Stream-lined Data Management

Stan Zdonik
Brown University

Preamble

Why all the sudden interest in data streams? In our view, a stream is simply an append-only collection of tuples that is ordered by some increasing key value (often time). The notion of append-only relations is not that startling. Is this just the latest trendy topic or is there something more fundamental going on?

Data streams are interesting primarily because of the ways in which they arise and the ways in which they are used. Unlike other recent "hot topics" in which novel *structure* in the data (e.g., XML, design data, human genome) drove the interest, the excitement here lies in the unusual processing requirements that streams and stream-based applications bring to the table. The focus is not on new data models, but rather on new system architectures, non-traditional optimizations, and algorithms.

Whenever the following conditions hold, it can be better to push data toward clients than to store it and wait for a pull request.

a. High request load could swamp the server.
b. Highly volatile data presents a data freshening problem (encourages polling).
c. The storage potential of the server is severely limited.
d. The server is "far away" from the clients and latency really matters.

Push-based systems [3] naturally create data streams. Data streams can be captured in a repository for later processing or they can be processed on-the-fly. Storing streams places stress on the capabilities of a standard DBMS because queries often must take into account the natural order of the stream elements. For example, find all the "peaks" in a time series. Many modern applications must process streams on-the-fly because they are interested in monitoring the current condition and reacting to that condition as soon as possible.

What does this have to do with pervasive data management? Since pervasive computing environments will include many, small embedded devices, most of these devices have one or more of the characteristics listed above (consider sensors). These devices often push data into the world hoping that someone upstream will be able to use it. Thus, managing push-based data delivery becomes a requirement in this setting.

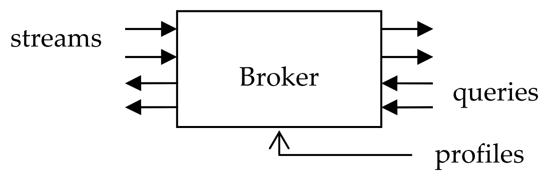
Of course, all data will not be pushed. Thus, research is needed to understand how best to seamlessly integrate push-based and pull-based data delivery schemes. What data is made available for future pulls? What data is immediately pushed? How are **both** kinds of data organized?

Research Themes

In this section, we will sketch some topics that are important to a successful pervasive data management infrastructure. Most of these topics can be linked to the existence of large numbers of push-based sources (stream producers) and to the fact that the combined data rates for these sources might be quite high.

Architectures

What components are best suited to the construction of stream processing systems? A dissemination-based information system [1] (DBIS) disseminates data from multiple push-based sources to interested users. A DBIS is constructed out of data sources, information brokers, and clients. The brokers take input (pushed or pulled) from data sources or other brokers and perform data management functions such as content-based routing, caching, aggregation, filtering, and value-added computation. Data management decisions are guided by user profiles. Thus, a broker has the following form..



More research is need to determine what goes on inside a broker, how best to build / customize them and whether or not it is really possible to deploy complex applications using a simple, uniform architectural approach similar to this.

Predicate-based connections

In a pervasive computing environment, naming of objects and computing elements must often be based on predicates instead of static addresses. If we consider publish/subscribe systems, a subscription is a name for collections of objects that match a predicate. Since applications can find themselves in unpredictable environments, they must qualify their needs instead of demanding specific sources. Connections to a network (or broker) may be specified in terms of predicates. For example, if an input stream shows up that is describing the position of a soldier who has just walked within range of the system, must get connected to a new input that can process such data. A particular re-supply report might get routed to all soldiers who are in need of more water.

Approximate queries

A modern RDBMS dutifully produces exact answers to declarative queries. While precision is obviously appreciated and important, there are times when it is not worth the resources that are consumed to achieve that precision. Moreover, in a pervasive setting, there are times when the resources are just not available. Thus, producing a good approximation to the precise answer is a reasonable alternative, especially if the system can characterize the degree of imprecision in the answer (e.g., ± 5%). In a pervasive environment, the available resources will vary over time. Determining how to adjust the precision dynamically is an important research topic.

Languages / Specification

Profiles. Like all push-based systems, a stream-oriented data management system is strongly dependant on profiles [2]. A profile is a specification of what data a user or an application is likely to need along with some way to rank this data in terms of its importance (i.e., its utility). Profiles inform the system where interest lies and, therefore, what data items should be pushed. Profiles must be expressed in a formal language if they are to be used to make automatic decisions. A profile language must be expressive enough to capture meaningful application needs, yet, at the same time; it must be simple enough to allow for efficient processing. It must provide a way to specify data of interest (this is related to queries) and the relative importance of qualifying data (data utility). Should utility be expressed in terms of numerical scores or in terms of partial orders?

Stream processing primitives. Much as the relational algebra provides the basic building blocks for computations in an RDBMS, a stream-processing system should be composed from a set of well-defined primitives. Such a set of primitives would enable an optimizer to manipulate programs to produce more efficient and possibly more meaningful results (with respect to a given set of profiles). Designing the "right" set of primitives here is obviously crucial to data management success.

Automatic data management

A human acting as the database administrator has traditionally made data management decisions. In the kinds of applications that we are envisioning, there is often no way that a human could be injected into the policy making process. Too many of the relevant variables are changing dynamically, too rapidly for a human to track. Moreover, the locus of control for much of the data is spread over many sites making it difficult for a human to coordinate this. Thus, there is a need for the database administrator role to be automated as much as possible. This involves collecting the right statistics and reacting to changes as rapidly as possible. We believe that user profiles can make this job easier since they can be used to predict future access patterns.

References

[1] Mehmet Altinel, Demet Aksoy, Thomas Baby, Michael J. Franklin, William Shapiro, Stanley B. Zdonik, *DBIS-Toolkit: Adaptable Middleware for Large Scale Data Delivery*. SIGMOD Conference 1999: 544-546.

[2] Mitch Cherniack, Michael J. Franklin, Stan Zdonik , *Expressing User Profiles for Data Recharging*, IEEE Personal Communications, August 2001.

[3] Michael J. Franklin, Stanley B. Zdonik: *"Data In Your Face": Push Technology in Perspective*, SIGMOD Conference 1998: 516-519.