StreamFS: A system for energy-data OLAP in buildings

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

Abstract

1 Introduction

Buildings consume 40% of the energy produced in the United States and nearly three quarters of the electricity produced [7]. It is conjectured that as much as 30-80% of their energy is wasted [3, 4]. With such astounding figures, buildings are a clear optimization target. In order to optimize building performance, we need fine grained visibility into the energy flow throughout the building. With over 70% of commercial buildings, 100,000 square feet or larger, having a building management system [1], the infrastructure to attain that level of visibility is available and widespread.

Building management systems (BMS) consist of thousands of sensors deployed throughout the entire building, measuring temperature, pressure, flow, and other physical phenomenon. They collect data from each of these sensors and present them to the building manager to show the current state of building and the system components within it. Despite this level of penetration and data availablity, energy-flow visibility is poor. This is mainly due to two reasons:

1. Building management system are designed for monitoring, not analytics.
2. Deep analysis requires careful meta/data management and processing to clean the sensor data.

In this paper we describe a system we have designed to address these short-comings. StreamFS is an analytical framework that uses an entity-relationship graph (ERG) to model the logical and physical objects and inter-relationships that inform the questions posed to the system about energy-flow. StreamFS provides traditional *slice and dice*, *drill-down*, and *roll-up* OLAP [2] operations; which come naturally from the graphical structure. It also informs our naming scheme, which we use to allow the user to see and manipulate the state of graph to capture the underlying physical and logical relationships and consequently update aggregate data calculations upstream.

Our naming scheme is hierarchically structured, like traditional filesystem naming, with support for symbolic links, allowing arbitrary links between sub-trees. We argue that this naming scheme is crucial, as it exposes the inter-relationships which inform aggregation semantics intended by the user. StreamFS distinguishes between nodes that represent streaming data sources in the real-world and those that do not. Those that do not, however, can be tagged as aggregation points. As part of the tagging processes, a user specifies the units of aggregation, with additional options for cleaning and processing.

We use StreamFS to organize and support applications using building data from three different buildings. The first one is a 110,000 square foot, seven-story building, the second one is an eleven-story 250,000 square-foot building, and the third is a 150,000 square-foot eleven-story building. Our contributions are:

- A naming scheme for physical objects and inter-relationship that is used to construct an entity-relationship graph.
- Use of the entity-relationship graph to provide OLAP *roll-up*, *drill-down*, and *slice and dice* operations.
- Show how sliding-window operations can be used on real-time data in combination with the entity-relationship graph to maintain accurate aggregates as the underlying objects and inter-relationships change.

We also discuss how we deal with the fundamental challenges that come with sensor data. Specifically, we address *re-sampling* and *processing models*. The incoming data does not have a common time source, so combining the signals meaningfully involves interpolation. There are various options that we provide for performing the interpolation, chosen by the user depending on the units of the data. For example, temperature data may involve fitting a heat model with the data to attain missing values in time. In addition, aggregation is done as a function of the underlying constituents: they can be combined arbitrarily, by adding subtracting, multiplying or dividing corresponding values. We provide an interface to the user that allows them to specify how to combine the aggregate signals as a function of the child nodes in the entity-graph. Futhermore, they can filter the data by unit. This kind of flexibility useful for visualizing energy consumption over time.

1.1 Entity-relationship model

Tracking the operational energy consumption of a building requires the ability answer a series of questions about energy flow – energy data aggregated across multiple logical classes to determine how, where, and how much is being used. Sometimes, it even involves extrapolating forward in time to estimate future consumption patterns that could influence immedaite decisions. The ability to *slice and dice* the data allows the analyst to gain better insight into how the energy is being used, where it can be used more effectively, and how to change the operation of the building – through better equipment or activity scheduling – in order to optimize and reduce its energy consumption. Below is a typical list of questions:

1. How much energy is consumed in this room/floor/building? On average?
2. What is the current power draw by this pump? cooling tower? heating sub-system? Over the last month?
3. How much power is this device currently drawing? Over the last hour?
4. How much energy have I consumed today? Versus yesterday?
5. How much energy does the computing equipment in this building consume?

Notice, these question span spatial, temporal, and other arbitrary aggregates – some physical, some categorical. There is also an implicit hierarchical aspect to the grouping, in some cases. For example, there are many rooms on a floor and many floors in a building. Naturally, to answer the first question we can aggregate the data from the room up to the whole building. This hierarchical relationship is not as evident in the HVAC sub-components specified in the second question. However, local hierarchically relationships *do exist*. For example, the cooling system consists of the set of pumps, cooling towers, and condensers in the HVAC system that push condensor fluid and water to remove heat from spaces in the building.

We can model this as a set of objects and inter-relationships which inform how to *drill-down*, *roll-up*, and *slice and dice* the data – traditional OLAP operations. The main difference between this setup and traditional OLAP is the underlying dynamics of the inter-relationships: objects, particularly those meant to represent physical entities, are added and removed and their inter-relationships change over time. *The natural evolution of buildings and activities within them makes tracking energy-flow fundamentally challenging*.

In this paper, we show how the entity-relationship model [6] helps simplify this problem, both as an interface to the user and a data structure for the aggregation processes. We argue that the use of this model is a cleaner fit for this application scenario because it captures important semantic information about the real-world; facts critical for picking which questions to ask and how to answer them. In contrast, it has been shown that a relational model loses this information [5].

1.2 An example

Lets examine the requirements for answering the first question. A building is unware that there are rooms. Typically spaces in a building are called *zones* and, at construction time, walls are added to make rooms within zones. This makes rooms an abstract entity, used to group associated items with respect to it. It also means we typically do not have a single meter that is measuring the energy of a room; it must be calculated from the set of energy-consuming constituents.

What are the energy consuming constituents of a typical room? It is the set of energy-consumers that are active within or onto the room. Broadly, it consists of three things:

- Plug-loads
- Lights
- HVAC

For simplicity of demonstration, lets consider only plug-loads. In our construction of an entity-relationship graph lets assume there are nodes for each plug-load item and each room. For the room in question, the relationship between the plug-loads and the room is child to parent, respectively. The total energy consumed by the plug-loads can be aggregated at the parent node, the room, so the user can query the room for the total. Over time, plug-loads are removed and added to/from the room, but the relationship does not change. This simplifies the query; to obtain the total consumption over time, the query need only go to the room node. The parent-child relationship informs which constituents to aggregate over time to calculate the total.

1.3 General Approach

To realize this design we need to maintain the entity-relationship graph, present it to the user in a meaningful way; allowing them to update it directly to capture physical state and relationship changes. We also need to use this graphical structure to direct data flow throughout the underlying network. This allows us to accurately maintain the running aggregates as the deployment and activities churn.

We present the graph to the user through a filesystem-like naming and linking mechanisms. The combination of a hierarhical naming scheme and support for symbolic links allows the user to access and manipualte underlying objects and relationships. Moreover, the underlying graph structure is overloaded with upstream communication mechanisms and buffering to allow data to flow from the data-producing leave nodes to the aggregation-performing parent nodes. Furthermore, the buffering lets us deal with the streaming nature of data flow from the physical world to StreamFS and lets us maintain a real-time view of energy flow in the system. Traversing the graph provides a natural way for the user to implicitly execute the OLAP operations necessary to give the user the kind of insight into energy usage in the building necessary to understand, optimize and reduce it.

2 Mapping OLAP to ERG

- Introduction to OLAP.
- Explanation of ERG in StreamFS.

Online analytical processing (OLAP) is a processing layer that provides summurization of data from a set of underlying data repository (date warehouses). Traditionally, OLAP is used to process business data. Business data summurization allows an analyst ask targetted questions about aggregates and trends in their data. The data is typically multidimensional in nature and operations can be performed with respect to those dimensions and their inter-relationship.

2.1 Measures, Dimenions, and Levels

2.2 Operations: drill-down and roll-up

2.3 Operations: slice and dice

2.4 Operations: pivoting

3 Related Work

4 Namespaces

StreamFS manages two namespaces. The first is a flat namespaces that identifies a particular object instance. The second is a hiearchical namespace that identifies the current instance of a particular object. In this section, we discuss why we have two namespaces, how they are managed, and how they are used by the query interface and analytical layer.

4.1 Object identifier namespace

Each new object that's created is assigned a 128-bit unique identifier:

- 96 high-order bits identify the object
- 32 low-order bits identify the version of the object

In this paper we concentrate on the high-order bits used to identify the object. Management of the low-order bits is the subject of related, ongoing work.

4.2 Hierarchical namespace

5 Dynamic Aggregation

Dynamic aggregation combines the underlying entity-relationship graph with in-network aggregation. It treats each node in the graph as a potential point of aggregation on a particular data type. For example, if we need to compute aggregates of *KW* data and we declare the node for a particular room as the point of aggregation, we accept data from all children of that node that, whose units are in *KW*, and add the streams together over pre-defined window size or pre-defined timeouts.

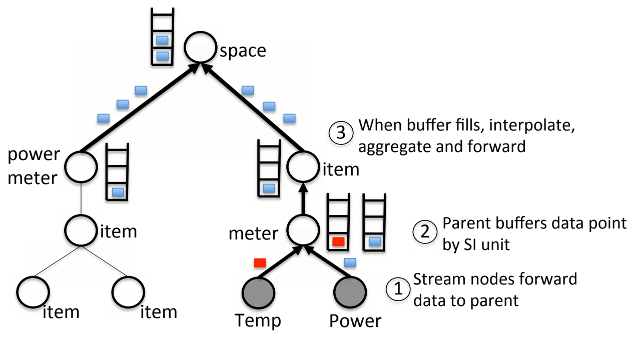
The scheme is hiearchical, so a node only accepts data from its children and only sends data to its parent. StreamFS checks for cycles when before node insertion and prevents double-counting errors by only allowing aggregation-points that are roots of a tree that is a subgraph of the entity-relationship graph. In our deployment, each view is a managed as an independent hierarchy. So the hierarchy of *spaces* is separate from the *inventory* hierarchy or the *taxonomy* hierarchy. This allows us to ask questions with a particular view in mind, without conflict, and is a natural fit for our aggregation scheme.

5.1 How it works

Although there are different semantics applied to different node types at the application layer, StreamFS only knows about two types of nodes: (1) default nodes and (2) stream nodes. The main difference is that *default* nodes are not explicitly associated with data coming from a sensor and *stream* nodes are. Furthermore, default nodes can have children, while stream nodes cannot. In our application, meters are represented by default nodes and each stream of data they produce is a stream node.

When an aggregation point is chosen and enabled, dynamic aggregation places a buffer at the node for the type of data that should be aggregated. If we want to aggregate *KW* data, we specify the type and send an enable-aggregation request to the node through an HTTP `POST` to the path for that node. The flow of data starts at the leaves when a stream node received data from a sensor through HTTP `POST`. As data arrives it is immediately forwarded upstream to the parent(s). If a node that receives data from its children is an aggregation point it buffers the data, otherwise it fowards it to its parent.

Ignoring the timeouts for now, lets imagine the parent is a point of aggregation and its buffer is full. At this point the parent separates data into bins for each source and cleans it for aggregation through interpolation. The main operation is to *stretch* and *fill* that data with linearly interpolated values. The *stretch* operation orders all the timestamps in increasing order and for each bin (signal) interpolates the values using the first (last) pair of data points. If there is only a single data point, the stretch uses it as the missing value. The *fill* operation find the nearest timestamps that are less-than and greater-than the missing sampling time, uses their values to determine the equation of a line between them and interpolates the missing value using that equation. Once this is done for each signal, the values are added together for each unqiue timestamp and the aggregated signal is reported to the parent, where the operation occurs recursively to the root. Figure 1 shows an illustration of its operation.



space
power meter
item
item
meter
item
item
Temp
Power
3 When buffer fills, interpolate, aggregate and forward
2 Parent buffers data point by SI unit
1 Stream nodes forward data to parent

Figure 1. This shows an illustration of the aggregation tree used by *dynamic aggregation*. Data flows from the leaves to the root through user-specified aggregation points. When the local buffer is full the streams are separated by source, interpolated, and summed. The aggregated signal is foward up the tree.

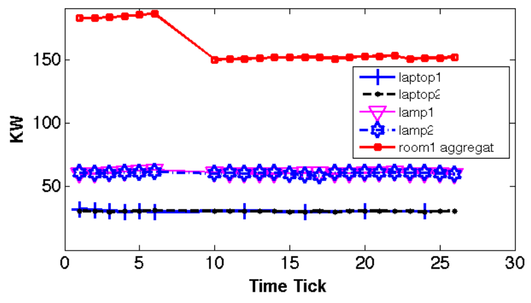
5.1.1 *Dealing with dynamics*

This approach deals with changes in the graph quite naturally. All aggregation point deal only with local data, so a node is only concerned about the children that give it data and the parent to send data to. As objects in the environment move from place to place and these changes are captured, the entity-relationship graph also changes to reflect the move. This change in aggregation constituents is naturally accounted for in the aggregate. If a child is removed, it no longer forwards data to the old parent, therefore the aggregate will reflect that change. Note, however, that changes in the entity-relationship graph are indistinguishable from energy-consuming items that have been turned off. For the purposes of aggregation, that is okay.

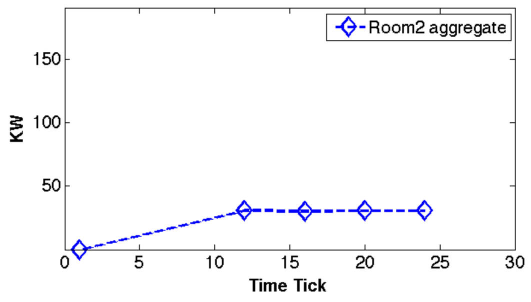
5.1.2 *Two scenarios*

We illustrate dynamic aggregation with a common usage scenario. Imagine there are a number of people in a building, each owning a number of plug-load applicances and a laptop. Assume that when a person is in a room their laptop is plugged in and when they leave the room they unplug their laptop and take it with them. People come and go throughout the day, changing the aggregate power consumption of the room and it happens. In addition, some of those people move to other rooms and plug their laptop in the new location. As this happens, we will assume all actions are being recorded in StreamFS.

Figure 2 illustrate the aggregation results of that scenario. Notice how...



(a) Room 1 object and aggregate streams.



(b) Room 2 aggregate.

Figure 2. The power consumes by a laptop in *room 1* is shifted to *room 2* a time t=7. Notice the aggregagate drops in room 1 while it rises in room 2.

6 References

[1] Commercial buildings energy consumption survey, 2003.

[2] E. F. Codd, S. Codd, and C. Salley. Providing olap to user-analysts: An it mandate.

[3] N. Gershenfeld, S. Samouhos, and B. Nordman. Intelligent infrastructure for energy efficiency. *Science*, 327(5969):3, 2010.

[4] Next10. Untapped Potential of Commericial Buildings: Energy Use and Emissions, 2010.

[5] M. E. Senko, E. B. Altman, M. M. Astrahan, and P. L. Fehder. Data structures and accessing in database systems: Iii data representations and the data independent accessing model. *IBM Syst. J.*, 12:64–93, March 1973.

[6] P. P. shan Chen. The entity-relationship model: Toward a unified view of data. *ACM Transactions on Database Systems*, 1:9–36, 1976.

[7] U.S. Environmental Protection Agency. Buildings Energy Data Book, 2010.