StreamFS

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

1 Physical-data applications

Physical-data applications are applications built to provide insight about the physical world. They range from analytics and visualization to control and actuation. Analytics and visualization apps typically provide aggregates, statistical summaries, and future predictions about the behavior of the physical phenomenon being measured. The fundamental challenges for these application are dealing with flawed or missing sensor data, integrating smoothing and processing models to fill in gaps in the data, and maintaining consistency between the physical world and the representation of it within the processing system, particular as it evolves. *Context maintainence* is a fundamental challenge for correct interpretation of the physical data.

Control applications allow users to control their environment. Control applications are either built off analytics, providing intelligent, model-based control or they can provide simple, local, direct actuation by users through item or context-specific user interfaces.

Security is also a major concern in physical-data applications. Physical data reveals personal information about users through their affect on physical measurements. For example, light sensors and location information indicates occupation. Energy-consumption aggregates, when grouped by user or localtion can give information about the energy consumption habbits the occupants or users of those spaces.

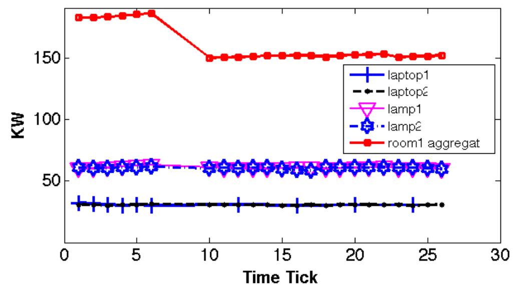

Figure 1.

1.1 Data organization, data services

1.2 Context consistency maintenance

1.3 Dashboards

1.4 Analytics

1.5 Control

2 Subscription semantics

A subscription expresses the intent to receive all associated streams whose topic is expressed by its hardlink name or any associated alias. Fundamentally, the subscriber expresses an interest in specific streams, referred to directly by name or indirectly by topic matching as it pertains to regular-expression name matching.

2.1 Continuous group-by for aggregation

Many of the application queries require aggregation of data over groups of streams. However, membership to those groups change over time, so aggregate recalculation must account for the changing constituents in order to maintain accuracy. Because of the way subscriptions are implemented in StreamFS, this is handled automatically. Recall, only streams whose tags match a subscriber's regular expression is forwarded to the subscription target. By removing a tag, you change group membership. Tag removal *can* change the membership of a stream to a group of streams defined by the regular expression. The data from streams belonging to the group are no longer accounted for in the aggregate calculation. The same is true when new members join the group. Aggregate calculation accuracy is maitained in either case.

Lets go through an example.

3 Stream processing

StreamFS allows users to perform stream processing on collections of streams.

Preserving timing semantics is challenging for two reasons:

1. NodeJs is single threaded; although it's an implementation issues, it's clear that the scheduler can only handle one job at a time, so threading is necessary
2. load is a metric of cpu activity, but the real metric we care about is expected versus actual start and completion time of a job.

The second point is the important factor that determines when a job gets moved to another server. We aim to maximize CPU utilization on a single server, without sacrificing timing constraints specified by the user. Job execution timing quality is measured by the absolute deviation of the completion time of a job from the scheduled completion time, that quality measure degrades as computing cycles become more scarce. In other words, the less cpu cycles there are to schedule the CPU-bound job, the larger the average error and variance of job completion times.

Now I have to show this experimentally and describe the methodology and job scheduling/migration algorithm.

3.1 Job scheduling and migration

Lets assume we have a job J that takes T_c to complete is scheduled to run every T_p seconds.

3.2 Experimental results

4 Filesystem for streaming data

A file system provides an abstraction for managing a collection of bytes on disk. The standard file itself provides logically sequential access to bytes distributed through the disk. Directories provides a simple container mechanism for grouping together files that are in some way related to each other. We think deployment metadata and data can similarly benefit from a simple hiearchical naming structure. Grouping items that share overlapping name-elements separated by forward-slashes in their tags. It provides a simple way to specify groups of streams as the unit of access or processing. File systems have also developed a convenient set of tools for quickly locating information, sharing reading, and pipelining processing; simple tools that fit the access, sharing, and processing needs for local-area sensor deployments. Filesystem also provide a security model baked into their construction. We examine the filesystem interface to StreamFS and examine how well the interface fits the needs for the class of applications StreamFS supports.

4.1 Naming and access

For local-area deployments, where spatial or categorical referral patterns are common in the tagging structure, hierarchical grouping provides a good, simple fit for naming objects. However, we separate logical grouping from physical, and by doing so, also separate naming and access. This prevents overly-stringent access patterns that may degrades performance over time. How the data is accessed is independent of how the object named. It also support multiple names without affecting without affecting access.

Tags are abitrary, but forward-slash separated naming conventions map cleanly into a filesystem naming. Each string between the forward slashes is treated like a directory in the filesystem. If a user attempts to create a file with an element in the name that is already set for a non-container object in StreamFS, then creation of the file is not allowed.

4.2 File types: directories, regular, special, pipes

4.3 Filesystem tools: grep, find, ls

5 Publish/subscribe model

Eugster et al. [1].

StreamFS uses a flexible flavor of the publish/subscribe model in order to support a wide range of applications. Publish/subscribe is necessary is physical data application development in order to scale and support many diverse applications over potentially thousands of raw and derived data streams.

The publish/subscribe model used in StreamFS provides mechanisms that enable a flexible combination of space, time, and syncrhonization decoupling. This enable StreamFS to support of a wide arrange of application requirements.

Our pub/sub engine is also tightly coupled with the namespaces expose to users, and this design choice allows an application to control the space coupling between the publisher and the subscriber.

Raw timeseries data is always stored while the user can choose to store derived data. Storage allows for synchronization decoupling, since the publisher and consumer may produce/consume data indepdently of one another.

5.1 Space decoupling

5.2 Time decoupling

5.3 Synchronization decoupling

6 References

[1] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec. The many faces of publish/subscribe. *ACM Comput. Surv.*, 35(2):114–131, June 2003.