Demo Abstract: Efficient Data Retrieval for Interactive Browsing of Large Sensor Network Data Sets

Matthias Keller and Jan Beutel
Computer Engineering and Networks Laboratory
ETH Zurich, Switzerland
{kellmatt,beutel}@tik.ee.ethz.ch

ABSTRACT

Data gathered from sensor networks can easily reach the dimension of millions of data points. While there exists a great variety of sophisticated components for web-based data visualization, efficient and timely data retrieval is a non-trivial problem when dealing with large data sets. In this paper, we present a client-server architecture for a web application that allows a user to interactively browse through thousands or even millions of data points without having to sacrifice detail.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems

General Terms

Management, Performance, Measurement

1. INTRODUCTION

Wireless sensor network (WSN) applications typically generate large amounts of data. Examples are readings of the sensors, samples of the internal state of a sensor node, and network performance indicators. Considering only one signal, *i.e.*, battery voltage, a single sensor node that samples its health status every 2 minutes will generate 262,800 battery voltage samples per year. Depending on the signal, there are several use cases that require reading this data on different scales. For instance, a user looking for long-term trends is interested in (possibly aggregated) data from the maximum available time range. In contrast, a user trying to locate a specific artifact needs to browse data in the highest available detail. The number of data points to be considered rapidly adds up to millions when displaying multiple signals in one view.

Considering more than 50 million packets that have been collected from the PermaSense [6] Matterhorn deployment in the past 2.5 years, there is the necessity for a visualization tool that is designed and thus optimized for browsing large amounts of data.

PermaSense currently uses the web interface provided by GSN (Global Sensor Network) [1] to offer data access [9]. Based on experiences, it is definitely favorable to use lightweight web technologies instead of using heavyweight applications, *i.e.*, MATLAB, that ask for high customization efforts and possibly come with license costs. Several other projects are following the same design choice, *i.e.*, the public data interfaces [4, 11, 10] of GlacsWeb [8], SensorScope [2], and Powertron [7] are also web-based. However, to the best of our knowledge, existing solutions do not support to browse through complete, large data sets at the finest detail by solely using a zoomable user interface. Instead of limiting the amount of transferred data by either limiting the displayable time range or only showing aggregated data, we propose to enhance the capabilities of such a tool by employing sophisticated on-demand data fetching and caching strategies.

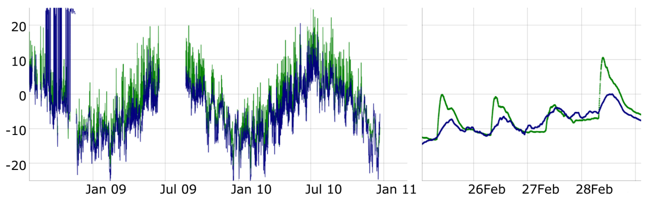

Figure 1: Temperatures measured by two sensor nodes. Access to highest available detail is crucial when analyzing rapidly changing signals. Graphs are generated using *dygraphs* [3].

In this paper, we propose an architecture for interactive exploration of large data sets: 1) A user can access the application from everywhere using a web browser, the data processing logic is located on an Apache Tomcat application server running Java. 2) Available sensor signals are arranged in a hierarchical directory that mirrors the system architecture of our sensor network application. 3) Data can either be retrieved as CSV (comma separated values) or in the Google DataTable [5] format. This allows to choose from a variety of available software components for visualization. 4) Optimized data fetching and caching strategies allow the use of a zoomable user interface for interactively exploring large data sets down to the lowest detail level of single data points. 5) Only little customization efforts are needed for adapting our solution to existing storage infrastructures used in other WSN applications.

2. SYSTEM ARCHITECTURE

Client-server architecture. An exemplary session is depicted in Figure 2. A user starts a session by accessing the application server with a web browser (1). The user interface included in the response (2) then requests data by sending back a pre-defined query to the application server (3). The application server processes the request, fetches relevant data from memory caches, and sends the sensor data to the client (4). A few moments later, a zoom interaction leads to a new data request (5). The requested sensor data leads to a cache miss, thus the data must first be fetched from the database server (6, 7) before it can be sent to the client (8).

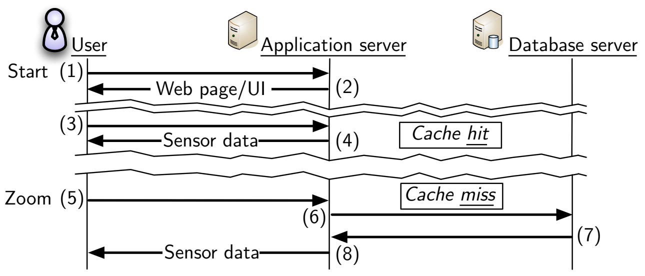

Figure 2: Exemplary session of a user loading the service and then issuing a request for sensor data at a higher detail.

Sensor signal directory. We propose to arrange available sensor signals in a hierarchical directory structure that mirrors the system architecture of PermaSense, and is similarly applicable to the architectures of other WSN applications: A multitude of different sensors, *i.e.*, temperature gauges, or weather stations, are installed at several deployment locations. Each deployment location can be described by a unique name and a set of numbered positions that correspond to geographic locations on site. There is at most one sensor of a certain kind per position, a sensor is always connected to a networked embedded system, *i.e.*, a wireless sensor node, or a server integrated in a sensor. A certain sensor signal can either be addressed by the identifier of a position within a deployment, *i.e.*, `matterhorn/position/0/vaisalawxt520/air_temperature`, or by the numerical identifier of the related networked embedded system, *i.e.*, `matterhorn/device/2055/dozer_nodehealth/sysvoltage`.

Data aggregation. A user can browse data in a fixed number of detail levels, data of higher detail is dynamically loaded on request. Data of the highest level of detail corresponds to single data points, data of all other detail levels is computed by aggregating a number of data points. The current implementation aggregates data by averaging the values of all data points that are within a sliding window, other aggregate functions can be plugged in as well. For explaining the used sliding window function, let us consider a data point i consisting of a timestamp $t(i)$ and a measured value $v(i)$. Given a window length l, we first calculate truncated timestamps $t_r(i) := \lfloor t(i)/l \rfloor \cdot l$. The value $V(j)$ of an aggregated data point j having the timestamp $T(j)$ is then given by

$$N := \{n | t_r(n) \equiv T(j)\}, V(j) := \frac{1}{|N|} \cdot \sum_{n \in N} v(n)$$

The window length l for each level of detail is given by the rate used for sampling data, the maximum number of data points per sensor signal to be transferred to a client within one request, and the maximum duration of the returned fraction of a signal. For instance, data collected within three years by sampling every 2 minutes is reduced to 3,600 data points when using a window length of 7.3 hours. Currently used four window lengths range between 7.3 hours and 12 minutes, data of highest detail is not aggregated.

Interface to existing storage infrastructure. The majority of the small customization effort needed for integrating our solution into existing systems is given in the access to the existing storage infrastructure. In our case, this corresponds to deriving SQL statements from the user input consisting of selected sensor signals and a desired level of detail. This procedure is straightforward and thus not explained here in further detail.

```
double values[SIZE];
double startTime;
int    windowLength;
```

Figure 3: Data structure used for caching aggregated data. *SIZE* is a constant passed once at initialization. Time information is implicitly given by the position of a value in *values*.

Caching. The proposed data aggregation method can be performed within a single query on a standard DBMS. However, the performance of such a query is very poor since aggregating data based on calculated properties (truncated timestamps) requires the use of temporary tables that are usually located on slow disks. In our tests with MySQL 5.1, queries of this kind often took up to 10 seconds and thus too long for a satisfying user experience. Although having the option to optimize the DBMS, highest performance is achieved when caching data on the application server. Caching applies only to aggregated data, the data structure used for storing a single sensor signal of a certain detail level is depicted in Figure 3. Only the timestamp of the aggregated point with the smallest timestamp is stored, timestamps of other data points can be easily derived by adding the array index multiplied by the window size. This data structure requires only little space in memory while allowing to efficiently read and update values.

Cache updater thread. A background thread running on the application server is in charge of 1) filling cold caches after a restart of the application, and 2) periodically updating caches by incrementally adding recently arrived data points.

3. ACKNOWLEDGEMENTS

The work presented was supported by NCCR-MICS, a center supported by the Swiss National Science Foundation under grant number 5005-67322, and the Swiss-Experiment, a project funded by the Competence Center Environment and Sustainability of the ETH Domain (CCES).

4. REFERENCES

[1] K. Aberer, M. Hauswirth, and A. Salehi. A middleware for fast and flexible sensor network deployment. In *Proc. 32nd Int'l Conf. Very Large Data Bases (VLDB '06)*, pages 1199–1202, 2006.
[2] G. Barrenetxea, F. Ingelrest, G. Schaefer, M. Vetterli, O. Couach, and M. Parlange. Sensorscope: Out-of-the-box environmental monitoring. In *Proc. 7th Int'l Conf. Information Processing Sensor Networks (IPSN '08)*, pages 332–343, 2008.
[3] dygraphs JavaScript Visualization Library. http://dygraphs.com/.
[4] GlacsWeb Iceland Graphs. http://env.ecs.soton.ac.uk/glacsweb/iceland/graph/.
[5] Google Visualization API. http://code.google.com/apis/visualization/.
[6] A. Hasler, I. Talzi, J. Beutel, C. Tschudin, and S. Gruber. Wireless sensor networks in permafrost research - concept, requirements, implementation and challenges. In *Proc. 9th Int'l Conf. on Permafrost (NICOP '08)*, volume 1, pages 669–674, 2008.
[7] M. Kazandjieva, O. Gnawali, and P. Levis. Visualizing sensor network data with powertron. In *Proc. 8th ACM Conference on Embedded Networked Sensor Systems (SenSys '10)*, pages 395–396, 2010.
[8] K. Martinez, R. Ong, and J. Hart. Glacsweb: a sensor network for hostile environments. In *Proc. 1st IEEE Int'l Conf. Sensor and Ad Hoc Communications and Networks (IEEE SECON '04)*, pages 81–87, 2004.
[9] PermaSense Public Data Interface. http://data.permasense.ch.
[10] Powertron Home. http://powernet.stanford.edu/powertron/.
[11] SensorScope Wannengrat Data. http://sensorbox.epfl.ch/main/drawData.php?d=wannengrat.