StreamFS: An analytical framework for physical data

Jorge Ortiz
Computer Science Division
University of California, Berkeley
jortiz@cs.berkeley.edu

Abstract

1 Introduction

Fundamentally what we want to represent is our knowledge about the building. Any description of our knowledge ultimately describes a family of subjects and objects and some predicate to indicate the relationship between them. For example, we wish to query about the location of the temperature sensors in room 410. Internally, we must capture the notion of an "is-in" relationship and the notion of an "is-a" relationship. In this example, the sensor *is-a* temperature devices, 410 *is-a* room, and every sensor in 410 would be listed individually as "is-in" room 410. In this example, we want the list of all sensors with an "is-in" relationship with 410 and that have an "is-a" relationship with temperature.

1.1 Filesystem representation

The main challange is buildings is related to data integration. Every building is uniquely described in the various systems and sketches that describe it. Even across the various representations of the building's internals there is no standard naming or labeling convention. This unsystematic approach to capturing the building's state (the subsystem, locations, and sensors) makes it almost impossible to scale any solution for collecting and managing large collections of buildings.

Fundamentally, each of the representations attempt to capture physical and abstract objects in the building and their inter-relationship, however the vocabulary – what the objects and relatinos are actually called – is different from building to building. There is lots of overlap from building to building, but the actual names change from building to building. Therefore any system or user that wishes to run a query must first discover the vocabulary and semantics for each building, uniquely.

The filesystem abstraction fulfills three needs:

1. Vocab and relationship discovery
2. Limiting the view and interation with the underlying objects

2 Motivation

As networked, embedded sensing becomes more and more ubiquitous, it becomes important to formulate new ways to use the data produced by these devices. One can view these devices as fundamentally linked to the physical world, reporting on physical measurement taken in the context of where they are placed. For example, a sensor deployment on the Golden Gate bridge [3] were used to monitor its structural health over time. By taking fine-grained, high-frequency accelerometer readings, the authors were able to formulate a time-varying view of the bridge as a whole. Another important application is in buildings, where sensors on embedded throughout the building environment to measure the ambient condition and to monitor the health of the equipment used to maintain safety, comfort, and security. In either application, the placement of sensors must be known a priori in order to construct a wholistic view of the phenomena being observed.

The building context is particularly challenging in this regard. The GGB project, for example, used 64 nodes spread throughout the bridge. The location of each node was carefully recorded. Buildings, on the other hand, typically have over an order of magnitude more sensors, distributed throughout the building. Sutardja Dai Hall contains over 3000 sensors spread throughout the building, and the number of sensors typically increases with the size (in square feet) of the building. In each case, the location, type, and other information is important to record. This metadata, is crucial for data interpretation. Moreover, the relationship between the sensors, as described through the metadata, serves an even more critical role, as it allows the analyst to construct a holistic view/interpretation of the data. Without it, the data is useless.

Observation 1: The metadata that describes the context of the embedded device, is as important as the data the device produces.

Observation 2: The metadata must be normalized, for each deployment, in order to formulate a holistic in-

terpretation of the measurements.

The phenomena being sensed is time-varying in nature. Each data value has an associated timestamp and theres some work must be done to either synchronized, normalize, or align timestamps across streams before analysis. However, the real challenge lies in long-lived deployments. Long-lived are challenging because the physical environment changes and these changes are difficult to track, at scale. For example, buildings have lifespans that last multiple decades. In that time, the environment and placement of sensors in them, goes through many, many changes. In order to maintain an accurate, continuous assessment of the environment, such changes must be tracked.

Observation 3: Changes in the physical environment must be systematically tracked in order to maintain an accurate interpretation of the data produced by sensors embedded in that environment.

This observation is corroborated in several experimental deployments [references?] and captured in simulation engines, such as EnergyPlus, explicitly. Most deployments collect data throughout the lifetime of the deployment and do both real-time and historical analysis of the data. Designing for the physical changes, then, has deep implications on the design of the system. The first is that there should be mechanisms in place either 1) automatically describe the environment/placement, type, etc. and 2) there are verification processes that continuous check the validity of the metadata. Both are necessary to assure the accuracy of the analysis. Moreover, 3) history must be recorded and accounted for throughout the lifetime of the deployment. This assures that the analyst will place the data in the appropriate context, even as she runs through the historical data.

Finally, soft context should also be systematically tracked. Software changes can cause changes in stream behavior that are an actual source of error during analysis. This should also be captured in the metadata history.

Observation 4: Metadata history is necessary in order to properly audit the environment and accurately perform a historical analysis.

We have designed a system, StreamFS, that address each of these observations for sensor deployments. However, for this paper we focus on the portion of the system that specifically addresses observations 1, 2, and 4. We discuss the part of the system that addresses observation 3, but refer the reader to other papers that specifically address this issue. We also describe the use of StreamFS for managing sensor deployments in buildings. We integrate StreamFS into the Building Applciation Stack [**?**] and benchmark it i a real world deployment.

3 Related work

All the data values are compressed using snappy [2], before insertion into the database.

1. Git version control
   - any change in the metadata is "committed"
   - tags name commits
   - SHA-1 is used to save the contents of a commit operation (i.e. the struct that described the operation)
2. Log-structured file system
   - all changes made to the metadata is recorded as a timestamped operation in the database
   - the details of the operation are stored elsewhere and *only* fetched if explicitly queried
   - The log is replayed from the start time to the end time of a query
3. provenance database systems
4. timeseries databases systems
5. BAS and BOSS
6. spatio-temporal databases

4 Motivating example

Indoor localization work is on the rise again [**?**]. In buildings, the community has realized that coarse grained (room-level) localization is sufficient for most applications. Although challenges remain in boundary-discovery [**?**] and auto-calibration [**?**].

5 Data

In buildings, we collect data about the internal sub-systems, the spatial organization, and the sensors throughout the infrastructure. We also collect data from sensors and use the infrastructural information to organize and categorize the data collected from the sensors. For example, we note which temperature sensors are in which room and which pumps, cooling coils, fans are driven by their readings. The former denotes physical placement, while the latter denotes the control-loop relationship. We also categorize according to various classification schemes [**?**] which help us understand the data better.

6 Queries

We might just want to get a list of all sensors of a certain type that are manufactured by Seimens.

We might want to ascertain the temperature distribution on the first floor of a building, or determine the average temperature for all rooms controlled by a particular heat-pump. The former allows the user to examine how well the HVAC system is maintaining a consistent climate while the latter can be used to determine if there are any problems in the system by seeing which rooms are driving the energy consumption for that heat-pump.

We are seeing a move towards mobile sensing [**?**]. Sensors produce a stream of readings, temporally associated with locations in the building. If we have a CO2 sensors on every occupant's phone and we have wifi to determine coarse-grained association with locations in the building, then the query that determines the

CO2 distribution at a particularly location becomes non-trivial, as it requires the association history for segments of the reading-stream produced over a certain time interval. For a given time interval, we must determine all the phones that passed through a location and gather only the segment in those intervals when the phone produced readings at that location. We can narrow it further, by adding a filter by owner. So if i want to know the CO2 distribution experience by me at a particular location in the building for the coarse of a week, I would filter the list according to an ownership tag on the stream.

The queries we ask fall into three categories:

1. Graph queries
2. timeseries queries
3. metadata search

timeseries queries imply consistent association between inter-relationship state and labels – all change over time

By exposing the referential namespace, we expose the vocabulary and relationships between the objects, explicitly.

7 Implementation

When you get information about a node, the children bins are labeled by type. The type information can be fetched from the filesystem as well in `/rel`.

8 Timeseries metadata

On create, increase the version number of all entries and set the initial version for the newly created entry. On an update, increase the version for all entries. On delete, increase the version of all node except the one that is deleted.

9 Rationale

We need the ability to integrate external applications easily. Therefore we implement a POSIX-compliant FS interface.

Hierarchical naming and symbolic linking can express a directed graph. Symbolic links also enable aliasing – multiple names to refer to a object.

10 Queries: Graphical, historical, histgraphical?

The building can be represented as a collection of objects and the inter-relationship between them.

Typically the vocabulary is unknown, only general descriptions tend to be known beforehand. We expose the inter-relationships through hierarchical naming and symbolic links. We also also allow arbitrary tags on the nodes. These can be used to support general name search and queries based on general descriptions. For example, a building analyst may want to know how many occupants were the chillers in the building or to compare the relative occupancy load handled by the chillers versus the heaters, without having knowing specifically which how each chiller/heater is physically connected to each space in the building and without knowing exactly when/where each occupant was throughout the day. Another, perhaps even more important query, is one related to mobile sensors – i.e. ambient sensors on a mobile phone carried by occupants. What's the distribution of temperature for the chillers versus the heaters, as experienced by the occupants throughout the day.

Lets take an example in a different domain. We could imagine a mobile environmental sensing application that associates individuals with different locations (based on coordinate-based boundary conditions) as you walk around and passively collect environmental readings, such as sound, light, Co2, temperature, etc. A example query in this domain is "what is the average Co2 level in location 1 from time 1 to time 2?" or "what's the path with the smallest polution between location 1 and location 2 between time 1 and time 2?".

In manufactoring applications tracking context and associations over time is also necessary. They ask questions like "which batches of powder are being used to form the ceramics at the heart of the batteries, how high a temperature is being used to bake them, how much energy is required to make each battery, and even the local air pressure." [1]. In some cases, there is only a single power meter taking measurement for a multi-stage machine with other sensors at each stage. A related energy question, then, is what's the total power drawn by this machine for a specific batch of widgets – requiring a search through the database to deduce an indirect relationship between the meter and the widget over a certain time interval.

In a traditional database, these would have to be constructed iteratively. We would have to determine the equipment and sensors involved, and their inter-relationship. We would have to determine when that relationship existed and for how long and use those intervals to fetch the associated sensor readings for processing. The implication for each is that a relationship history is maintained *along with* timeseries data produced by sensors. In addition, this process implies the need for the end-user to know the names of the devices in order to query for them explicitly, since the relationship between the equipment, the sensors, and other objects is a uniquely component in the aggregate analysis.

11 References

[1] An internet for manufactoring. `http://www.technologyreview.com/news/509331/an-internet-for-manufacturing/`, 2013.

[2] Snappy. `https://code.google.com/p/snappy/`, Feb. 2013.

[3] S. Kim, S. Pakzad, D. Culler, J. Demmel, G. Fenves, S. Glaser, and M. Turon. Health monitoring of civil infrastructures using wireless sensor networks. In *Proceedings of the 6th international conference on Information processing in sensor networks*, IPSN

’07, pages 254–263, New York, NY, USA, 2007. ACM.