Dissertation Notes: StreamFS and everything that comes out of it.

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

Abstract

In a nutshell, StreamFS is an analytical framework for organizing, storing, and processing streaming sensor data. The main components of the architecture consist of a pub-sub substrate, stream operator management, and a historical metadata manager. The pub-sub substrate and operator manager are combined to provide a visual dataflow application that allows a user to construct analytical processes on incoming and historical sensor data.

Metadata management in StreamFS consists of a naming axis, a storage axis, a historical snapshot component and a query optimizer. These pieces intersect to provide stream data context for physical sensor deployments as the physical configuration evolves over time – a fundamental property physical sensor deployments.

1 Open questions

1. Are we really treating the metadata as a timeseries itself? (i.e. Can we ask the same questions of the metadata that we do of typical timeseries data?)
2. How do we perform query optimization for timeseries queries that must account for changes in the metadata over time?

2 Motivation, Problems, and Metrics

In each part of the thesis work we need a good motivation for the problem, a formal articulation of the problem that is being addressed, and how to demonstrate that the method for addressing the problem has been successful.

2.1 Setup and assumptions

For this work we assume that the data is streaming is available for streaming into the system. This assume the conversion from a data source native format to an HTTP-POST translator has been written and is being used to push the data into StreamFS or that the conversion has been done for pushing the data into the systems mentioned in the related-work section as alternatives.

2.2 Problem 1: Online analytical data processing

Related work areas: Stream data processing, streaming queries, soft real-time

Related projects:

- Dataflow programming, expression
  - Ptolemy, Click
- Stream query scalability
  - On-line Sensing Task Optimization for Shared Sensors (Arsalan Tavakoli)
  - Scalable Delivery of Stream Query Result (Yongluan Zhou)
  - Sensys disco (Prabal Dutta)
  - XSense (Jorge and Prabal – never published)
  - SEDA (Matt Welsh)
  - River (remzy or remzi?)
  - Brewer's work on threading system. (Events vs Threads, etc etc)
  - Armando (distallation - "transcend")
- General stream query processing – in relation to dynamic queries
  - Streaming Queries over Streaming Data (Sirish Chandrasekaran, Michael J. Franklin)
- Horizontally scaling the namespace and processing across machines
  - How do you run queries across namespaces?
  - is there a way to join namespaces into one larger namespace?
  - can you move computation around? (closer to where the data is)

2.2.1 Metrics

Scalability: How many simultaneous flows can we support? Processing? Delivery?

We want to be able to scale as the number of incoming flows increases, the number of processing element (threads) increases, and the number of forwarding targets increases. What mechanisms are in place to scale automatically? How much sharing can we do between dynamic queries? **Dynamic queries** are useful, but they lead to problems with scalability. What mechanisms exist to deal with scaling *dynamic queries*.

Delivery Scaling the delivery of reports to targets:

1. Get the report times to overlap
   - Scheduling algorithm for maximum overlap.
   - Constraints include strict periodicity +/- epsilon
2. Consolidate readings into JSON
3. Compress the delivery and send it to target
4. Offer local buffering
   - Design fetch protocol where buffer fetch and kept if necessary. Shared buffer only accessible by specified target (same hostname/ip) or client explicitly specified as a fetcher.
   - buffer size could be specified in request

* Take readings and do the interpolation and compare with the data quality of the result oif you don't do that. – go into that with the smap data that's available in the building

* Solve the metadata problem – it's really important - caveat – doing it with the tree/hierarchy thing might make it pretty hard - mobility under TCP problem (handoff) - INS, Internet indirection

Corner cases include:

- delivery-time difference drift (the difference between the delivery time of the next delivery for a pair of schedules changes over time – often it cycles). We need a mechanism for dealing with this. Example: [period=2, start_time=0], [period=3, start_time=1]

Expressivity: How does this compare against the traditional way of setting up the analytical pipeline? (lines of code?) What are the types of analytical tasks that we want to construct and how does it compare to the alternative?

2.3 Problem 2: Slicing, dicing, and molding data

Referred to by David Culler as producing distillates of the data. In particular, the processing model is in the context of moving data and stationary operators. For example, you write the interpolator and aggregator to run on moving/buffered windows of data.

TODO: generalize the backend fetch.

Related work areas: OLAP (online analytical processing), OLTP

2.3.1 Metrics

Query speed: In referring to query speed, we compare the query speed of running the typical OLAP queries on the historical data.

2.4 Problem 3: Metadata management

No prior work in the management of timeseries metadata.

Using the sfs data model, we need to maintain two main data structures:

1. metadata map
2. node properties

As a third component, we maintain timeseries data produced by the stream resources in the system. Queries along the data-time dimension must be merged across the mmap time-dimension and the node-properties time-dimension.

Solution outline: The general approach here is to timestamp each node, identified by a path in the mmap. There are up to two timestamps for each node. 1) the timestamp at the point of creation and 2) the timestamp at the point of deletion.

Challenges:

- Query that goes back to a particular point in time.
- Dynamic query that goes back to a particular point in time and walks forward to the current time (any kind of aggregation query that uses all the items in a particular context over time). This is especially challenging because of symbolic links.

Queries to consider are:

- **Was the database ever in a particular state?** (i.e.Did this room ever contain a laptop belonging to jorge?)
- timeseries query from now to some point in time where there were no changes in properties, but there was in naming, and data
- timeseries query from now to some point in time where there were changes in properties and data, but not naming
- timeseries query from now to some point in time where there were changes in properties, data, and naming
- query that sets now to some point in time in the past
- operators that are provided with the different types of queries:
  - filter by property type and aggregate (sum values)
  - filter by property type, interpolate, aggregate

Baseline performance should be compared with doing a standard query on timeseries data. What's the overhead for performing each of these types of queries. **In-**

time queries run immediately and should have queries latency comparable to a standard timeseries query. **Dynamic queries** run forever (until they are uninstalled) and should have query latencies reasonable consistent and close to the time that data was received from each query. Dynamic queries handle changes in naming, properties, and data. **Continuous queries** are the simpler version of dynamic queries. No changes to naming and properties has occurred. There are only changes to the data – it is continuously streaming in from a particular data source.

According to [**?**], there are four type of databases that deal with time.

1. snapshot
2. rollback
3. historical
4. temporal

Snapshot databases are databases where each operation performed on the db overwrite the previous state before the operation was committed; yielding the latest snapshot of the database. Rollback databases are collections of snapshots. To run a query at a particular point in the past is to rollback the database and run the query on the active snapshot at that point in time. Once the state is rolled forward, past snapshot states cannot be altered. The main drawback here is that **updates to past states cannot be corrected once the snapshot is produced.** Historical databases are concerned only with relations, or facts, with respect to when they were true in the real world. It is not concerned with the time when the fact was stored in the database itself. For example, Jorge has lived in Berkeley since August 21, 2005 and will move out on September 30, 2011. This is the time for which this fact is true, even if this fact was stored today, August 30, 2011. When it was stored is irrelevant. Temporal databases combine rollback and historical, combining transaction time and valid time, supporting retro/proactive queries. User-defined time should also be handled, doubling the kinds of databases.

2.5 Problem 4: Global object lookup

Object identifer with version numbers as fundamental aspect of the namespace.

3 Design Motivation

Our system is designed for a class of sensor deployments for distributed monitoring applications where the number of sensors is large and each sensor's data rate is on the order of seconds to minutes. The motivating scenario is a building monitoring system that monitors health of HVAC system components, plug-load power draw, lighting system power-draw, temperature sensors, pressure sensors, and other physical processes and items within the building. Although the individual data rates are low, the aggregate data rates per deployment can be quite large. For example, a typical 5-10 story building can easily produce at rates that match and exceed high-quality streaming video rates (i.e. 700-1200 Kbps). StreamFS uses a pub-sub architecture and generalizes the pub-sub model to support real-time stream data processing that produce derivative streams as new data sources. This can easily multiply the amount of data being stored and/or delivered subscribers.

StreamFS organizes *physical data* from *real-world physical objects* whose relationship is *as* important as the data they produce. The relationship informs our queries and motivates our decision to expose these relationships, explicitly, through naming. Although we're fundamentally building our interface on top of a relational data model, the decision to use a hierarchical namespace with symbolic links to expose the underlying entity-relationship graph is useful for managing physical data from the real-world. With the relational model you may lose important semantic information information about the real-world in return for a high degree of data independence [**?**]. Our system is more concerned with capturing entities and relationships which exist in the real world and our minds as well as information structure (organization of information in which entities and relationships are represented by data) – we use an entity-relationship data model [**?**].

The data is timeseries in nature, and since it's fundamentally related to readings taken in physical space, the associated metadata must be tightly coupled with each data stream. The metadata describes the units, calibration parameters, placement and other important stream-context information. Metadata should be treated as a first-class citizen in the system. Such treatment is typical in fields in the natural sciences, such as environmental and climate science, where the metadata sets the framework for in. Strict guidelines are given for recording and managing metadata [2]. For the class of applications we are designing for, such concerns are just as relevant and metadata management must be done with great care. For example, temperature measurements taken in various locations – a room, a hot or cold water pipe, an air vent – and in order to interpret the measurement, we need the units of measurement, the location of the sensor, and calibration parameters (if we are getting raw readings that need to be converted to measurement values).

In addition, we must track the deployment as it evolves. Changes in metadata change the interpretation-context of the associated data stream. For example, as a sensor gets older it goes through natural wear and tear which change the calibration coefficients. When such coefficients are updated, the associated data must be coupled with the newly reported data. Also, some sensors are mobile and move from location to location. We have designed various mechanisms for handling deployment evolution and discuss these in section 3.3.

3.1 Location information

We wish to record and keep track of the placement of sensors throughout a space. One method for recording spatial information is to include geo-spatial coordinates

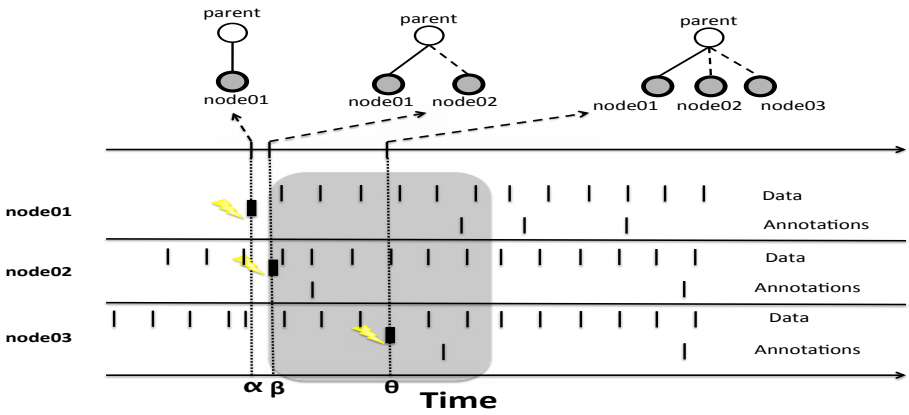

Figure 1.

for each sensor. However, geospatial coordinates do not capture more relevant information about what is being measured. In a building, placement is more abstractly defined as the system or space in which the sensor is placed, not the specific coordinates of placement. Although a coordinate system could also be useful, system or space association is more relevant in interpreting sensor readings.

In addition, geospatial coordinates are difficult to assign or determine in some locations – particularly indoors due to weak signals, fading, and multipath [**?**]. One could argue for the use of other infrastructure for determining coordinates in such a setting, however, the challenges remain [**?**, **?**, **?**]. Still, referring to sensors based on placement (with semantics ascribed by the user) is more important than determining the exact location. Our system takes the approach of describing placement through naming. Our naming mechanism is simple and flexible enough to allow for arbitrary names with ascribing meaning to the naming convention. We leave interpretation semantics to the application.

3.2 Naming

Names should be human readable and interpretable, similar to DNS. In buildings, sensors are referred to by the system or space associated with the point of measurement. Our approach was to use a hierarchical naming scheme, similar to a Unix filesystem. As explained in [**?**], hierarchy restricts the user to retrieve their data according to how that piece of data is named. The full pathname conflates naming and access. Although this can be limiting for data in a traditional sense, this restriction serves a very useful role in naming and retrieving data for the building application example and deployments like it. We illustrate this with an example in building sensor deployments.

Suppose there's a sensor on the first floor in room 100 in Soda Hall. If we have different deployments, we can set the hierarchical structure to have a path as follows '`/buildings/soda/01/100/tempSensor`' that walks us down from the '`/buildings`' which can represent that set of buildings in our multi-deployment all the way from the building to the floor and finally the room where the sensor is placed. It's natural to organize the data this way and useful from a human-readable perspective to immediate extract the context where the sensor is based entirely on the access-path for that sensor. This naming convention does not entirely cover all medata but it simplifies access to relevant information.

The experienced reader may have noticed in a building a sensor that drives a system may also sit in a space. A sensor can be referred to through it's association with the system or its association with a space. This observation implies the need for supporting multiple names. We support multiple names through the use of symbolic links, similar to symlinks used in typical filesystems. For example, if the `tempSensor` is attached to a variable air volume (VAV) component in the heating, ventilation and air conditioning (HVAC) system; in other words, the temperature sensor drives the heating/cooling sub-system for that room. Another name for the same sensor is '`/buildings/hvac/ventilation/vav/tempSensor`'. To assure that both names refer to the same object we differentiate between *hardlinks* and *symlinks* and either we make one of the two paths point to the hardlink or make both names symlinks and create a hardlink with a different name that both paths point to. In the mobile context, the latter approach is useful. The object reference remains static while the symlink references to the object change as the object moves from place to place.

3.2.1 Additional metadata

Naming does not capture all the metadata, so we included object/node annotations: user-defined properties that are attached to each object. We support search by property values. Geospatial, type, or other information can included in the annotation for the object. Annotations are flexible enough to allow for a wide range of application semantics. For example, our building monitoring application defines a set of node *types* to represent different components in the HVAC sub-systems, electrical load tree, and zones inside the building. More details can be found at [**?**].

3.3 Time and deployment evolution

Long-term deployments naturally evolve over time. There are changes to the deployment settings and changes to sensors. Sensors are replaced, removed, and added. The space being monitored expands or morphs. Meter upgrades join new facts about the sensor(s) with the sensor that replaced it. As a concrete example, lets return to the building sensor deployment setting. Buildings are often up for decades. During this time period, rooms are added and broken sensors are replaced. If the temperature sensor in a room breaks it needs to be fixed or replaced. Both involve certain new information to be recorded about the deployment, such as updates calibration parameters. Although the logical sensor-access name is unchanged, the physical object the name references has changed and such changes are important.

At a more infrequent rate, there may be changes to the physical structure of the building. Two rooms may be combined into one. An extra floor may be added to accommodate more people or new building functions. Such changes usually require changes to the underlying climate and electrical systems. New ducts and vents are added into the new spaces, expanding the reach of the current HVAC system. A new, independent HVAC system may be added altogether. The electrical load tree must accommodate the new load, which requires additions to the underlying structure of the electrical load tree.

These are impact the context in which measurements are being collected. Without mechanisms to accurately track and account for such changes, analysis about the behavior and consumption with respect to stale information will lead to gross miscalculations that get worse and worse over time.

3.3.1 Timeseries data, naming, metadata

The data collected from deployments is fundamentally timeseries in nature. However, physical data cannot be interpreted without its associated context. Such context information is recorded in the metadata for each sensor stream. As described in the previous sections, we partition the metadata into two pieces. The first piece captures placement information through naming while the second piece captures everything else. In particular, the latter piece should be used to record calibration information for the readings being collected. The data and associated metadata are tightly coupled and we must design mechanisms to maintain the integrity of this relationship through time. Changes in placement, as recorded by the naming structure, are hereafter referred to as placement context while change in calibration or other descriptive information is hereafter referred to as descriptive or measurement context (both will be used interchangeably).

Timeseries data, placement context, and measurement context are bound through time. For example, imagine a temperature sensor in a room sending periodic temperature readings. Upon installation, the temperature sensor is named with respect to its placement in the room, (i.e. `/room/tempsensor01`). With the newly created reference we record calibration information associated with the reference. At some point in the future, the temperature sensor is replaced. We wish to keep the logical reference to the sensor, so we keep the name reference to same, however, the new sensor has different calibration parameter that override the old ones. In calculating, the average temperature of the room over that timespan, we need to re-structure the placement and measurement context in order to make an accurate assessment of the average temperature. Using latest calibration parameters would give false readings for the original sensor. Moreover, if at some point later, we install a completely new temperature sensor model and delete the current reference altogether, we still want to "remember" that we had the old model in the past in order to calculate the average temperature with the old readings.

Both scenarios require the coupling of separate timelines. Attached to each set of readings is the metadata that describes the context of the data. The metadata in StreamFS consists of the path(s) and properties associated with the object. The state of this set uniquely identifies an object and any changes to the set of path(s) or properties generates a new object identifier. The object identifier (*oid*) is unique 128-bit number. The high-order 96 bits are unique to the object, while the remaining low-order 32 bits are the version number. When a change occurs to the metadata we increment the version portion of the identifier. We also record the time when the change was made. As the data is stored, the oid is attached to each inputted value. This allows us to explicitly couple the metadata with the data. The timestamps are used for snapshot and rollback queries.

In section 4 we discuss the different types of MTSQ's, their relationship to traditional mechanisms used in temporal databases, and the complexity and performance issues for storing timelines and executing TLS's for MTSQ's efficiently. Dealing with MTSQ's over time intervals with many changes is the main challenge in running fast, efficient queries. We have implemented a set of algorithms and caching technique that allow MTSQ's to run with very little overhead, when compared to a tranditional timeseries query. The average overhead is only `X`% in the average case and no worse that `Y`% when compared to the standard timeseries query performance. In addition, we show how MTSQ's help simplify the tracking and subsequent accuracy of derivative historical calculations.

3.4 Derivative streams

3.5 Mapping relationships explicitly

4 Metadata timeseries queries

whataksndlkasndlkasn

4.1 Objects and data

4.2 Structural snapshot

4.2.1 Storage

4.2.2 Querying

5 Relationship to OLAP

The main terms in OLAP consist of Dimensions, Measures, Hierarchies, and Grain. Dimensions are the axes of aggregation. For example, you may want to aggregate with respect to location, time, or type. Measures are the units or category of measurement that you're making for the data values assocaited with a dimension. For example, cost is a type of measure, so is revenus and quatity. There are hierarchies in the data that dictate the relationship between the dimensions and how that relationship influences the amount of derived data and computation that needs to be done to satisfy these hierarchical aggregations. The grain of the data is the lowest level of granularity. Abstractly, the lowest grain in OLAP is the actual transaction. In the context of StreamFS, it's the raw stream coming in from a stream source.

In a typical OLAP setup, hierarchies do not change, dimenions do not change, grain does not change, measures do not change. OLAP is perfect for industries with structured analytical accounting such as finance and accounting but less of a fit for sales, operations, marketing, and R&D.

What's different here is that the dimensions change – measures have multiple coordinates at any point in time and the coordinates change as a function of time, not just measures themselves. Dimensional coordinates are expressed by tags, where the time dimension, exists both for measures produced by stream objects and the tags themselves. The tags are used to reconstructed the relational-DAG between the dimensions and the objects at any point in time.

- Explain how are the tags used to handle dynamic dimensionality.
- Explain how the tags are used to handle hierarchical relationships.
- Explain how tags are used to support multi-dimenionality (multi-naming).
- Explain how OLAP operations are performed using these naming approach.
  - Rollup.
  - Drill-down.
  - Pivoting.
  - Slice and Dice.
  - time.
- Explain how the indexing is done to support these operations efficiently – use each of the operations for demonstration.
- Explain how timeseries queries account for dimensionality and hierarchical changes.

StreamFS is an analytical framework that provides streaming OLAP for operational processes with schema timeline consistency.

- streaming olap
- changing dimensions
- metadata timeline consistency management

6 Mobile SFS

This work is fundamentally motivated by the premise that personal energy-use transparency leads to more efficient management of your personal energy consumption.

The phone is the natural point of intersection between the user, the phyiscal world, and virtual services.

Goals:

- Personalized energy attribution and tracking

Accounts:

- spaces
- users

Users use their mobile phone to tag items they are actively using.

Tagging and registration mechanisms:

- scan the qr code of the item you are currently using
- if the item is a shared item, then it can be tagged by multiple users, and the energy can be split amongst all the users
  - if a non-personal item is actively being used and nobody claims it, it is charged to all the users that we were in the same space as the item when the item became active.
  - if a non-personal item is actively being used and nobody claims it and nobody was in the space when the item was turned on, it is charged to the space.
- if the item is a single-use item, it can only be used by a single user at time
- if the item is a personal item, it can only belong to a single user and all energy is charged to that single user

Hypotheses:

1. Personal energy attribution helps induce changes in behavior that lower energy consumption.
2. Personal energy attribution requires active user participation and engagement.
   - Localization can be automated with indoor wifi; lowering participation overhead.

6.1 Tracking steps

1. Bootstrap

2. Tracking
3. Analysis
4. Visualization

6.1.1 Bootstrap

The bootstrap phase is step where users become aware that there is energy information out there and it's the essential bootstrapping processes of getting the user invovled in their energy attribution. This step consists of:

1. Tagging items of everyday use with qr codes.
2. Binding meters and items.
3. Tag items as personal, shared, n-multi-shared

All items should be tagged with qr codes. This involves physically sticking a qr code onto the item, scanning it, and describing the item that it is attached to. This should be done for **all items with a measurable energy footprint**, including lights, computers, appliances, chargers, space heaters, etc. – miscenallenous eletrical loads. A more advanced setup could include eletrical load information or HVAC component tagging, if visibility of total energy footprint is of interest.

Binding meters and items is a way of informing the system that the meter serves as an energy-consumption proxy for the item that the meter is attached to. This involves scanning both the meter and the item and creating their relationship explicitly. This allows the application to query the appropriate meter for historical consumption information when the user wishes to learn about the consumption of the item that is attached to that meter. This model also assumes that meters are essentially free with respect to energy consumption. Generally this assumption is not that far from reality. For example, the ACme [1] plugload meter draws on the order of several hundred milliwatts of power versus the tens to hundreds of watts being consumed by the items it is typically measuring.

Tagging is the final, important step in the engagement phase. This step requires that user scan items and essentially mark them as personal or shared. This is used for analytical purposes and attribution. You cannot determine which account the energy consumption of an item should be debited to without ownership information. Our application distinguishes between three types of sharing. Personal items means that there is an owner relationship between the item and its owner. Any energy consumed by the item is attributed to the owner. Shared items are charged to the user that last made use of the item. If no user takes ownership of the item it is charged to every user of the system and each user is actively informed of the added charge (discussed in section **??**). Finally, the n-multi-shared items are items that can only be shared by N users at any one time. This feature is essentially sets an upperbound on the number of accounts that can be chared for the energy consumption of the device.

The cost of engagement is intially high but reduces over time. The feature that we like best here is that the infrastructure captured through the engagement step is beneficial for all users in the system and can be constructed iteratively. The more users you have engaged, the faster the physical infrastructure can be virtualized.

6.1.2 Tracking

Tracking is the most active step in the attribution cycle. This is where the user uses the application to record their location and energy consumption information. The infrastructure constructed in the engagement step allows the user to scan their location and item being used. For example, when the user walks into a building, she scans the qr code for the building to set their context. As the user enters their office, they scan the qr code for the office. Finally, before using any electrical device, the user scans it to register it's use. Personal tags are most useful here, as they reduce this scanning step. When the user is done using their item, they scan the item again to record that they are no longer using it.

This phase in the processes is the most tedious for the user and we think that more infrastructure and learning can be done to improve tracking. To minimize interaction we offer mechanisms for grouping usage patterns. For example, the user can decide to group items into an activity and combine streams for items that are associated with that activity. For instance, each morning a graduate student comes to lab at the same time, turns on their PC or laptop, lamp, and prints out some papers. The student pay group these streams into a single activity which alerts the application to bin the energy consumed by the associated device to the account belonging to the user.

We also run correlation analysis in the background that looks to associate patterns of usage. These may be presented to the user for corroboration and ease the grouping processes. We discuss its use in section **??**.

6.1.3 Analysis

Using traces generated by user activity, we track their energy usage through detailed accounting. Our application combines energy consumption data with contextual streaming data (i.e. location, usage information). We perform aggregation in time, space, and category per user and group by activity and location. We also looks for trends in the data that indicate correlated usage patterns between items and users. This allows us to ask the user specific questions about grouping to make out analytics more accurate and reduce the burden imposed on the user during the tracking phase. It also enables us to present the data to users about correlations amongst one another that could help them plan to use items more efficiently from an energy perspective. Details on our analysis is presented in **??**.

6.1.4 Visualization

Finally, we looks for interesting ways of presenting real-time analytical data to users that could help them understand their own energy usage, as well as the con-

text of energy usage in their environment. Our specific aim in this phase is to induce changes in behavior that cause the user to make better use of their devices and reduce their overall energy consumption. We demonstrate some of the visualization in section **??** and the overall energy consumption before and after using the application in section **??**.

7 Results

This section will go start, first by talking about the overall energy consumption results. For this, a baseline must be established, either at the room-aggregate level or the individual user level. The latter is simpler and is useful for comparing the effect of the system on invidual energy consumption. That's really the point of the system.

8 References

[1] X. Jiang, M. V. Ly, J. Taneja, P. Dutta, and D. Culler. Experiences with a high-fidelity wireless building energy auditing network.

[2] *Metadata and Provenance Management, Ewa Deelman, G. Bruce Berriman, Ann Chervenak, Oscar Corcho, Paul Groth, Luc Moreau, Chapter 12: Scientific Data Management: Challenges, Technology, and Deployment*. Chapman and Hall/CRC, Taylor and Francis Group, 2010.