# Shortcomings of the Existing Life Under Your Feet

# Data Processing Pipeline

August -8-2011

Jayant Gupchup

gupchup@jhu.edu

## Section 1: Introduction

The goal of this document is to describe the architecture, design and shortcomings of the data processing pipeline for the data collected by the Life Under Your Feet [LUYF] sensor deployments. The document also discusses ways in which the shortcomings of the pipeline can be improved. This data pipeline was developed as an end-to-end system that store and processes data collected by the LUYF networks. We begin by discussing the overall architecture of this pipeline at a high level and then describe the salient aspects of various components. Based on over four years of deployment experience, involving a spectrum of deployment conditions, we understand the drawbacks of this system and discuss various options and ways to make the system more robust and improve on its overall operations. The main focus of the pipeline from the very beginning is to develop a system which minimizes the involvement of environmental scientists that have little or no knowledge of managing data at medium to large scales. The scientists are provided access to the data where they can visualize and download the data using a web-based system.

## Section 2: Overall Architecture

The architecture of the data pipeline is shown in Figure 1. We begin by understanding how data is transformed as it is moved from one end of the pipeline (stored in motes) and delivered to the other end (the database accessed by scientists).

We download the data stored in the motes every 6 hours using an ultra low power data retrieval mechanism known as “Koala” [KOALA]. The gateway collects and uploads the data to a web server using a HTTP post. The authenticity of the data is protected using an HMAC-SHA-1 signature. Upon authentication, the raw data is parsed and pushed to a stage database of that deployment. A stage database is created for every deployment. Figure 1 shows the complete end-to-end delivery of data.

Typically, many different sensors are connected to each mote. Sensors are referenced by their hardware id and the motes are referenced by a box id. Every unique location, referenced by the location id, hosts a mote to which different sensors are attached at one of four available channels. This location-box-sensor arrangement is recorded during the time of deployment by scientists (or field technicians) and entered in to a web-based portal. We refer to this information as “metadata” and it is important for converting the sensors’ raw measurements to physical values and for provenance purposes. Our system ensures that unless this information is entered in the database, processed data will not appear at the other end of the pipeline.



Figure : The overall architecture of the LUYF data processing pipeline



Each mote records sensor measurement in terms of its local timestamp (using its local clock). The stage database performs the following roles:

1. Re-orders the data received from the basestation appropriately and assigns each record a data block. We refer to this process as “segmentation”;
2. Assigns a global time stamp to the sensor measurements by a process of post-mortem reconstruction of timestamps;
3. Decouples the data from sensor channels connected to the box and assigns logical sensor ids to these measurements; and
4. Converts the raw sensor measurements to physical values using the sensor type information recorded in the metadata.

Based on the data stored in the stage database, automated reports are generated periodically (e.g. every three days) that monitor the health of the deployment (hardware, network connectivity etc). These reports are used to replace malfunctioning sensors and troubleshoot network connectivity to ensure that our scientists obtain a high data yield.

The stage database next pushes the converted values to a unified database known as the “Science Database”. The science database contains and hosts data from all the LUYF deployments.



Figure : Various steps involved in transferring data from the basestation to the stage database via the web serrver



The science database performs the following roles:

1. Resample’s the data based on the scientific goals
2. Stores data at various levels of detail (hourly, weekly, daily). We refer to this as a “Data pyramid”
3. Detects and flags measurements that appear to be faulty or erroneous
4. Exposes the data to the scientists in a format that can be consumed easily by them via visualizations or flat files (comma separated value files).

## Details of the Sub systems

In this section, we will describe three main sub systems of the data processing pipeline – The upload application, the stage database processing and the science database processing. The focus will be to provide a high level overview and the design philosophies of these subsystems. The details of the implementation are beyond the scope of this document.

### Upload Application

The basic mechanism for the data transfer from the gateway to the stage database is shown in . The process of uploading the outstanding data from the basestation to the stage database involves a process of handshaking described in the following text. The gateway requests the web server for the last known download (given by download timestamp) for each of the boxes that are known to be out in the field (step 1 in Figure 2). The web server or the upload application responds with a table (after consulting the database) containing <boxid, download ts> (step 2 in Figure 2). The “download ts” refers to the download timestamp (as recorded by the gateway) of the last known record available for each box at the stage database. This table lets the gateway know how much data is outstanding for each box from the database’s perspective. The gateway then puts together one file containing all the outstanding data from the various boxes and sends it back to the web server (step 3 in Figure 2). The upload application accepts this file if the HMAC-SHA-1 signature of the file matches the one computed locally at the web server. It then parses the contents, stores the file on the web server and pushes the contents to the stage database (step 4 in Figure 2).

The basestation uploads data as a collection of three types of records. The bulkiest of these are the actual sensor measurements that are recorded at regular intervals by the sensors connected to each box. These types of records are called “FlashRecords”. The second type of records contains information that let us assess the health and status of the mote. These types of records are called “JournalRecords” and are collected by the basestation whenever it comes in contact with a box (typically during download rounds). They help us diagnose the health of the motes and also inform us the motes time state (its local clock and number of reboots). Each box collects information (LQI, RSSI) about its links to other boxes and this information is stored by the basestation. The basestation, in turn, uploads this information to the stage database in addition to the path selected to download (or harvest) data from the boxes. These types are records are called “NetworkRecords”.

### Stage Database

The high level roles of the stage database were highlighted in Section 2. Over here, we will go into the details of these functions. The transformation of data in the stage database is represented in Figure 3.

Segmentation: The first processing step is referred to as “Segmentation”. A segment is a collection of records (of a given box) where the local clock is monotonically increasing. We say that a box has started a new segment when its logical clock resets. Upon resetting, the mote increments its reboot counter. A motes reboot counter is reset when a new version of the code is installed on the mote (denoted by installation time). Each segment is identified by its reboot counter and installation time. Each record is identified by its local timestamp and the segment number. This process of giving each record a segment number is referred to as segmentation. Prior to December 2009, the boxes did not record the reboot counter and installation time so these two values had to be inferred from the data itself. Hence there are two implementations of the code that identifies and assigns segments to data records.



Figure : Various processing steps involved in the stage database

Timestamp Reconstruction: Once the records are assigned a segment, we proceed to assign them a universal timestamp – we refer to this procedure as postmortem timestamp reconstruction. The segmented data is combined with the time-state information stored in the JournalRecord records. This time-state information is essentially a pair of values – first is the motes local time, and second is the basestation’s universal time at the time when the mote was contacted by the basestation. These pairs are referred to as “anchor points”. Using these anchor points, we can map the motes local time base to the global time scale as they are related linearly. In practice, we need to identify the segments that each anchor point belongs to and obtain one mapping (also known as a fit) for each segment using linear regression. After obtaining such a mapping for a segment, all the records belonging to that segment are translated to their corresponding global timestamps.

Channel Decoupling: Recall that sensors are connected to motes. In order to maximize the flexibility, sensors can be connected to any of the available sensing channels. To be more specific, the LUYF motes have 4 sensing channels to which external sensors (soil temperature, soil moisture and CO2) can be connected in no fixed order. Given this design, it is important to have a map between the sensing channels and the sensor types for each mote. This map (henceforth referred to as metadata) is created at the time of recording the location-node-sensor information as described in Section 2. Now, the time corrected records contain box number followed by the ADC (raw) values associated with the sensing channels. The metadata information needs to be referenced to unpack the record and identify the sensor information associated with each sensing channel. The result of this process is stored in a table (RawData) where each record contains three columns – sensor id, timestamp and raw (ADC) value.

### Calibration: Records stored in the RawData table need to be converted to their physical values. For instance, 1023 ADC corresponds to 0.31([0 1]) in terms of soil humidity. This transformation is referred to as calibration. A calibration function is defined and stored in the database as a stored function for each sensor type. The metadata table is used to extract the sensor type of each sensor and this information is used to invoke the appropriate calibration function. The results of applying the calibration function are stored in a table referred to as Calibrated data. This data is then pushed to the unified science database.

## Science Database

While the stage database undertakes tasks that are tightly coupled to the architecture of a sensor network, the science database performs tasks that can be applied to environmental data streams in general. For example, data collected from weather stations can be stored in the science database and treated the same way as data collected by the LUYF sensor network. The various stages in the science database are shown in Figure 4.

Time Gridding: Data needs to be presented at scales that are relevant to the scientific question at hand. Co-located sensor streams often collect data in an asynchronous manner. For example, a flux tower and a weather station might be operated by two different institutions and hence their data collection rate and schedules might be completely different. Scientists, on the other hand, prefer to work with data that is aligned in time across the sensors. Let us consider an example in sensor networks where time gridding plays a role. Motes collect data based on their independent schedules. As an example, the schedules of two motes collecting data every 10 minutes might look something like this:

N1 : 5m, 15m, 25m …

N2 : 3m, 13m, 23m …

These schedules represent minutes past the hour mark. The scientific goals of a project might require the data to be delivered every half hour. Time gridding is a process where data is interpolated at pre-defined points in time (E.g. half hour past the hour, the hour mark and so on). A SQL stored procedure takes the grid interval (step size in terms of time) from the Site table, computes these pre-defined intervals and stores them in the TimeGrid table. Data from the CalibratedData table is interpolated around points defined in the TimeGrid table. The results are stored in the DataSeries table where the data from all the sensors belonging to the same deployment are represented at regular and uniform points in time.



Figure : Various processing steps involved in the Science database

Pyramiding: The amount of data gathered by long-term environmental monitoring projects can be large. Based on our experience in working with environmental scientists, it is important for data to be visualized quickly and an interactive fashion. We implement a simple and interactive scheme to return data in a layer manned inspired by the way google maps render images. The overall goal of this scheme is to limit the number of data points being visualized (since the screen resolution is fixed) without losing the prominent features of the data. The time period of interest is provided as an input. The number of points (NUM) returned to the user is a variable and the default value is kept small (E.g. 1024 points). This time period is chopped up into NUM equally spaced periods to create a custom time grid. We then interpolate around these custom time points and this result is thrown into a visualization module. This scheme is particularly useful when the users are exploring the data. They can change the time period (zoom in/zoom out) and visualize information effectively and quickly. In practice, the interpolated data is pre-computed and stored in the DataSeries at various zoom levels – This concept is inspired the image cutout application in SkyServer. Once the time period is provided, data interpolation is done by retrieving data at the nearest available zoom level – This minimizes the amount of data that needs to be interpolated on the fly and improves the interactivity (speed).

Fault Detection: Environmental sensors are subjected to harsh conditions often resulting in short-term and long-term faults in the data. These faults can significantly affect the quality of data visualization and analysis. Sensors exhibit a strong degree of correlation in space and time and these correlations are exploited to determine the integrity of the sensor measurements. Values that deviate significantly from their expected behavior are flagged.

Data Access: The Data is made available via a web-based portal referred to as Grazor [GRAZOR]. Grazor allows data to be visualized quickly using the pyramiding mechanism. Scientists typically browse around and are given the ability to bookmark data that is of interest. They can also download the data (as comma separated values) at the lowest level of the data pyramid.

## Section 3: Shortcomings of the pipeline

The data pipeline was designed based on experiences from two early pilot deployments. After having used the data pipeline to process and store data for over four years, a number of opportunities to improve the efficacy have presented themselves. These shortcomings are listed below:

1. Lack of flexibility in registering metadata
2. High maintenance costs for the upload application
3. Failure to monitor the health of the deployment effectively
4. Inability to deal with sensor faults for different modalities

We will now look at the details of these shortcomings and discuss the experiences we have had over the course of these four years.

Metadata Inflexibility: The importance of metadata has been highlighted in Sections 2 and 3. At the moment, the site-location-node-sensor metadata information is recorded during a deployment on field books. This information is then entered into the system using a web-based system. The existing metadata system has posed problems at a number of different levels. They are listed below:

1. A typical outdoor deployment is very chaotic and resource intensive due to the manual labor involved. During this process, researchers tend to focus on the immediate logistical challenges of setting up the deployment. Although collection of metadata is crucial, its value is only realized once the data needs to be looked at. Thus, metadata recording is not prioritized during the deployment resulting in inconsistencies in the final output.
2. Our deployment experiences in Ecuador have taught us that researchers often change the channel assignment during the middle of the deployment without notifying the database administrators. This results in the application of an incorrect calibration function to the raw stream for the channels that have changed. For example, let’s consider that mote X is connected with a soil moisture sensor on channel 1. 10 days into the deployment, it is determined that a soil CO2 sensor is more valuable at that location. Let’s say that this sensor replacement to channel 1 is not notified to the system. As a result the system assumes that data on channel 1 of mote X is coming from a soil moisture sensor. On a number of occasions, the data recorded on log books or loose papers in the field are often misplaced amidst deployment pressures and hence this information never gets entered into the system.
3. The system assumes that the location information needs to be collected during the deployment. This information is difficult to collect unless there is access to an accurate GPS or surveying equipment (which is expensive and labor-intensive). The lack of this information makes it difficult to show the data on a map. A related issue is that the system assumes that the external sensors connected to the box are co-located with the box but may differ in terms of their depth. At USDA, we encountered situations where the same box had sensors that were collecting data at the same depth at two different locations. This configuration cannot be represented effectively in the current schema.
4. Once changes in hardware configuration are made available to the system, the system should delete data resulting from the wrong calibration curve (data processed prior to the hardware change notification) and reprocess all the data with the correct calibration curve. At present, the system does not do this. This shortcoming is relatively easy to fix and can be implemented easily by making sure that upon detecting a hardware change, the system processes already processed data with the correct calibration curve.

### Upload Application Maintenance: The details of the upload application were described in Section 2. A major shortcoming of the current design is that two separate code bases need to be maintained – one for the basestation and another one for the upload application. Whenever the architecture changes, code needs to be updated at both places. This design proved costly during the Brazil deployment where we decided to compress records. Code was changed on the mote, the basestation and the upload application. The former two code changes are inevitable but the last one could be easily avoided if the basestation was set-up to speak with the database directly. Thus the lesson learnt here is that the maintenance costs would go down drastically if we maintain one code base at the basestation which can talk directly with the database server and upload the outstanding data directly.

Health Monitoring: Hardware and software failures are to be expected in an outdoor sensor deployment. In order to maintain a high yield, these failures need to be caught early and timely action is vital. Currently, reports are generated periodically that are looked at by members of the LUYF team – typically by developers of the group. Our experience has been that developers are unable to remember to look at these reports. On a number of occasions, networks have been left unmonitored for a number of days, during which time, a number of hardware and software failures have occurred resulting in loss of significant amount of data. For example, In the SERC deployment, a software bug caused the network to be awake for more than 15 days (Nov 19, 2009 – Dec 6, 2009). This resulted in draining the batteries for most of the motes thereby killing the entire network. If we had detected this early, we could have fixed the software bug and saved the network!

Dealing with Sensor Faults: Over the years, we have shared data with a number of scientists and a common denominator in their feedback has been that the data contains a lot of faults. Data gathered from sensor networks tend to be inherently faulty and noisy and the failure modes are highly unpredictable. At present, a simple median filter [MAD] is used to detect faults. This method is used because of its wide applicability - considering that the data collected by our network comprises of a number of modalities.

On the one hand, the scientists are in the best position to understand and pick out the faulty data points effectively due to their domain expertise. On the other hand, most environmental scientists do not have experience to deal with this much data to automate this process effectively. Furthermore, scientists prefer to invest their time analyzing the data rather than cleaning the data. Another aspect that makes this task particularly challenging is that modalities tend to respond differently to the changing environment. Therefore, picking a set of features that is effective for all modalities has been proven to be rather challenging.

## Section 4: Discussion on overcoming the shortcomings

In this section, we will discuss some possible solutions for overcoming the shortcomings mentioned in Section 3.

Metadata Inflexibility: The primary cause for the inconsistencies in the metadata lies with the premise that scientists and administrators can be depended upon to register hardware configuration changes. A design that auto detects changes in the metadata is more effective because it cuts the human out of the loop. The metadata changes can be stored on the mote as a new record type. While processing such a record, the system can directly update the metadata tables in the database. Although this takes care of challenge (a) and (b), it does not address challenge (c). It is not clear to me at this point how we will solve this problem. The solution for (d) has been discussed earlier.

Health Monitoring: A number of approaches have been taken to ensure this problem is not encountered. One such approach was that each student was assigned one deployment to monitor and report on at weekly meetings. This system failed as meetings stopped and students got busy with other tasks. The main problem here is that the responsibility of monitoring the health of the system lies with software developers and not people who would use the data. My personal opinion is that users (or owners) of the data need to take initiative in ensuring that the system is healthy and the data being received is devoid of problems. The developers could provide technical assistance in ensuring that the data owners have adequate tools to monitor the system. Although some tools already exist, they need to be refined and tuned. Another observation is that alerts need to be generated and sent via communication systems (such as email, text message) that are used daily. In the past, we tried to use twitter to follow the health of the system. After a while, people stopped following the twitter feeds and the monitoring stopped.

Dealing with Sensor Faults:

Summary of what the document contains

* Overall Architecture
* Description of various modules
* Shortcomings of the pipeline
* Suggestions for ways to improve the pipeline