

Architecture for Accessing Data Streams on the Grid

Beth Plale

Computer Science Dept.
Indiana University
plale@cs.indiana.edu

Abstract

Data streams are a prevalent and growing source of timely data. Existing systems that handle streaming data are often explicitly designed to serve the data streams. In the future we expect data streams to be viewed as just another input source to be consulted at will and on demand. These on-demand applications are often distributed, and either have significant computational or data access needs.

This paper introduces an architecture for flexible access to real-time streaming data on the grid based on the following fundamental observations:

- Aggregation of data streams as data resource – for a class of data stream systems, a viable view of the data streams they produce is as a data resource, where the canonical data resource is a database.
- Stream access through database operations – an intuitive view of stream access operations is in terms of database operations, specifically by means of a database query language. The recent burgeoning interest in the database research community on the topic of streams attests to this viability.
- Grid service-based access to data streams – access to data streams has a natural realization through the proposed Global Grid Forum Grid Data Service specification, which currently specifies access to databases.

The primary contributions of this paper are the definition of a data stream resource, an architecture for a data stream resource on the grid, and an identification of open research issues.

Keywords: grid computing, data stream systems,

publish-subscribe, continuous queries, OGSA-DAI, data management, dQUOB.

1 Introduction

Data streams are a prevalent and growing source of timely data [14]. Stream applications are broad: sensor networks monitor traffic flow on US Interstates; NEXRAD Doppler radars continuously generate streamed data for weather forecasting and prediction. Network traffic, financial tickers, and web servers continuously generate data of interest. The literature describes numerous systems in existence that handle streaming data. But whereas existing applications are often designed explicitly to serve the data streams, in the future we expect data streams to be viewed as yet another input source to be consulted at will and on demand. Just as it is common today to read starting conditions for an environmental simulation from a file, it should be equally as easy to draw those starting conditions on demand from live data streams.

These on-demand applications will be distributed and will have either significant computational needs or significant data access needs. As such, the Grid is an attractive computing framework because it promotes modularity through a service-oriented computing model, and provides scalability by virtue of its ability to amass resources that cross administrative domains. Early grids, those in existence today that span multiple departments on a university campus or sites on a company intranet, demonstrate the benefits attained from harnessing disparate and widely disbursed computational resources. Existing ef-

forts to date to integrate stream systems into the grid have been ad hoc in nature [5]. Needed is a general architecture under which existing stream systems can be brought onto the grid. The model needs to be closely tied to the specifications that are developing in the Global Grid Forum (GGF) because these specifications effectively define the future direction of data access in grid computing.

The purpose of this paper is to define the problem space, and introduce an architecture for flexible access to real-time streaming data on the Grid based on three assumptions:

- *Aggregation of data streams as data resource* – for a class of data stream systems, identified in this paper, a viable view of the data streams they produce is as a coherent data resource.
- *Stream access through database operations* — an intuitive way to think about stream access is in terms of database operations. The recent burgeoning interest in the database research community on data streams attests to the viability of this view.
- *Grid service-based access to data streams through Grid Data Service* – we posit that access to data streams should be through the framework specified in the proposed GGF Grid Data Service (GDS) specification [2].

The contribution of this paper is severalfold. It categorizes existing stream systems and proves that one class of systems has the necessary properties to be a data resource. We define an architecture for a data streams resource that extends the data access model and framework proposed in the GGF Grid Data Service specification. We demonstrate that access to streams by means of a database query language can be intuitive. Finally, we discuss key research issues in realizing the data streams model. Our current effort is to realize this architecture using our dQUOB system [22].

The conceptualization of a set of data streams as a unified data resource is not immediately obvious. We think there are several reasons for this. First, the term “streams” is very broadly defined and second, existing approaches to managing streams span a broad range of functionality as well. In Section 2 we address this ambiguity by categorizing streaming systems along orthogonal axes in order to expose the essential defining characteristics. We pinpoint one class of applications and show that this class has the properties that enable it to be categorized as a data

resource. In Section 3 we define the architecture of the *data stream store*, a grid enabled data resource (that is, a grid service for data) that delivers data stream results to clients through the grid services infrastructure defined by OGSi [26, 11]. Through two example systems, in Section 4 we establish the feasibility of realizing a streaming system as a data resource by identifying key features typically associated with data service systems (*i.e.*, a database) and show their syntactic and semantic representation in data streams applications. The paper concludes with a discussion of related work and conclusions, Sections 5 and 6 respectively.

2 Data Streams Systems

The term “streams” can mean many things. Rowset results stream back from a database a row at a time; a sequence of requests streams to a web server; a stock ticker, a click stream, and keystrokes from a keyboard are all streams. A stream could be a streaming MPEG movie, a set of readings from a sensor, or readings from an instrumented system. We distinguish the term “data stream” from the general term “stream”. A *data stream* is an indefinite sequence of time-sequenced events (also called “messages” or “documents”.) Events are marked with a timestamp indicating the time at which the event is generated, and often include a logical time indicating the time at which the application specific event occurred. Data streams differ from message passing systems in that data streams loosely couple distributed components with asynchronous time sequenced data whereas message passing systems support parallel or tightly coupled systems where communication is often synchronous. Data streams differ from mouse or keyboard events because the latter tightly couple an I/O device to a process.

Data stream systems are systems, often middleware, that operate on data streams. Data stream systems fall into three general categories: stream routing systems, data manipulation systems, and stream detection systems. The categories can be distinguished on the orthogonal characteristics of timeliness demands on the response and on the quantity of stream data that must be examined to process an arriving event (see Figure 1.) Timeliness, along the X-axis, is the latency between event generation and delivery of a result at one or more consumers. If a deci-

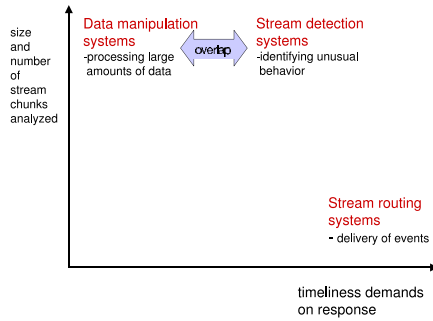


Figure 1: Types of data stream systems.

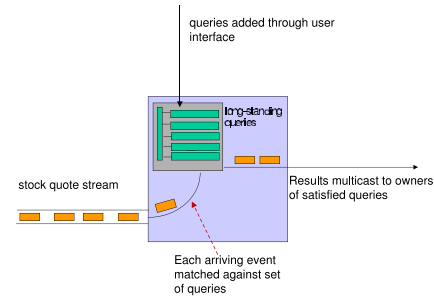


Figure 2: Stream routing system example.

sion must be made in on the order of milliseconds, then the timeliness demands are high; if source to destination delivery and processing can take minutes, hours, or days, then timeliness demands are low. Stream routing and detection systems have high timeliness requirements. The second characteristic is size and number of stream chunks analyzed. Systems that make decisions based on localized, single-stream, single-event chunks are low along the Y-axis. Those that must examine larger amounts of data either spatially (*i.e.* across streams) or temporally (*i.e.* within streams) in order to make a decision are high along the Y-axis. Though stream systems do not always partition neatly into these categories, in particular there is considerable overlap in data manipulation and detection systems, the general distinction is valid and forms the basis for the conditions under which a set of data streams can be considered a data resource. We refer to the act of operating on stream data as *decision-making*. Determining to where an event should be routed, whether or not to discard an event, or apply to it a transformation, whether or not a query is satisfied by the recent arrival of an event, are all forms of decision-making over data streams. We discuss each system in more detail.

Stream routing systems. Stream routing systems disseminate events (or documents or information about events) to interested recipients. These systems are known by many names: publish/subscribe systems such as ECho [9], NaradaBroker [12], and Bayeux [28]; selective data dissemination system such as XFilter [1], document filtering system such as Xyleme [20], message-oriented middleware (MOM) [15]. Stream routing systems

are distinguished by the locality of the information needed to make the decision. Decisions are made based almost exclusively upon the arriving event. Though some history may be maintained for instance to optimize performance, decision-making is largely localized to the immediate event at hand. The high delivery rates expected of such systems ensure that the decisions are kept simple.

A simple stream routing system is illustrated in Figure 2. A remote broker is shown receiving and distributing stock quote events. Users register their interest in particular stocks through submission of a query to the broker. The query might be, for instance, a Boolean expression, path expression, or Xpath query. Event arrival triggers query evaluation. This is quite unlike a database where query evaluation is triggered by query arrival. An arriving event is matched against the long-standing queries, and then is routed to the consumers indicated by the set of matching queries. The queries are long lived and are executed repeatedly. The expectation of these systems is that millions of queries can be active at any time. Key to achieving timeliness is the efficient matching of arriving events against a large number of long standing queries.

Data manipulation systems. Data manipulation systems are general stream processing systems that transform, filter, and aggregate data streams. Processing often results in the generation of new streams. There are looser timeliness requirements on the results on these systems than for stream routing or stream detection systems. For example, a large-scale instrument or set of instruments that generates large data sets can make the data sets available to the science community on the scale of hours later,

and after having undergone extensive transformative processing. The types of decisions in data manipulation systems can be framed as requests for data and for manipulation of the data, thus the language used to express the requests must be flexible enough to express these complex requests [4, 19, 22, 21]. Data manipulation systems can be based on the assumption of periodic streams, that is, the assumption of periodicity for all streams in the system. Sensor network systems display this characteristic.

Data flow programming problems are another form of data manipulation system wherein data flows that originate at one or more data generators, are consumed at one or more consumers, and undergo filtering and transformation along the way. For instance, large-scale real-time visualization data undergoes several transformations between the parallel models that generate the timestep data and the tool that visualizes it. This functionality is provided in systems such as dQUOB [22] and DataCutter [6]. In work done at Cornell on ad hoc wireless networking, intermediate nodes aggregate information flowing from sensors to source [27].

Detection systems. Detection systems detect anomalous or changed behavior in remote distributed entities. In these systems asynchronous streams are the norm, that is, no assumptions about periodicity can be made. Stream detection systems are used to monitor performance, such as in R-GMA [10], Autopilot [23], Gigascope [7], changes in HTML pages, such as in Conquer [18], or safety critical systems, such as dQUOB [24]. Though overlap exists between detection systems and data manipulation systems, the focus of the system drives the kind of support provided to users. A detection system that provides timely detection of anomalous behavior might put an emphasis on specialized operators for processing time markers.

2.1 Distributed Global Snapshot of Stream System

We assert that data manipulation and detection systems form a class of stream systems that meet the criteria of a data resource whereas stream routing systems do not (though exceptions do exist.) We define a *data resource* as an organized collection of information that has coherence, meaning, and value to a community. A relational

database has coherence and meaning in that the tables in the database are related to one another and the relationships have meaning. As such, the database is amenable to rich interrogation, analysis, and manipulation.

Data manipulation and detection systems meet the criteria of a data resource because these systems export their state or behavior, thus the streams collectively form a global snapshot of selected and relevant behavior of an application. Specifically, a distributed global snapshot typically consists of a snapshot of the processes in a distributed application plus all messages in progress between the processes. But by the nature of stream applications, which are organized around the production and consumption of data, the distributed global snapshot of a data stream application can be determined from examination of the data streams alone. That is, we can safely draw conclusions about behavior of the distributed application simply by examining the streams. This defining characteristic makes stream systems quite different from stream routing systems and from distributed systems in general. We illustrate with an example from a safety critical monitoring system at a nuclear power plant where temperature, pressure, and device sensors are employed to detect anomalous behavior. Each sensor exports an aspect of the behavior of the instrument it monitors. At any moment in time, the data streams from all sensors collectively form a global snapshot of selected and relevant behavior of the nuclear power plant distributed application. Events in one water temperature data stream are related to events in one water pressure data stream, and the water temperature events in one data stream are related to one another through time.

Stream routing systems, on the other hand, efficiently move events from providers to consumers in support of any number of simultaneous users. Payload is not important unless it affects the routing decision. Decisions are made on an event-by-event basis; an event either goes to a destination or it does not. The events in transit may or may not encode the state of a producing entity. Streams may or may not have meaning relative to one another. The implication of these broader requirements is that a distributed global snapshot for a routing system must include the state of the entities at each end of the pipe in addition to the messages themselves.

Data manipulation and stream detection systems exhibit the characteristic that embodied in their data streams

is a global snapshot. This condition is sufficient for these systems to be considered a data resource. That is, the global snapshot over a set of streams satisfies the requirements of coherence and meaning.

3 Architecture for Stream Resource

Data-driven applications require access to data resident in files, databases, and streams. We assert that access to data in streams should be as intuitive and uniform as access to files or databases. We believe that this goal is best achieved within the data access and integration framework proposed by the DAIS working group [2] of the Global Grid Forum. In this section we define the relevant terminology [2, 11] and framework [17] used in access and integration of databases on the Grid, then introduce a model and architecture that extends the framework to a data stream resource.

An *external data resource* is a source of data that has an existence independent from the grid services architecture. It is often a well-established system that operates autonomously and can be accessed through non-grid mechanisms. An external data resource is represented by proxies that are grid services. The canonical example of an external data resource is a database. A *Grid Data Service* is a grid service that provides access to an external data resource. It implements data-specific interfaces for data description, that is inquiring as to the kind of database and number of tables; data access, for issuing a query for instance; data management, for managing the grid service itself; and data factory, for creating a new instance of a data service or rowset data service. A *Rowset Data Service* is a virtualization of a result set resulting from a query to the Data Service. It is a new service instantiated through an invocation of a factory port for purposes of handling longer running activities, such as a query that takes numerous hours to complete. A Rowset Data Service maintains state about the session. The Grid Data Service and Rowset Data Service are depicted in Figure 3, and are shown interfacing to a stream resource instead of a traditional database.

The main thesis of this paper is that *data manipulation and detection systems can and should be viewed as an external data resource* and hence are amenable to access by means of the DAIS data access and integration

framework. An architecture that supports data streams should be general enough to accommodate multiple existing stream systems, many of which are cited throughout Section 2. These systems should retain the ability to be accessed by mechanisms besides the grid service mechanism. By modeling a data stream system as an external data resource, we provide a means whereby existing stream systems can be brought onto the grid.

We define a **data stream store** as a *data resource consisting of possibly distributed, domain-related data streams and associated computational resources, located in physical proximity to data streams and on which query processing is carried out. The data stream store is a grid enabled data resource able to interact with and serve clients and other grid services through the grid services infrastructure defined by OGSF [26, 11]. Providers of events are external to the data resource. The streams they generate are external to the data resource unless explicitly published to the store. Data streams produced as a product of data stream processing (i.e., views) are automatically part of the virtual data stream store.*

An example data stream store, illustrated in Figure 3, consists of nine data streams and associated computational resources. The computational resources, C_i , are contained within the virtual stream store but can be physically widely disbursed. Computational resources are located in physical proximity to data streams. The definition does not prohibit a computational resource from also being a provider. The providers, which include the radar in the lower left, and six sensors, two per sensor node for nodes $C1$, $C2$, and $C4$, are excluded from the virtual stream store. This is an important feature of the model. Through the feature, the model can accommodate a database implementation of a data streams system, that is, where data streams are resident in a database and are serviced by long running queries managed through extensions to the database management system [4, 13]. Exclusion of providers benefits the provider by allowing it to retain control over which data streams are visible to a virtual stream store and when the streams become visible. In the following section we probe more deeply into the suitability of a database access interface for a data stream store and define open research issues.

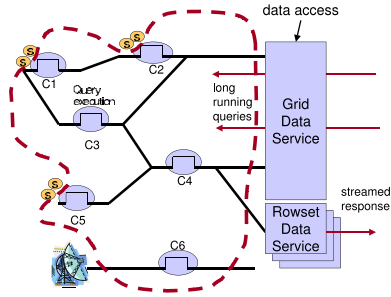


Figure 3: Data stream resource consists of streams and supporting computational resources encompassed within thick dotted line and grid services interface to the streaming store. Note that data providers are external to the stream store.

4 Details

We have demonstrated that the data stream store architecture supports an important class of data streaming problems on the grid. In this section we claim that a database access interface (e.g., a database query language) is a viable access API for stream systems on the grid. For example, a very practical solution to monitoring is to feed data streams from a nuclear reactor monitoring system for instance into a time-based database and perform post mortem processing of the data by means of issuing queries over the data streams that define hazard conditions. The usefulness of this query capability does not diminish were we instead to issue those same queries so as to obtain answers directly from the streams.

A database access interface provides a couple major gains beyond an intuitive interface. First, in examining the literature on streaming systems, one finds that most existing stream work has a database access interface. Thus our architecture can enable existing systems to be brought onto the grid. Second, the Grid Data Service specification and OGSA Data Services document together define a hierarchy of ports (or interfaces) for access to data resources. By adopting a database access interface, we leverage this work and are in a position to influence an extension to the Grid Data Service specification for streaming resources. The ability of the specification to accommodate a non-database data resource strengthens and validates its gen-

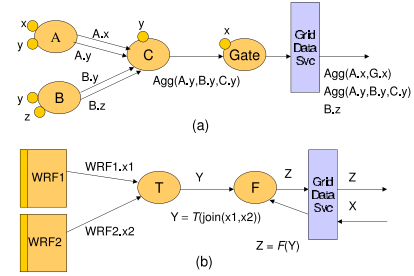


Figure 4: Data distribution in (a) sensor network example and (b) data flow example.

erality. On the other side, developers of streaming systems can develop in accordance to an architecture that is tied to an widely supported GGF specification.

In this section we highlight a few operations of the data access API for the model, then go on to identify open research issues. We first set the stage by giving two examples of existing data stream systems.

The first example is from the sensor network described in [27]. Sensor nodes are limited computational devices equipped with several kinds of sensors (e.g., temperature, light, PIR). Nodes are connected to neighbors in an ad hoc wireless network and use a multi-hop routing protocol to communicate with nodes that are spatially distant. Special gateway nodes connect to components outside the sensor network through long- range communication such as cables or satellite links; all communication with users goes through the gateway node.

Queries are long running and periodic; streams are assumed to be synchronous. The query sensor network aggregates readings of the distributed sensors at each timestep. In Figure 4 a) four sensor nodes are depicted, A, B, C, and Gate. Gate is the gateway node that communicates with the Grid Data Service to return a stream of aggregated values, one per timestep, for each sensor in the system. As can be seen, the periodic output from the sensor network is an aggregation of values for sensors x, y, and z at each timestep. The user in this example might be a portal interface that graphically displays results in real time.

The second organization is visualization flow processing as provided in dQUOB [22] where compute nodes are

located on the path between the parallel model (WRF1, WRF2) and its visualization; see Figure 4. Each node on the path performs filtering, transformation, or aggregating. Transformation might convert the data from spectral domain to grid domain by application of an FFT. Another might join data streams from the model with a stream from the user and filter according to the user preference in a particular region of 3D space. Figure 4 b) depicts two model components, WRF1 and WRF2, that push data through a transformation node (T) and filter node (F). At node T the streams WRF1.x1 and WRF2.x2 are joined and the function T applied to the result. The resulting stream of events, Y, are streamed to node F where the function F is applied, resulting in stream Z. The user in this example could be a visualization tool enabled for application steering.

4.1 Data Access Interface

The access mechanism to the data stream store is similar to the access mechanism in a database management system. In Table 1 we highlight several operations on a data stream store, show their similar database functionality, and list the relational grid access interface in terms of the base portTypes (left of the double colons) [11] followed by the relational realization [3].

<i>Stream request</i>	<i>Database operation</i>	<i>Grid services portType operation/SDE</i>
publish new stream to store	create new table	<i>not currently defined</i>
information about a stream	information about schema	DataDescription::RelationalDescription
issue stream query	issue database query	DataAccess::sqlQuery
describe stream result	info about instance of query result	DataDescription::RowsetDescription

Table 1: Data access interface: sample operations.

Publication of a stream to a store is analogous to table creation in a relational database. A stream is not visible to the stream store until explicitly published to it. The provider has control over which streams to publish

to the store and when to publish them. The proposed GGF specification for relational database access [3] does not yet define commands such as this that are traditionally specified through a database data definition language (DDL). Retrieval, manipulation, and operation of stream data is done by means of issuing database-like queries to the data stream store. A query is long-running, possibly distributed across the available computational resources, and must be pushed into the stream store. It is specified through a port analogous to the sqlQuery portType. The response from a query is a stream of results returned by the Rowset Data Service. Information about the result instance is obtained by issuing a request to the analogy of the RowsetDescription portType.

4.2 Research Issues

Integrating data streams from instruments and sensors into grid computing raises a number of research issues that we identify in this section.

Query distribution. The data stream store will in most cases be a distributed resource that provides location transparency. That is, the user writes a single query as if all streams (tables) were centrally located. But since the streams are distributed, some form of query distribution must be provided. As this kind of functionality is nearly universal in all stream systems we examined, it could be provided as a pluggable component in the Grid Data Service. The OGSA-DAI [8] reference implementation, for instance, is structured to handle pluggable modules in the Grid Data Service.

Data distribution. Data distribution models differ across the systems we examined. For instance, in the sensor network example of Figure 4a), records of the same sensor type but from different nodes have the same schema, and collectively form a distributed table. Where sensor nodes A, B, and C export the stream of their sensor y. The distributed table Y is the union of all A.y, B.y, and C.y. The operation performed on that distributed table is the aggregation of events based on timestamp. This is seen by the result streamed from C, namely Agg(A.y,B.y,C.y). Not shown is that C acts as an interme-

diate node also for streams A.x and B.z but does not operate on these. In the sensor network, data is distributed.

In the flow example, the data is federated. The parallel model components WRF1 and WRF2 both generate data of type X. But the stream for X generated by WRF1 is in a "table" named X1 whereas the stream generated by WRF2 is in a "table" named X2. While neither approach is better than the other, the examples illustrate that the Grid Data Service should be flexible enough to handle different distribution schemes.

Query distribution often includes assembling the results from the distributed sources. This functionality is less important in a Grid Data Service because for instance, in the sensor network, aggregation of results is done within the virtual stream store. In the flow-programming example, the query is broken into subqueries and placed in the stream resource in relation to other subqueries and based on proximity to the sources. Once again, the results are returned from the stream system in their completed form.

Management of long running queries. The queries themselves reside in and are executed in the virtual stream store. Query lifetime is often specified by means of extensions to the query language. If this support is not provided or local nodes of the virtual stream store can not manage query termination, this functionality would need to be provided by the Grid Data Service. The result of a long running query is a series of tuples generated over time that must be delivered by means of a stream delivery mechanism. The DAIS specification accommodates asynchronous request such as this by means of instantiating a Rowset Data Service.

Query instantiation. Query instantiation is the realization of a query in the virtual stream store. The user specifies a query as say, an ASCII SQL statement, but by some mechanism the query is executed inside the virtual stream store. This instantiation is typically handled in an application specific way thus support in the Grid Data Service for this as well must be modular and pluggable. To illustrate, suppose a set of hosts are available for use. In the Gigascope system [7], queries are pre-compiled into the executables that run on each of the nodes. In the case of dQUOB, the user submits a query that is compiled into a Tcl script that is then sent to the remote host. At the

host are a runtime system and a dQUOB library. The runtime system receives the script and invokes a Tcl interpreter. Through the process of interpretation the script calls the dQUOB library which instantiates the query as C++ objects of operators (e.g., select, project) connected as a DAG. Thus the query is transported as a compact script, but runs efficiently as compiled code at the node. Further, the dQUOB query language allows definition of a user-defined procedure to be executed as part of the query. These code chunks are dynamically linked into the query execution environment at runtime.

Updates. An update is the publication to a data stream that is a member of a virtual stream store. For reasons of performance and flexibility to support a broad range of systems, data publication from devices should be allowed to bypass the Grid Data Service. As such, the model allows point to point transfer of events using the underlying communication system. Sensor nodes on an ad hoc wireless network, for instance, do not have sufficient power to route communication through a Grid Data Service. Nor can they handle the 100-700 ms latencies it introduces [16]. But the Grid Data Service must support an update operation to enable client participation. For instance, stream X in Figure 4b) is the user operating through the visualization tool to push infrequent requests for particular data regions into the transformation *F*. DAIS support for update could be by means of persistent sessions in the Rowset Data Service.

5 Related Work

Related efforts in grid services for stream data are small in number. The OGSA-DAI project [8] has developed a set of services for connecting databases to the grid based on the GGF Grid Data Services specification. The system was recently released to widespread acclaim. The OGSA-DAI work is complementary to our work and in fact will serve as a key component in our implementation of the streaming architecture proposed in this paper. In the Antarctic monitoring project [5], the authors propose a grid services architecture for interacting with an environment sensing device located in Antarctica. The grid services architecture provides access to and control of the sensor device, and accepts the data stream from the re-

mote device via an iridium satellite phone network. The work differs from ours in that the service supports the combined functionalities of device management and access a stream of data. We view the two functionalities as distinct. Our data stream store is a mechanism for accessing data streams emanating from numerous devices. We suggest that access to the hardware that generates the streams should be via a device management service.

Narayanan *et al* [25] discuss a services oriented software infrastructure that provides database support for accessing, manipulating, and moving large scale scientific data sets. The service model differs from our work in that the target data resource is a database of large scale data sets. R-GMA [10] is a stream detection system for performance monitoring of the grid middleware. It could be cast into the architecture described in this paper, however we envision the data stream store as having value to application users, not grid middleware services. Additionally, R-GMA supports a more limited access language than is provided in such tools as dQUOB.

6 Conclusion

We propose a new model and architecture for accessing stream data on the grid. The model has several major strengths. It is general so as to accommodate existing stream systems. It is scalable to accommodate networks of sensors and large scale instruments. It extends the proposed GDS specification of GGF and as such gains the benefits of approximate compliance while testing the generality of the model implicit in the specification.

This is not to say that supporting a data stream store within the DAIS framework is trivial. The similarities are at a relatively coarse level; at the semantics level the differences are fundamental. But it is our belief that the architecture described herein will provide intuitive access to a valuable data source for the grid community. Ongoing work towards realizing this model is being done at IU in the dQUOB project.

7 Acknowledgments

The author wishes to thank the National e-Science Center, Edinburgh, Scotland for their gracious hospitality

while the author was there as a Visiting Researcher August-September 2003. Thanks also to Martin Westhead, Neil Chue-Hong, and Mario Antonioletti of the University of Edinburgh for discussion and insight.

References

- [1] Mehmet Altmel and Michael J. Franklin. Efficient filtering of XML documents for selective dissemination of information. In *Proceedings of 26th VLDB Conference*, 2000.
- [2] Mario Antonioletti, Malcolm Atkinson, Susan Malaika, Simon Laws, Norman Paton, Dave Pearson, and Greg Riccardi. Grid data service specification. In *Global Grid Forum GWD-R*, September 2003.
- [3] Mario Antonioletti, Amy Krause, Shannon Hastings, Stephen Langella, Susan Malaika, James Magowan, Simon Laws, and Norman Paton. Grid data service specification: the relational realization. In *Global Grid Forum GWD-R*, September 2003.
- [4] Shivnath Babu and Jennifer Widom. Continuous queries over data streams. In *International Conference on Management of Data (SIGMOD)*, 2001.
- [5] Steven Benford and et al. e-Science from the antarctic to the GRID. In *Proceedings of UK e-Science All Hands Meeting*, September 2003.
- [6] M. Beynon, R. Ferreira, T. Kurc, A. Sussman, and J. Saltz. Datacutter: Middleware for filtering very large scientific datasets on archival storage systems. In *Eighth Goddard Conference on Mass Storage Systems and Technologies/17th IEEE Symposium on Mass Storage Systems*, College Park, Maryland, March 2000.
- [7] Chuck Cranoe, Theodore Johnson, Vladislav Shkapenyuk, and Oliver Spatscheck. Gigascope: a stream database for network applications. In *International Conference on Management of Data (SIGMOD)*, 2003.
- [8] e Science Institute. Open grid services architecture data access and integration (OGSA-DAI). <http://www.ogsadai.org>, 2003.

- [9] Greg Eisenhauer. The ECho event delivery system. Technical Report GIT-CC-99-08, College of Computing, Georgia Institute of Technology, 1999. http://www.cc.gatech.edu/tech_reports.
- [10] Steve Fisher. Relational model for information and monitoring. In *Global Grid Forum, GWD-Perf-7-1*, 2001.
- [11] I. Foster, S. Tuecke, and J. Unger. OGSA data services. In *Global Grid Forum GWD-I*, August 2003.
- [12] Geoffrey Fox and Shrideep Pallickara. An event service to support grid computational environments. *Journal of Concurrency and Computation: Practice and Experience. Special Issue on Grid Computing Environments.*, 2002.
- [13] Dieter Gawlick and Shailendra Mishra. Information sharing with the Oracle database. 2003.
- [14] Lukasz Golab and M. Tamer Ozsu. Issues in data stream management. *SIGMOD Record*, 32(2):5–14, June 2003.
- [15] Ashish Kumar Gupta and Dan Suci. Stream processing of Xpath queries with predicates. In *International Conference on Management of Data (SIGMOD)*, 2003.
- [16] Deepti Kodeboyina and Beth Plale. Experiences with ogas-dai: Portlet access and benchmark. In *Global Grid Forum Workshop on Designing and Building Grid Services*, September 2003.
- [17] A. Krause, T. Sugden, and A. Borley. Grid data service. Technical Report Technical report OGSA-DAI 2003, July 2003. Document Identifier: OGSA-DAI-USER-UG-GDS-v4.1.
- [18] Ling Liu, Calton Pu, and Wei Tang. Continual queries for internet scale event-driven information delivery. *IEEE Transactions on Knowledge and Data Engineering, Special issue on Web Technologies*, January 1999.
- [19] Sam Madden and Michael J. Franklin. Fjording the stream: An architecture for queries over streaming sensor data. In *International Conference on Data Engineering ICDE*, 2002.
- [20] Benjamin Nguyen, Serge Abiteboul, Gregory Cobena, and Mihai Preda. Monitoring XML data on the web. In *International Conference on Management of Data (SIGMOD)*, 2001.
- [21] Clara Nippl, Ralf Rantza, and Bernhard Mitschang. Streamjon: A generic database approach to support the class of stream-oriented applications. In *International Database Engineering and Applications Symposium IDEAS*, 2000.
- [22] Beth Plale and Karsten Schwan. Dynamic querying of streaming data with the dQUOB system. *IEEE Transactions in Parallel and Distributed Systems*, 14(4):422 – 432, April 2003.
- [23] Randy Ribler, Jeffrey Vetter, Huseyin Simitci, and Daniel Reed. Autopilot: Adaptive control of distributed applications. In *IEEE International High Performance Distributed Computing (HPDC)*, August 1999.
- [24] Beth (Plale) Schroeder, Sudhir Aggarwal, and Karsten Schwan. Software approach to hazard detection using on-line analysis of safety constraints. In *Proceedings 16th Symposium on Reliable and Distributed Systems SRDS97*, pages 80–87. IEEE Computer Society, October 1997.
- [25] Narayanan Sivaramakris, Tahsin Kurc, Umit Catalyurek, and Joel Saltz. Database support for data-driven scientific applications in the grid. *Parallel Processing Letters*, 13(2), 2003.
- [26] S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, D. Snelling, and P. Vanderbilt. Open grid services infrastructure, version 1.0. In *Global Grid Forum GWD-R*, March 2003.
- [27] Yong Yao and Johannes Gehrke. Query processing in sensor networks. In *First Biennial Conference on Innovative Data Systems Research*, Asilomar, CA, January 2003.
- [28] S. Zhuang, B. Zhao, A. Joseph, R. Katz, and J. Kubiawicz. Bayeux: An architecture for scalable and fault-tolerant wide area data dissemination. In *Proceedings Eleventh International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV 2001)*, June 2001.