Title
SensorBase.org - A Centralized Repository to Slog Sensor Network Data

Authors
Chang, Kevin
Yau, Nathan
Hansen, Mark
et al.

Publication Date
2006-05-05

Peer reviewed

SensorBase.org– A CENTRALIZED REPOSITORY TO SLOG SENSOR NETWORK DATA *

Kevin Chang, Nathan Yau, Mark Hansen, Deborah Estrin
University of California, Los Angeles
Center for Embedded Network Sensing
Computer Science Department
Statistics Department
Los Angeles, CA 90095
{kchang,destrin}@cs.ucla.edu, {nyau,cocteau}@stat.ucla.edu

Abstract Various sensor networks use different data storage and management mechanisms. In particular, UCLA's ESS2 mechanism forwards information from low powered 8-bit Mica2 motes to one or more low powered sinks that then push log files to a secure and centralized repository. While this data storage and management mechanism is straightforward to implement, publishing and sharing various data to various users has been a challenge. Users that need to parse the data often find it time consuming and error prone to have to log-on to different systems, calibrate different units, and to understand the different semantics of various log files.

Similar to blog sites, SensorBase.org is our solution for a certain domain of sensor networks that allows users to "slog" sensor network data and other relevant information. It is a centralized common data storage and management system that provides a uniform and consistent method for publishing and sharing data. It allows users to define a subset of EML data types, project groups, and permission levels. It also serves as a search engine that provides APIs a way to easily query for specific data sets based on geographic location, sensor type, date/time range, and other relevant fields.

Keywords: C.2.7.c Sensor networks, H.3.5.b Data sharing, D.2.1.d Management

*(slog is a portmanteau of "sensor" and "log" and was coined to reflect the spirit of sharing information represented by blogs)

*This material is based upon work supported by the National Science Foundation under Grant No. CCF-0120778.

1. Introduction

In recent years, blogs have become increasingly popular on the World Wide Web. Writers share their daily activities, opinions, thoughts, rants, or raves with the rest of the world. Blog user interfaces, for the most part, are easy to use with a few required text fields and a button to publish writings. Entries can easily be categorized, edited, deleted, made public or private, and just as easily read, receive feedback, searched, and received via various forms of XML. The advantages of sharing well defined XML are numerous. Applications using XML can interoperate and interface with each other easily.

While the general text sharing mechanism is straightforward in blog sites, sharing domain specific data provides new challenges. For example, the data structure to describe "Housing" is very different than the data structure to describe "Vehicles". Google Base is an example of an attempt at gathering different data structures and presenting them on a uniform and easily accessible database via intuitive interfaces [1].

Like Google Base, SensorBase.org is created to meet our needs for sharing and managing a specific domain of sensor network data. It is our solution to data storage and management that provides a uniform and consistent method for publishing and sharing our sensor network data. SensorBase.org allows users to "slog" data. Figure 1 shows the overview of how sensor network data is slogged.

Similar to blogging, SensorBase.org provides a user-friendly environment to publish data with fields for a subset of EML data types, project groups, permission levels, and more. As with blogging, users slogging on SensorBase.org can send and retrieve data via standardized formats, and provides API's a way to effectively utilize shared data. Additionally, with millions of stored data sets, SensorBase.org also serves as a search engine that provides users the ability to query for specific data sets based on geographic location, sensor type, range of time, and eventually patterns in the sensor signals themselves.

SensorBase.org aims to solve some of the storage and sharing problems we have encountered in the past several years of deploying sensor networks. For example, various sensor networks use different data storage and management mechanisms, which presents difficulties for data consumers who have to first understand a data format and then parse the data. This can be time-consuming and error-prone. This problem is alleviated by SensorBase.org's standardized data format. The standardization coupled with fine-grained queries make it easy for applications to interoperate and interface with SensorBase.org. Just as importantly, SensorBase.org's search capabilities make it easy for researchers to find

data specific to geographic areas and SensorBase.org's group feature allows for easy collaboration.

2. Related Work

As the amount of sensor network data grows, there is a greater need for heterogeneous data storage and retrieval systems. Different approaches to data storage and management evolve out of different needs. Some past works create "in-network" database-like systems while others take a data warehouse approach. There are also numerous related projects that attempt to represent and unify data in an intuitive and meaningful way.

The Sensor Network as a Database describes the importance of a query-based database system that is essential to maintaining data [2]. Work such as TinyDB and Cougar treat sensor network as the database and focus on the distributed query mechanisms as the way of extracting data from sensor network nodes [3] [4]. Since it is difficult to store large amounts of data "in-network", this approach is complementary to data management systems such as SensorBase.org that store gigabytes of data to be shared with the public.

IrisNET is a platform that specifically addresses high-bit-rate, off-the-shelf sensors, such as webcams, to drive data hungry applications [5]. It

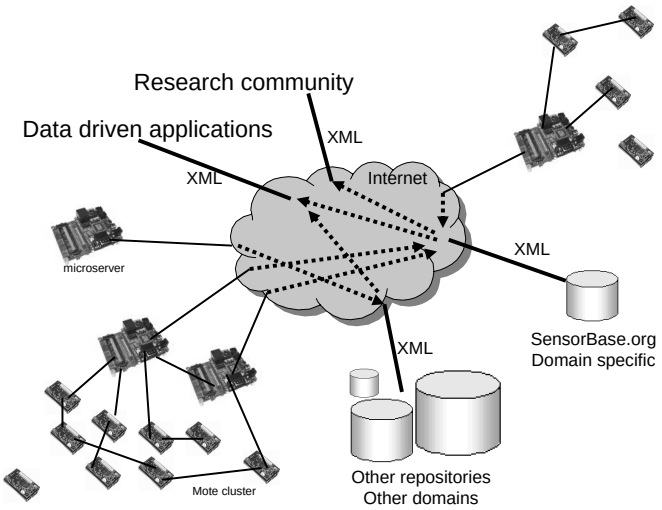

Figure 1. Sensor Network Data Sharing Overview

discusses all of the levels of data push and sharing mechanisms, and tries to address issues such as authoring, data representations using its own geo-aware XML data formats, caching, data consistency, and others. Like SensorBase.org, the IrisNET architecture is ever evolving to meet the needs and demands of users who need to publish and share sensor network data.

In 2000, the German Federal Environmental Agency in Berlin proposed a standardized method of representing environmental data, an XML-based Environmental Markup Language (EML) [6]. In this format, highly structured environmental sensor data can be serialized using typical XML structure. Additionally, semantic information can be embedded in the names of entities and attributes. This in turn solves the problem of out-dated or missing documentation meant to explain ambiguous data sets.

Although EML provides a solution to the data storage problem, there is still the task of data recall. Researchers need to find the data that they want quickly and easily. Several groups use EML, which can be seen online at Sevilleta Long Term Ecological Research; however, data sets are available only in the form of flat EML files [7]. Searches are based on only four fields–project, research theme, principal investigator, and description. To find a data set users need good prior knowledge on what they are looking for. Once users find a data set, they need to download an entire log file. With SensorBase.org, users can make more fine grained queries as well as focus their attention on subsets of data results.

CRAWDAD, a Community Resource for Archiving Wireless Data at Dartmouth, is similar to SensorBase.org in that it serves as a database for data sets [8]. Like SensorBase.org, CRAWDAD also has searchable data sets and outputs are in a metadata format, which makes data easy for analysis after retrieval. Although similar in structure, CRAWDAD is a database mostly for packet and network data while SensorBase.org is built and tuned specifically for sensor network data.

The James Reserve Data Management System (DMS) database is a successful example of sharing raw streams of sensor data, and SensorBase.org's schema design is influenced by it [14]. Many applications are built using DMS, including Google Earth.

The Deployment Analysis System (DAS) tool is another example of sharing data as it presents environmental and system metric visualizations directly to users [13]. DAS contains its own database and a web interactive front-end. In addition to storing environmental data, the DAS database is especially tuned to hold system metrics (from the Sympathy debugger) to quickly generate useful diagnostic visualizations for debug-

ging purposes. DAS is currently used for many deployments, and there are plans to migrate information in the DAS database to SensorBase.org.

3. Background

During the early phases of deployments, data management was not an issue since there were only a small number of projects, most of which were short-lived and did not produce unmanageably large amounts of data. However, as more motes start to gather more data points, and the type of deployments start to diverge and became more complex, the simple mechanism of storing log files in various file servers poses challenges to people who need to share data.

In order to give context to the motivations behind SensorBase.org, we briefly explain the overview of UCLA CENS' sensor network architecture and describe our deployment experiences in the past several years.

A common data flow mechanism used in UCLA CENS is the 2-tier heterogeneous architecture. Low powered 8-bit CrossBow Mica2 motes forward environmental data as well as system metrics (by using Ramanathan's Sympathy debugger) to a low-power 400Mhz Stargate sink running Linux [10]. The motes use OS-like libraries from TinyOS, while the sink uses the EmStar environment [11] [12]. ESS2 is a layer that spans both the motes and microserver base-stations that provides interfaces for controlling data samples, transformations, and data collection [9]. It also includes energy-efficient routing algorithms and sensor interface drivers.

Using various software components described above, motes forward data to a local sink that queues up data in the form of log files stored on the compact flash card. When internet connection is available, the sink automatically uploads data to various file servers using 802.11b, ethernet, or GPRS (General Packet Radio Service for cell phones). As a method of last resort, when an internet connection is not available, data are gathered manually from the field.

In the past few years UCLA CENS experimented with deploying different sensor networks at various locations, some of which include James Reserve preserve in the San Bernadino Mountains, a water treatment facility in Palmdale CA, the Mildried E. Mathias Botanical Garden on the UCLA campus, and even a rural agricultural region near Bangladesh. Our early attempts at data management entailed uploading various time stamped log files back to UCLA. Individual users who need to use the data would download relevant log files stored at different locations, parse the log files each with a slightly different format, and regenerate outputs in the format that specific analysis software requires. For example, a

few common tasks users perform repeatedly include downloading various files to a local storage, reparsing those files, and outputting data in the standardized gnuplot or Excel format. In another example, data-driven programs such as DAS (Deployment Analysis System), which offer near real time displays, also have to reparse files in near real-time, and reinsert data points in their own application specific databases for event triggers and visualization [13]. Currently DAS is used in many current deployments and contains over 5 million data points.

4. Design

Based on our past deployment needs and a few years of data sharing experiences, several goals are considered and prioritized. First, users prefer gathering and publishing data in a repository without having the overhead of setting up a file server, a web server, and access control mechanisms. Secondly, users prefer querying for data easily without having to search and repeatedly perform post-processing. Thirdly, administrators of the data repository prefer projects to be self-managed in the group and still be able to micromanage data, meta-data, and users when necessary.

We considered setting up a file server with a web server such as Apache2.0 and create unix accounts for the different projects/deployments. Each user would simply use scp or use HTTP POST to upload well defined XML/EML formats on his/her own respective directories. For access control, users would setup their own .htaccess-like mechanisms from the web server. While this method is simple and straightforward, it does not check or enforce users to publish files in a standardized format. In addition, search and retrieval for particular data sets is difficult and cumbersome. Lastly, managing a growing number of users and data points is time consuming and tedious for the administrator.

To address the issue of ease of query and retrieval, we also considered setting up a separate database schema for each project. Creation and setup of unique schemas for specific projects however, is time consuming for both the users and the administrator. In addition, schemas may diverge so much that users may no longer see a uniform and efficient interface to retrieve data.

Lastly, we considered building a centralized repository that is similar to popular blog (web-log) sites. This solution creates a uniform and centralized storage that allows automatic checking and enforcement of well-formed data types in well specificed csv or XML/EML files. The uniform data also allows users to make fine-grained queries. Other advantages include creating Wiki-like interfaces that allow distributed user, project,

and data management. A serious disadvantage in this approach is that a significant amount of resources needs to be invested initially to create both the backend and the front-end. Other challenges include making the system scalable and flexible to contain different data types. Despite these challeges, we think that the initial investment is worthwhile given the growing demands of sensor networks. Our solution, SensorBase.org, is based on the above considerations.

4.1 Implementation

SensorBase.org uses common light-weight off-the-shelf software components consisting of MySQL 5.0, PHP libraries, and common UNIX utilities.

4.1.1 Interfaces. SensorBase.org access requires a browser connected to the Internet. The user first logs into SensorBase.org, and has options to create new projects, measurement types, and sensor types. The user also can invite new users, set permissions, and other options. Currently each project can contain one or more measurement types (temperature, humidity, voltage). Each measurement type is associated with a specific sensor type (brand, conversion routine). A key feature of SensorBase.org is the use of geocodes, which enables other users to easily make queries based on geographic locations.

Currently, the search feature is primitive and resembles SQL queries. However, it is constantly evolving as users demand for more query expressiveness. By default the query returns data sets in HTML, but also allows users to query for raw comma separated text entries, which allows applications to more easily interface with SensorBase.org. XML and EML output generators are being implemented.

Uploading data to SensorBase.org is done using the standard HTTP POST mechanism. The user needs to first write a meta-data file that describes the format of a data file being uploaded, as well as other relevant information such as project measurement types and sensor locations. During an upload process a user uploads both a meta-data fuke and a data file to SensorBase.org. To automate the process in a script or cron job, a user can utilize the curl utility which can perform HTTP POST from the command line. Like SensorBase.org queries, the upload mechanism is ever evolving and may change in the future.

4.1.2 Backend. Aside from normalizing the schema and changing the run-time configuration parameters in the database, we make no attempts to modify, recompile, or optimize the database internally. In

other words, SensorBase.org is an application that treats MySQL 5.0 as a black box.

Currently the SensorBase.org schema consists of two distinct parts– the user and project description tables and the data tables. Project tables describe information about projects such as geographic coverage, project owner, allowable measurements, measurement description, permissions, and other relevant information. User tables describe information about users such as permissions, how many account invitations left, which group a user can manage, and other relevant information. Both user and project description tables use the InnoDB engine to allow consistent and reliable transactions for user management.

Data tables describe information about the data set. They contain a subset of EML's dataTable module specification– logical information about data table entities. The size of data tables dominate SensorBase.org, and because it is heavily read-only data, data tables use the ISAM engine that is designed for the purpose of performance. ISAM tables perform fast read operations with the disadvantage of lacking transactions and not fault-tolerant.

5. Evaluation

We evaluated the performance and scalability of SensorBase.org. User load was simulated based on past access patterns and experiences and projected user growth. In addition to the simulated user load, we evaluated simulated user load on four different schema designs, which SensorBase.org could be based on in the future.

5.1 Simulated User Load

First, we populated all four schemas with an identical set of one year's worth of simulated data. The assumed conditions were are as follows: 20 projects actively inserting data with each project containing 25-35 motes, each mote generating the typical ESS2 data values (temperature, humidity, and voltage), and each data value publishing every 10 minutes. Under these conditions, we generated approximately 97 million data points in the data table.

Using the simulated data in the database, we evaluated each clients' read/write response time, and the server's request throughput to estimate its request upper bound. Since the overhead of HTTP and the Apache web server are negligent compared to the time it takes for each query, we did not consider them in calculations. Each simulated client was modeled to perform some of the common intermittent user access patterns we have observed from DAS; in our case, the simulated clients

performed requests repeatedly and non-stop. Actions included consecutive queries for hourly and daily individual data points, sum, and averages in the 5-15 day range.

Simulation results were dependent on many factors such as the type of hardware, software configuration, and others. The test platform that the database runs on is a single processor AMD Opteron SunFire X2100 with two regular desktop grade EIDE Seagate 300G drives running Linux 2.6.15-1-k1-smp with 3G of RAM. The second drive contains only data and index files. MySQL 5.0 is tuned to use 2G RAM; the rest of the MySQL parameters are simply doubled from the standard my.huge.cnf parameters that came with the distribution. Simulated client requests were run on another lighter weight machine.

5.2 Schema Designs

In the early stages of development, we found it acceptable to create a simple schema for data values that are simple to use. We did not want to optimize too early without knowing more about the different types of data inputs that users may store in the future. As the number of requests increases, different optimization techniques will be explored to improve performance.

SensorBase.org currently uses a simple indexed table to hold environment data. It contains attributes such as measurement date/time, measurement type (sensor_id), raw value, value, upload date/time, location id, project id, value, and user id. While this approach is generic and simple to use, there are many drawbacks. First, because it is a generic table, it is not normalized for all sets of possible input. For example, if a particular project contains several data values that correspond to a single time stamp, the table would contain redundant time stamps. In addition, because data is not normalized for particular inputs, the number of rows increases tremendously which causes more memory to be used. Lastly, access time increases significantly.

To evaluate other design possibilities, as mentioned earlier we created four different types of schemas that SensorBase.org may use in the future. The schemas used were: (1) a single table with one index on the measurement date/time (no idx), (2) a single table with indices on columns that are frequently used in the query predicates that includes location id, measurement type, project id, and others (idx), (3) per project specific tables with one index on the measurement date/time (no idx), and (4) per project specific tables with indices on columns that are frequently used (idx).

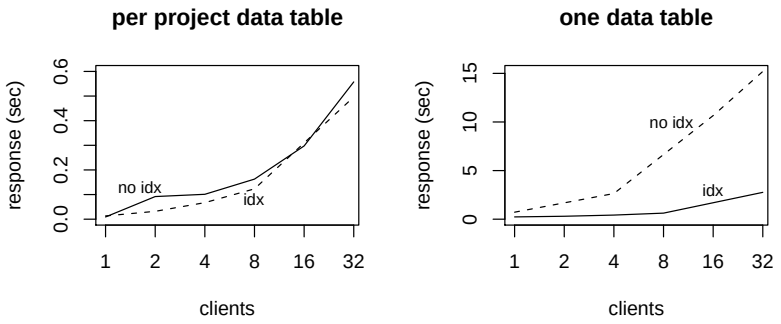

Figure 2. Average Response Time Per Client Request

5.2.1 Client Request Evaluations. Figure 2 shows the response time of each simulated client request, where the clients continuously accessed the database. The x-axis is number of simulated clients, and the y-axis is average number of seconds it takes per request. As was expected, as the number of clients increased, response time increased. The graph on the right shows performance when using a simple table where each row in the table contains one sensor measurement value.

The graph on the left shows the performance by splitting each project into four separate tables normalized for the four different simulated project types. More specifically, each row in the table contains three measurements in the project.

Again, as expected, the performance using one data table was much worse than several normalized project-specific tables. In both cases, indexing on frequently used columns (idx) helped the database search faster than without extra indexing (no idx).

Figure 3 shows server throughput. The number of requests per minute using one single data table (graph to the right) was approximately four times slower than using project specific tables (graph to the left). In both cases, the behavior was predictable– as the number of clients increased, the throughput of the server increased to a point, and then decreases due to context switch, resource conflict, and other factors.

Since SensorBase.org currently uses only one data table, the server throughput upper bound, based on past client patterns and the configuration assumptions stated above, was on average approximately 900 requests per minute, or 15 requests per second. Obviously, there is much room for improvement.

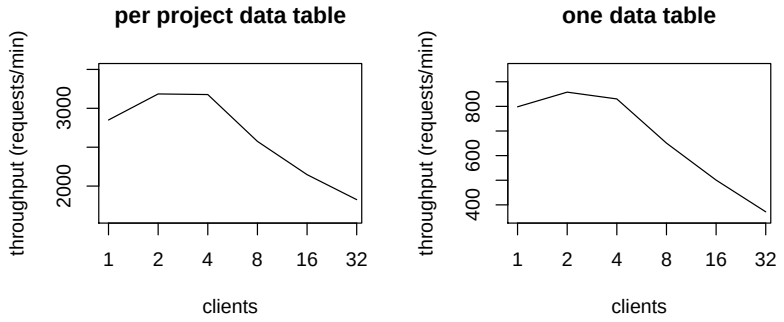

Figure 3. Server requests per minute

6. Our Vision

SensorBase.org shares sensor network data. However, raw data values alone are not very useful to individual users unless higher level applications use them and present them to users in a more user-friendly and meaningful manner. We hope SensorBase.org is the vehicle for application programmers to use. Some of the applications that could interface with SensorBase.org include Google Earth, Google/Yahoo Maps, ArcGIS, and RSS (publication of summaries, various indexing).

As the number of users increases, performance will guide many future design decisions. From the evaluation we see that properly designed schemas specific for projects can increase performance tremendously. While creating project specific tables optimizes for speed and space, application and PHP programmers may need to create more complex queries (using UNIONS) that span across different tables per query. This can be alleviated using MySQL 5.0's VIEW, stored procedures, or both.

In addition to performance, usability remains a high priority. The interface will continue to evolve to meet the demands of users. Likewise, additional features such as binary data, pre-aggregated data, data event triggers (RSS-like mechanisms), and many others will be pursued.

Prior to SensorBase.org we considered and experimented with several existing methods for managing sensor network data and finally decided to create an in-house solution to meet our needs. We have and will continue to invest a significant amount of resources in improving SensorBase.org and we hope it is a decision that contributes positively to the sensor network community.

7. Acknowledgments

We thank Eric Graham, Richard Guy, and countless number of people in the CENS lab for invaluable inputs and suggestions.

References

[1] http://base.google.com

[2] Ramesh Govindan, Joseph M. Hellerstein, Wei Hong, Samuel Madden, Michael Franklin, Scott Shenker "The Sensor Network as a Database," USC Technical Report No. 02-771, September 2002.

[3] Samuel Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong, "Tinydb: An acquisitional query processing system for sensor networks," *Transactions on Database Systems (TODS)*, Mar. 2005.

[4] http://www.cs.cornell.edu/database/cougar/

[5] Phillip B. Gibbons, Brad Karp, Yan Ke, Suman Nath, Srinivasan Seshan "IrisNet: Internet-scale Resource-Intensive Sensor Network services. A core architecture and software infrastructure for a worldwide sensor web," Overview paper in IEEE Pervasive Computing: IrisNet: An Architecture for a Worldwide Sensor Web

[6] Hans Knud Arndt, Thomas Bandholtz, Oliver Gunther, Maria Ruther, Thomas Schutz "EML - the Environmental Markup Language," http://www.bandholtaz.info/publications/2000-ISESS-EML.pdf, July 2000.

[7] http://sev.lternet.edu/

[8] CRAWDAD, http://crawdad.cs.dartmouth.edu/.

[9] Richard Guy, Ben Greenstein, John Hicks, Rahul Kapur, Nithya Ramanathan, Tom Schoellhammer, Thanos Stathapoulos, Karen Weeks, Kevin Chang, Lew Girod, Deborah Estrin "Experiences with the Extensible Sensing System ESS," in proceedings of *CENS Technical Report #60*, March 2006.

[10] Nithya Ramanathan, Kevin Chang, Rahul Kapur, Lewis Girod, Eddie Kohler, Deborah Estrin, "Sympathy for the Sensor Network Debugger," in *SenSys, November 2005.*

[11] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David E. Culler, and Kristofer S. J. Pister, "System Architecture Directions for Networked Sensors," in *Architectural Support for Programming Languages and Operating Systems*, 2000, pp. 93–104,

[12] Lewis Girod, Jeremy Elson, Alberto Cerpa, Thanos Stathopoulos, Nithya Ramanathan, Deborah Estrin "EmStar: A Software Environment for Developing and Deploying Wireless Sensor Networks,", in *USENIX Annual Technical Conference*, General Track 2004: 283-296

[13] Kevin Chang, Nithya Ramanathan, Deborah Estrin, Jens Palsberg. "D.A.S. – Deployment Analysis System,", in *Sensys Demo: 301*, 2005.

[14] M. Wimbrow, Eric Graham "http://dms.jamesreserve.edu/"