with the various SenseWeb components that interface these two entities. The various components expose their functionality using web services interfaces that allows applications developed on a variety of platforms to access SenseWeb.

Coordinator: Central to the architecture is the *Coordinator*, which serves as a common point of access for contributors to share sensor data streams and for applications to gain access to available data. It can be implemented in a hierarchical manner across multiple machines distributed across the Internet, although our current implementation has a centralized Coordinator. We divide the functionalities of the Coordinator into three components: *Application Manager*, *Sensor Manager* and *User Manager*.

The user manager implements user authentication mechanism. Based on their identity, users are granted different

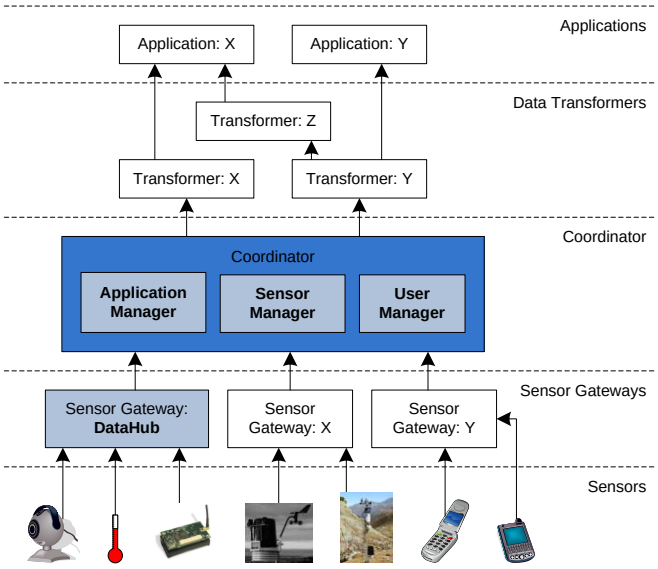


Figure 1: Architecture of the SenseWeb system

access privileges for different sensors. In-depth discussion on this subject is out of scope of the paper.

The sensor manager serves as an indexing engine of the shared sensors and their characteristics. It acts similar to a DNS server on the Internet, converting user friendly sensor descriptions such as location boundaries, logical names, or sensor types to physical sensor identifiers. To efficiently support spatial queries, the sensor manager indexes sensors by using a hierarchical triangular mesh (HTM) indexing scheme [18], which is particularly suitable for geographic queries. Besides providing indexing service to upper layers, it also provides APIs for sensor contributors to manipulate both sensors and their types. More concretely, it allows creation of sensor types, addition of sensors to the index, as well as modification of indexed sensors.

While the sensor manager takes care of sensor meta-data, the Application Manager is responsible for sensor data management. It incorporates a novel indexing technique called COLR-Tree [5] to optimize end-to-end latencies for real-time queries on sensor data streams by exploring spatio-temporal locality in query workloads.

Sensor Gateways: A sensor gateway interfaces sensors to SenseWeb. It manages a subset of sensors connected to SenseWeb. It accepts data collection tasks from the Coordinator and responds with relevant data. Based on the willingness of a contributor to share the physical deployment, sensor gateways can be categorized into two types: *physical* and *virtual*.

Physical gateways expose sensor control interfaces to the Coordinator. Sensors physically take samples upon the reception of data collection tasks. The Coordinator is able to track the quality, availability or capabilities of the corresponding sensors and inject tasks to a subset that best satisfies application requirements. Virtual gateways, on the other hand, add another layer of abstraction to isolate sensor-specific data sampling and collection from the shared system. They do not allow the Coordinator to control the sensors. Instead, they automatically collect data samples into

a local cache or database, and respond to data collection requests from stored data. The two types provides the SenseWeb system opportunities to optimize data collection while giving the contributors the freedom to keep their physical sensors private.

We have implemented a default virtual gateway named *DataHub* for contributors who do not host their own. For contributors who wish to store data locally, they are able to seamlessly integrate[2] their local gateways into SenseWeb by (i) implementing a set of predefined web service APIs, and (ii) providing URLs of the web service while registering sensors with SenseWeb. The Application Manager hides the complexity of distributed data sources from application developers by providing a unified query interface.

Data Transformers: The role of a transformer is to convert data semantics through processing. For example, a transformer may extract the people count from a video stream. Additional examples of data transformers are unit conversion, data fusion, and data visualization services. Domain experts can implement various transformers for different sensor data using suitable domain specific platforms.

The layered architecture of SenseWeb offers several benefits. First, it enables efficient sharing of heterogeneous sensors. SenseWeb abstracts away the heterogeneity of sensor hardware platforms and the complexity of physical data collection mechanisms. Applications can re-use existing sensor deployments, thereby amortizing deployment cost efficiently. Moreover, the spatio-temporal indexing and caching mechanism minimizes communication cost by exploring potential overlaps among applications.

Second, SenseWeb simplifies the process of application development while ensuring flexibility. Application developers can utilize data collection methods and relevant transformers already present in the system, so that they can focus their efforts on their specific applications. They can extend the system as per their needs and also share results from their development efforts as additional transformers.

Finally, by using distributed sensor gateways to serve sensor data, the architecture is able to accommodate potentially large number of sensors that continuously generate data.

3.2 Sensor Management

On stand-alone sensor systems, structures of sensor data streams are usually known and rarely change. However, as a platform for data sharing across sensor systems, SenseWeb confronts challenges in sensor management to support an unknown variety of sensors. An intuitive solution would be to grant contributors the full freedom of defining any sensor type. This results in enormous burdens on application developers who have to later interpret the dynamic types. Alternatively, a static type system which hard-codes a list of supported sensor types is apparently too rigid for a shared system. Therefore, we design a tree-based type system that allows new types to be defined hierarchically based on a list of known types.

Figure 2 illustrates the tree-based sensor type system. SenseWeb maintains a predefined set of data types such as scalar and binary. The system was created with only a set of *primary sensor types* that were thought to be commonly used, including thermometer, video camera, humidity sensor, etc. Each primary sensor is associated with a specific data type, that enables application developers to correctly interpret the corresponding data streams. As the sharing system grows, new contributors may wish to add new types of sensors, for example, rain meters. SenseWeb allows such contributors to define new primary types and their association with data types. A sensor does not necessarily generate only one data sample at a time. More complex sensors such as a weather station measure temperature, humidity, and rain amount simultaneously. To support such sensors, the type system supports *compositional sensor types* that are hierarchically defined. A compositional sensor type is defined as an array of existing sensor types, primary or compositional.

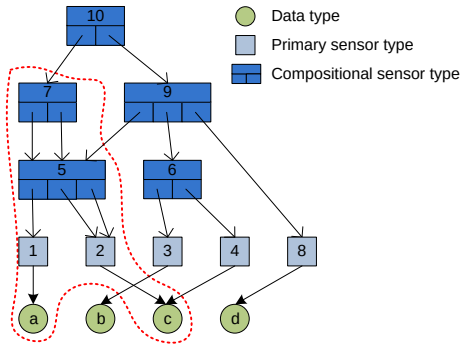


Figure 2: Tree-based sensor type system

The tree-based type system lets the contributors enjoy the freedom to construct any complex types. Meanwhile, it retains programmability in type interpretation. Following the type tree, any given type can be decomposed into an ordered array of primary types by recursively unfolding its child types. The primary types can be further mapped into a list of known data types, which reveals the structure of the corresponding sensor streams. A concrete example of interpreting type 7 in Figure 2 is given below:

$$[7] \mapsto [5,5] \mapsto [[1,2,2],[1,2,2]] \mapsto [[a,c,c],[a,c,c]]$$

The deployed version of SenseWeb has been extended by users to add new types such as weather stations and other environmental sensors using the initially created types (details in Section 5.1).

4. SPATIO-TEMPORAL DATA EXPLORATION

The SenseWeb infrastructure enables sharing of heterogeneous sensor streams. Users can issue queries to access both real-time and historic data from geographic regions of interest. To further simplify the data exploration process, we provide an interactive geocentric interface letting users graphically issue spatio-temporal queries for sensor data, and view results (raw or sampled sensor data) directly over a map that can be viewed, panned, and zoomed in a browser.

A user can specify the area of interest by drawing a polygon directly on the browsable map, or by typing a descriptive term (such as an address or city name) in the SensorMap web interface. SensorMap automatically aggregates the results at an appropriate granularity based on the zoom level

[2]Tutorials and sample code are available online: `http://research.microsoft.com/nec/senseweb/`

of the map and generates appropriate visualizations for display.

A user can explore sensor data streams along both temporal and spatial dimensions. Via the SensorMap interface, a user can select a list of sensors of interest to visualize their temporal distribution in a single comparison chart or in multiple side-by-side time series charts. SensorMap also generates map-overlaid contours of real-time or archive data of selected sensors in view.

Below, we present the design of the SensorMap application, focusing on scalable spatial data presentation techniques that enable rapid changes to map-overlaid contours.

4.1 SensorMap Design

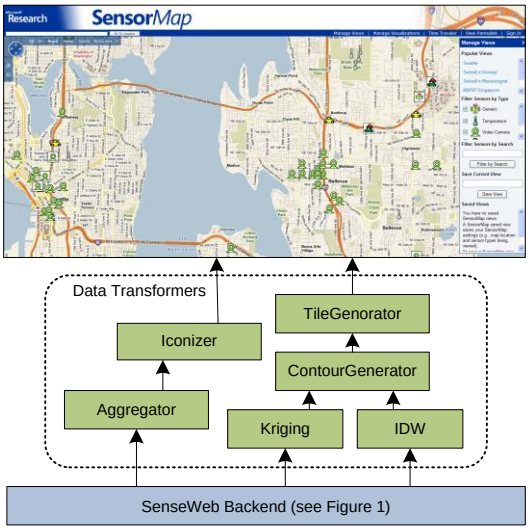


Figure 3: Architecture of the SensorMap application

Figure 3 shows the architecture of the SensorMap application. Bottom of the architecture is the SenseWeb back-end system. Above the back-end system there are multiple reusable data transformers.

Given a desired spatial granularity, the *Aggregator* appropriately clusters and aggregates readings of a set of selected sensors based on their distances along the Earth surface. The *Iconizer* further converts such aggregate readings to image icons, which are then displayed on top of a map at corresponding locations by the SensorMap application. By utilizing the two transformers, SensorMap can present snapshot readings of the sensors at an appropriate granularity based on the current zoom level of the map.

To present the spatial distribution of a measured metric such as humidity, another set of transformers are provided for generation and overlay of contour maps. User contributed sensors usually lead to partial and irregular coverage. To generate contour maps, it is necessary to interpolate the values of a specific metric (e.g., the temperature) at unobserved locations based on known sensor measurements at nearby locations. We implement two alternative spatial interpolation methods: *Kriging* [1] and *Inverse Distance Weight* (*IDW*) [17]. This is to give users the flexibility to trade off between accuracy and latency (described in detail in Section 4.2). Given a list of sensor measurements at several random locations within a geographical region, these interpolation transformers generate matrices of interpolated values that cover the whole region. The matrices are then processed by the *ContourGenerator* to create contour images. Finally, the *TileGenerator* converts the contour images into localized tiles that can be overlaid by SensorMap on top of a browsable map.

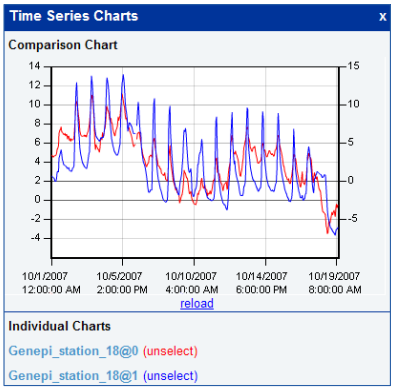


Figure 4: Visualization of temporal distributions and correlations

SensorMap visualizes temporal distributions of data in time series charts based on AJAX and Flash techniques, as shown in Figure 4. Both the time duration and resolution are controllable. This greatly simplifies exploration and correlation of temporal data. For instance, in the figure, users can clearly see the interactions between ambient (red curve) and surface (blue curve) temperatures.

The SenseWeb back-end allows users to query sensors and their data streams. The additional transformers supplied by SensorMap further enables users to query over aggregate data, icons, and contour maps. Preliminary experiments reveal that latency of icon queries ranges from sub-second to a few seconds depending on the number of sensor data streams in view. However, generation of contour visualizations is much more expensive, often taking more than 10 seconds for large datasets. Next, we focus on the techniques enabling rapid generation and update of contour maps.

4.2 Scalable Spatial Data Presentation

To visualize spatial data using contour maps, SensorMap provides two interpolation methods: Kriging and IDW, allowing users to switch between the two methods to opt for accuracy or latency. As early study shows [11], Kriging is relatively more accurate and less affected by the coefficient of variation. However, it is fairly computation-intensive since it includes an expensive $K \times K$ matrix inversion, especially for large K (K is the number of sensor measurements within the geographical region of interest). Consequently, large processing latency is expected. Comparatively, IDW is faster (though less accurate) since its computational cost is lower.

In the literature, several variations of Kriging [14, 13] have been proposed to accelerate the interpolation process in the context of scientific high-performance computing. However, this work considers a distinct application context, map-based spatial data browsing. As a user zooms or pans the map to navigate across spatially correlated regions, contour maps of the regions in view are generated on the fly to enable real-time exploration of spatial distributions. Unlike

domain specific scientific analysis where accuracy is critical while latency is usually a secondary concern, SensorMap based data exploration is meant for interactively discovering interesting data and phenomena. Hence, we focus on the less computation-intensive algorithm, IDW, and propose various techniques to further reduce its computational cost, thereby reducing response time.

Basic IDW: To generate a contour map for a specific region, the first step is to assign values to unknown locations based on scattered set of known sensor measurements. Prior work [17] suggests calculating the value of an unknown point as a weighted average of the known values, using a function of the distances between the unknown point and the known ones as the weights. The proposed interpolating function to estimate value Z at point x_0 is as follows:

$$\hat{Z}(x_0) = \begin{cases} Z(x_i), & \text{if } \exists i \in \{1, 2, \cdots, K\},\ d(x_i, x_0) = 0 \\ \dfrac{\sum_{i=1}^{K} (d(x_i, x_0))^{-\lambda} Z(x_i)}{\sum_{i=1}^{K} (d(x_i, x_0))^{-\lambda}}, & \text{otherwise.} \end{cases}$$

where K is the number of known points, x_i is a known point, $d(x_i, x_0)$ is the distance between x_i and x_0, and λ is a positive number to control the impact of closer points. Larger λ leads to higher influence from the closer points, therefore resulting in steep gradients between data points. The complexity of IDW when interpolating a $M \times N$ region is $O(M \times N \times K)$.

Localized Interpolation: It is clear that the basic IDW is not scalable with K, the number of known points or sensor measurements. However, when users of SensorMap zoom out to a large region, it is likely that hundreds or thousands of sensors come into view, leading to long delays in IDW-based interpolation. In IDW, one important observation is that the points further away from the interpolated point contribute less to the interpolated value. Hence, considering only sensors that are close to the interpolated point should not cause a large degrading in accuracy, which motivates a localized variation of IDW.

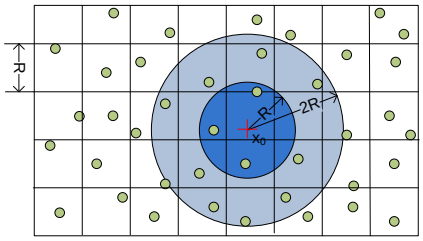


Figure 5: Localized interpolation of x_0 based on sensors within R, $2R$, $\cdots$

Figure 5 depicts the localized interpolation of an unknown point x_0 based on nearby sensors. Only the set of sensors whose distances to x_0 is within R (called *effective range*) are considered when calculating $\hat{Z}(x_0)$ while others are ignored. However, it is common that user contributed sensors are not uniformly distributed. To account for sparse regions with few sensors or no sensors, a threshold K_{min} is defined such that for each unknown point, the algorithm keeps expanding its effective range from R to $2R$, $3R$, $\cdots$ until at least K_{min} sensors are found.

Griding and Sampling: The localized version of IDW still requires traversing through all the K sensors to find closer ones to an interpolated point. We further optimize the algorithm by preprocessing the set of sensors and clustering them into a grid of $\lceil M/R \rceil \times \lceil N/R \rceil$ squares with width R as shown in Figure 5. Each square maps to a list of sensors within the square. The preprocessing occurs only once during the interpolation process. To find sensors within distance R from a certain point x_0, we only need to traverse sensors within the same square as x_0 and its neighboring squares. The overall complexity thereby becomes[3]

$$O(K + M \times N \times (9R^2D + \pi R^2 D)) = O(K + MNR^2D)$$

where D is the density of sensors. This algorithm is significantly more scalable with K compared to the original IDW.

To further reduce response time, we provides the option to compute approximate contours using only a subset of all available sensors within the query region. Users can define a sample cap K_{max}. When the total number of sensors in view exceeds K_{max}, a sampling algorithm randomly selects K_{max} sensors as inputs to the interpolation algorithm. Sampling essentially sets an upper-bound for K and D, therefore further enhancing scalability.

4.3 Multi-Resolution Caching

Due to the presence of spatio-temporal locality in query workloads, caching allows SensorMap to re-use already computed contour maps, avoiding redundant contour computation and data collection, and to deal with temporarily disconnected sensors. The basic idea of caching is simple: (i) store into local cache recently computed contour matrices containing sensor data and interpolated values, and (ii) for the contour matrix of a new query, use values from cached matrices as much as possible. The localized interpolation of our IDW algorithm enables us to easily reuse parts of disjoint cached matrices in a new matrix, without further processing them.

The following two factors make caching and reusing contour matrices in SensorMap challenging.

1. Most queries only partially overlap in space with each other, requiring parts of different overlapped matrices from the cache to be cropped and combined to answer new queries. With a large number of matrices in the cache, detecting overlapping matrices and combining them efficiently is challenging.

2. Different queries are made with different zoom levels and over regions of different sizes—at a higher zoom level, the same viewport includes a smaller physical region than that at a lower zoom level. Therefore, even though different cached matrices represent viewports of the same size, they in fact represent physical regions of different sizes. Such multi-resolution matrices further complicate the process of identifying, cropping, and combining overlapped matrices.

In summary, coordinates of physical regions represented by different cached matrices must be translated and scaled

[3]Special processing for sparse regions is ignored for simplicity.

with respect to each other and this must be taken into account during cache lookup.

Scaling and Translation: To support caching and efficient reuse of matrices, which might be arbitrarily translated and scaled with respect to one another, we use an imaginary reference matrix $\mathcal{M}$. The dimension of $\mathcal{M}$ is the same as that of the entire physical region (i.e., the entire surface of the Earth) at the maximum zoom level. Before comparing matrices, they are normalized by projecting to $\mathcal{M}$, as Figure 6 depicts.

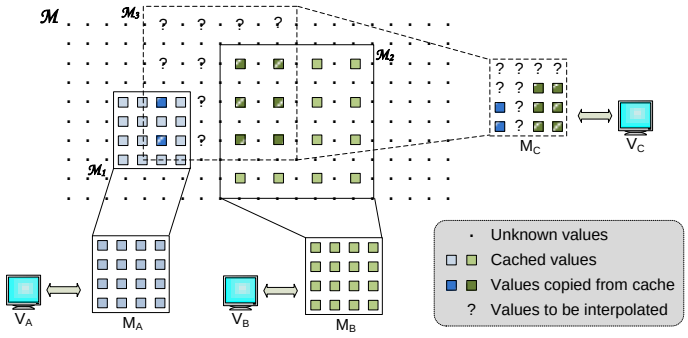


Figure 6: Interpolation of region V_C based on cached regions V_A and V_B

Suppose the current viewport V, and hence the matrix M representing contours in V, is of dimension $m \times n$. If the current viewport is at the maximum zoom level such as V_A, it covers an $m \times n$ region of $\mathcal{M}$, and hence there exists a one-to-one mapping between M_A to a submatrix $\mathcal{M}_1$ of size $m \times n$ in $\mathcal{M}$. If the current viewport is at a lower zoom level such as V_B, it shows a $k_1 m \times k_2 n$ region of the entire physical region, where k_1 and k_2 are scaling factors of the zoom level. Hence, V_B and M_B project to a submatrix $\mathcal{M}_2$ of size $k_1 m \times k_2 n$ in $\mathcal{M}$. Since $\mathcal{M}_2$ is larger than M_B, elements of M_B map uniformly sparsely in $\mathcal{M}_2$; more precisely, one of every $k_1 \times k_2$ elements of $\mathcal{M}_2$ is mapped by one element of M_B.

Conceptually, one can model the entire cache with the $\mathcal{M}$. Before caching a contour matrix M computed for the current viewport V, V is first projected to $\mathcal{M}$, and the projected elements of $\mathcal{M}$ are then populated by the elements of M. To lookup cache for a viewport, say V_C, it is first projected to a submatrix $\mathcal{M}_3$ of $\mathcal{M}$. After the elements of $\mathcal{M}_3$ are retrieved from the cache, $\mathcal{M}_3$ is then projected back to M_C corresponding to V_C. Some elements for M_C may not be available in cache, and they need to be computed to complete M. The computed elements can again be projected back and cached in $\mathcal{M}$.

The matrix $\mathcal{M}$ is very sparse and we do not need to maintain the entire matrix in cache. Rather, we cache only the contour matrices, which in effect contain the nonempty elements of $\mathcal{M}$. During cache lookup, each matrix is dynamically projected to $\mathcal{M}$ to determine if it overlaps with the query region, which has also been projected to $\mathcal{M}$. The entire process is shown in Figure 6, where parts of cached matrices M_A and M_B are used in the matrix for M_C. First, the three regions V_A, V_B, and V_C are converted to use the maximum zoom level to find out their overlaps in $\mathcal{M}$. Next, the cached values within overlapped regions are copied from M_A and M_B to M_C. Note that a 2×3 submatrix of M_A is copied to a 1×2 submatrix of M_C, since V_C covers a 2×2 bigger region than V_A. Similarly, a 2×3 submatrix of M_B is directly copied to a 2×3 submatrix of M_C, as both V_B and V_C use the same zoom level and hence represent physical regions of the same size. Finally, localized IDW is applied to calculate the remaining unknown values of region V_C.

Cache Lookup: During cache lookup, we need to find all cached matrices that overlap with the query region. One way of doing that is to maintain all cached matrices in an in-memory list, and to scan them sequentially while comparing with the query region. Since the list is maintained in memory, the performance of such scan might be acceptable to many applications. If the number of matrices is too big to cache in memory, only metadata (translation and scale with respect to $\mathcal{M}$) of each matrix is maintained in memory and actual matrices can be stored as separate files in disk. The metadata lookup can be made faster by organizing them as a tree structure; for example, the bounding boxes of cached matrices can be organized as an R-Tree, which can identify matrices overlapping with a query region in logarithmic time.

Expiring Cache: SensorMap removes cached matrices after a configurable expiry time. Each contour matrix is timestamped and put in one end of a circular buffer; while expired matrices are removed from the other end of the circular buffer. If the matrices are stored as files on disk, files corresponding to expired matrices are deleted too.

5. EVALUATION

In this section, we present empirical results collected based on a prototype system. The techniques discussed in the paper were deployed in March 2008.

Both the default sensor gateway (DataHub) and the Coordinator were implemented using C# with Microsoft SQL Server 2005 used as the back-end for sensor indexing and data storage. Applications (including SensorMap) access the Coordinator using web service interfaces. We prototyped most of the transformers in C#, except for the TileGenerator, which uses a binary tool MapCruncher [2] for localizing the contour images with respect to the map. The SensorMap front-end is written in ASP.NET and JavaScript. It accesses transformers and back-end services via their web service interfaces. Interpolation algorithms implemented in the transformers are Ordinary Kriging and the localized IDW described in Section 4.2. In the rest of the section, we use Kriging to refer to Ordinary Kriging and IDW to refer to the localized IDW unless otherwise stated.

Below, we first analyze the end-to-end performance of SenseWeb based on the real sensors shared on the deployed prototype. Next, we investigate the performance of different contouring techniques using two large datasets.

5.1 System Performance

The deployed prototype system was initialized with several predefined sensor types, including 9 primary types and one compositional type. Currently, the system contains l52 primary types, 9 compositional types, and over 3,000 sensors, which are created by contributors as needed. It is evident that users of SenseWeb have enjoyed the flexibility of defining and registering heterogeneous sensors.

We first analyze the performance of sensor and data queries using a set of popular sites, including temperature, traffic,

	#types	#primary sensors	#compositional sensors	Sensor gateway
Singapore	1	18	0	DataHub
Wannengrat, Switzerland	27	81	7	GSN [16]
Seattle, USA	5	102	0	DataHub
Taiwan	9	91	26	DataHub
Le Génépi, Switzerland	10	144	16	DataHub

Table 1: Characteristics of different sites

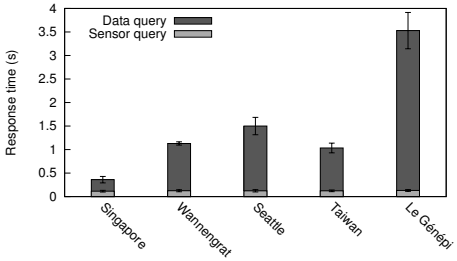


Figure 7: Response time of sensor and data queries

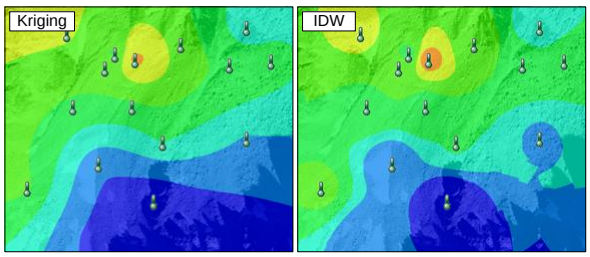


Figure 8: Comparison of contour maps generated by Kriging and IDW

and camera sensors in Seattle, weather stations in Le Génépi (Switzerland), weather towers in Wannengrat (Switzerland), camera and rain sensors in Taiwan, and weather stations in Singapore. The sensor type categorizations and gateways of the sites are listed in Table 1. Note that the weather towers in Wannengrat are hosted by a user supplied gateway, called GSN [16], instead of DataHub.

For each of the sites, we issued multiple queries to request the list of sensors as well as their most recent readings. Figure 7 depicts the response time of sensor meta-data and data queries for the five sites. As is seen, the time it takes to get the list of sensors within a site is almost constant. Comparatively, data query latency is proportional to the number of primary sensors. The underlying reason is that sensor readings are indexed by sensors, which determines querying the most recent reading always incurs certain amount of per-sensor cost (depending on the number of readings in record for a specific sensor). We did not observe longer delays for the Wannengrat site hosted by a remote gateway, since the amount of data transferred over the Internet is relatively small and the bandwidth between our server and the remote site is sufficient.

Next, we study the end-to-end performance of the system including sensor query and data fetching as well as contour generation. The experiments are based on the Le Génépi site which consists of 16 weather stations deployed on top of a rocky glacier in the SwissEx project [3]. Each sensor sampled once every 2 minutes from August 2007 to October 2007, resulting in approximately 40,000 samples per sensor stored in DataHub.

In the experiment, we repeatedly submitted 100 queries to get contour maps of the average surface temperature readings during the hour starting from 6PM of October 23, 2007. Based on the experiment, the end-to-end response time (including data query delay and IDW-based contouring delay) is 2.1s on average with little variation, out of which data query takes up to 0.8s. Note that during this experiment we disabled caching in all the transformers and the Coordinator. We observed similar distribution and end-to-end latency for Kriging-based contouring. Such observation is consistent with microbenchmark results in the next section, which shows that IDW and Kriging incurs similar latency for small number of sensors, say 16.

The resulting contour maps (overlaid on terrain maps) of Kriging and IDW are shown in Figure 8. They are visually similar, which indicates that the outcome of IDW is still acceptable though less smooth compared with Kriging.

5.2 Contouring Performance

This section presents evaluation results of the different contouring techniques presented in Section 4. We used response time as a major metric, which includes both the time to interpolate unknown values as well as the time to generate contour maps as bitmap files. The dimensions of the interpolated value matrices are set to be 256×256. We used the following two datasets:

Synthetic Data: To isolate the impact of non-uniform distribution of sensors, we use synthesized sensor data as inputs to the interpolation algorithms. Sensor density is the major factor that affects the performance of the adapted IDW. Therefore, in this dataset we synthetically created 65,535 sensors that are evenly distributed in the space to ensure a fair comparison. The virtual map containing the synthetic senors has 7 zoom levels.

YellowPage Queries: This dataset is a real workload from Live Search Maps, consisting of 10,493 queries, each characterized by a geographical region over which the query was posed, searching over approximately 370,000 restaurants in the US. The data serves as a proxy for the geographical regions that may be queried by users of SensorMap. Consider for instance the searches over restaurants as representing a workload for a hypothetical restaurant wait time finding service where each sensor publishes real-time wait time from the restaurant.

5.2.1 Synthetic Data

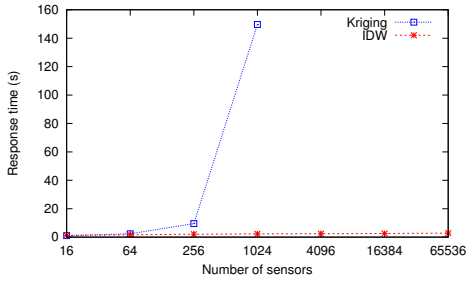


Figure 9: Comparison of Kriging and IDW with increasing number of sensors in view

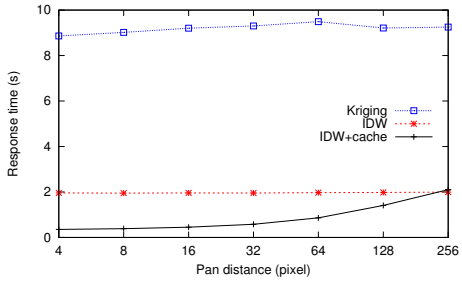


Figure 10: Comparison of Kriging, IDW and IDW with cache during panning

In this set of microbenchmarks, we compared the response time of Kriging and various adaptations of IDW.

Figure 9 shows the response time of Kriging and IDW as the view size is increased and more and more sensors come into view. As is seen, IDW is more scalable with the number of sensors. The latency of Kriging jumps to 150s for 1024 sensors, which is obviously intolerable in an interactive web-browsing application.

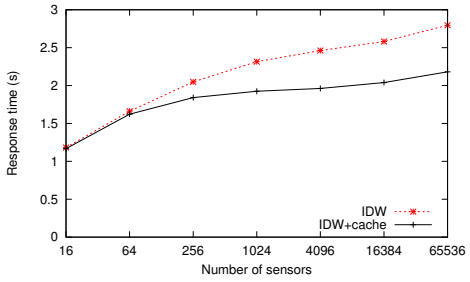
Figure 10 depicts the effect of caching as the user pans the map to view a slightly different region. The X axis shows the distance in terms of pixels the user panned in one step while the Y axis shows the response time. IDW with cache has significantly lower latency when the overlap fraction is high (small pan distances).

Figure 11(a) illustrates the impact of caching for IDW during zooming out. Caching reduces response time as when a user zooms out to a relatively coarse resolution, the previous region in view is a subset of the current one and the cached values are copied into the new value matrix, leading to saving in computation. Similarly, when the user zooms in to a finer resolution (Figure 11(b)), part of the cached values can again be reused to reduce latency.

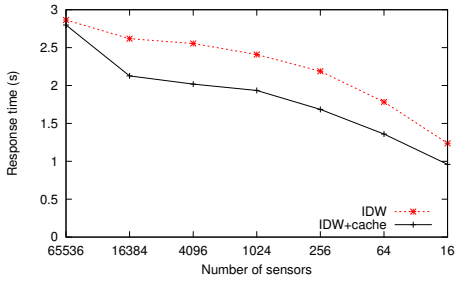
5.2.2 *YellowPage Queries*

Finally, we used the real workload from Live Search Maps since it provides a realistic distribution of geographical regions in user queries and their overlap fractions.

We first investigates the impact of caching durations (i.e., the duration after which a cached matrix is expired). Figure 12 depicts the histogram of response time for the scenarios without cache, with 1 minute cache, and with 5 minute



(a) Zoom out



(b) Zoom in (the number of sensors in view decreases when zooming in)

Figure 11: Comparison of IDW and IDW with cache during zooming in and zooming out

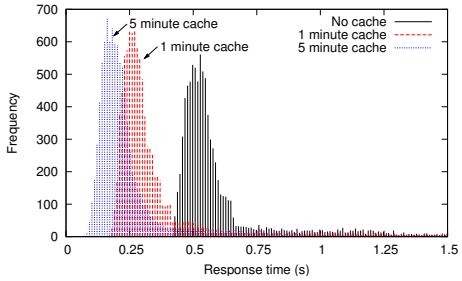


Figure 12: Effects of varying caching durations

cache. It shows on Y axis the number of queries falling into each time interval of length 0.01s. With a longer cache duration, we observe smaller response time as expected. This performance gain is achieved by sacrificing data freshness.

Next, we study the impact of sampling: randomly dropping available sensors beyond a sample cap size. Figure 13 depicts the average response time of the whole query set with varying cap size and caching durations: the response time is shorter for smaller sample caps and longer cache durations. Even a relatively large cap size of 1000 sensors leads to 50% performance gain in response time. This is due to the long-tail distribution of number of sensors per query: users occasionally zoom out to the state level and even the nation level asking for all the restaurants in view.

6. RELATED WORK

Several systems have been proposed for wide area sensing. IrisNet [9] provided a database abstraction to query sensor

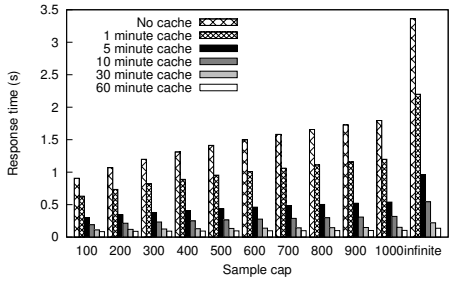


Figure 13: Effects of varying sample caps and caching durations

data distributed across multiple Internet-connected nodes. SenseWeb introduces several new capabilities, including the ability to host data externally rather than within the infrastructure, and support for heterogeneous devices. Other systems [10, 12, 15, 8] have also proposed the use of sensors contributed by multiple entities and many of their contributions can help improve SenseWeb design. We address several issues not previously considered, including specific support for sensor heterogeneity, and spatio-temporal data exploration; these contributions are beneficial for all the above projects.

While an open system for peer production is not currently available for sensor networks, several such systems have been successfully realized for other content including blogs, image or video (`Flickr.com`, `Myspace.com`). Sharing specialized sensor data is already supported in systems such as `WeatherUnderground.com` for weather data, `EarthCam.com` for webcams, and `SensorBase.org` for generic sensor data. SenseWeb extends such initial attempts to realize a system that allows developing sensing applications over shared resources, supporting multiple sensor types, and introducing design improvements for efficient and scalable exploration of the shared data.

The introduction of "Web 2.0" has enabled many interesting *mashup* applications, especially on maps, listed at `programmableweb.com`. While such mashups are not designed to achieve our vision of sharing sensing resources across multiple applications, their popularity motivates our data exploration front-end, SensorMap. The availability of map mashup tools like MapCruncher [2] further enables SensorMap to dynamically overlay spatial visualizations over terrain maps.

7. CONCLUSIONS

We presented a shared sensing system, SenseWeb, that enables sharing of sensor deployments at a global scale. We described the system architecture that addresses the challenges of heterogeneity and scalability in such a large scale system. A first application of SenseWeb is an interactive spatio-temporal sensor data exploration front-end, named SensorMap. We discussed the challenges in designing resource efficient and highly responsive geocentric data visualization in SensorMap and described our approaches. Through effective choices in computational methods, efficient re-use of visualizations across multiple user queries, and selective data caching, we showed that response times suitable for interactive human user data exploration could be achieved. The trade-offs relating various design choices were evaluated on real world data sets. A deployed version of our system that is already in use by several scientific and academic sensor deployments in different continents was also illustrated. The techniques proposed in this paper are likely to be beneficial for geocentric data exploration where a large number of data resources are accessed and spatio-temporal visualizations are required in near real time for interactive use.

Next we plan to further extend the work in several directions, including: (i) efficient querying and presentation of non-numerical data types such as images or videos, and (ii) indexing and visualization techniques for mobile sensors with dynamically changing geographical locations.

8. REFERENCES

[1] Kriging. http://en.wikipedia.org/wiki/Kriging.

[2] Msr mapcruncher. http://research.microsoft.com/mapcruncher.

[3] Swiss experiment: Interdisciplinary environmental research. http://www.swiss-experiment.ch.

[4] T. Abdelzaher, Y. Anokwa, P. Boda, J. Burke, D. Estrin, L. Guibas, A. Kansal, S. Madden, and J. Reich. Mobiscopes for human spaces. *IEEE Pervasive Computing*, 6(2):20–29, 2007.

[5] Y. Ahmad and S. Nath. Colr-tree: Communication-efficient spatio-temporal indexing for a sensor data web portal. In *ICDE*, April 2008.

[6] M. Balazinska, A. Deshpande, M. J. Franklin, P. B. Gibbons, J. Gray, M. Hansen, M. Liebhold, S. Nath, A. Szalay, and V. Tao. Data management in the worldwide sensor web. *IEEE Pervasive Computing*, 6(2):30–40, 2007.

[7] Y. Benkler. Coase's penguin, or, linux and the nature of the firm. *Yale Law Journal*, 112, 2002.

[8] J. Burke, D. Estrin, M. Hansen, A. Parker, N. Ramanathan, S. Reddy, and M. B. Srivastava. Participatory sensing. In *WSW*, October 2006.

[9] A. Deshpande, S. Nath, P. B. Gibbons, and S. Seshan. Cache-and-query for wide area sensor databases. In *ACM SIGMOD*, 2003.

[10] S. B. Eisenman, N. D. Lane, E. Miluzzo, R. A. Peterson, G.-S. Ahn, and A. T. Campbell. Metrosense project: People-centric sensing at scale. In *WSW*, October 2006.

[11] C. A. Gotway, R. B. Ferguson, G. W. Hergert, and T. A. Peterson. Comparison of kriging and inverse-distance methods for mapping soil parameters. In *Soil Sci Soc Am J 60:1237-1247 (1996).*

[12] B. Hull, V. Bychkovsky, K. Chen, M. Goraczko, A. Miu, E. Shih, Y. Zhang, H. Balakrishnan, and S. Madden. CarTel: A distributed mobile sensor computing system. In *ACM SenSys*, 2006.

[13] K. E. Kerry and K. A. Hawick. Kriging interpolation on high-performance computers. In *HPCN Europe 1998*, pages 429–438, London, UK, 1998. Springer-Verlag.

[14] D. C. Mason, M. O'Conaill, and I. McKendrick. Variable resolution block kriging using a hierarchical spatial data structure. In *International Journal of Geographical Information Science, Volume 8, Issue 5*, pages 429–449, 1994.

[15] O. Riva and C. Borcea. The urbanet revolution: Sensor power to the people! *IEEE Pervasive Computing*, 6(2):41–49, Apr-Jun 2007.

[16] A. Salehi and K. Aberer. Gsn, quick and simple sensor network deployment. In *EWSN*, 2007.

[17] D. Shepard. A two-dimensional interpolation function for irregularly-spaced data. In *Proceedings of the 1968 23rd ACM national conference*, pages 517–524, New York, NY, USA, 1968. ACM.

[18] A. Szalay, J. Gray, G. Fekete, P. Kunszt, P. Kukol, and A. Thakar. Indexing the sphere with the hierarchical triangular mesh. In *MSR-TR-2005-123*, September 2005.

[19] G. Tolle, J. Polastre, R. Szewczyk, N. Turner, K. Tu, P. Buonadonna, S. Burgess, D. Gay, W. Hong, T. Dawson, and D. Culler. A macroscope in the redwoods. In *ACM SenSys*, 2005.

[20] G. Werner-Allen, K. Lorincz, J. Johnson, J. Lees, and M. Welsh. Fidelity and yield in a volcano monitoring sensor network. In *OSDI*, 2006.