Realization of GGF DAIS Data Service Interface for Grid Access to Data Streams

Ying Liu, Beth Plale, Nithya Vijayakumar

Indiana University

Bloomington, IN

IU-CS TR 613

ABSTRACT

As the computation power of hardware increases, real-time processing of large amount of streaming data becomes possible. Streams are widely recognized as one of the major data resources. Meanwhile, many efforts are spent on the grid framework research, which enables people and applications across physical and administrative domains to easily share data and computational resources. Our work leverages the cutting-edge technologies in both areas to define a new grid-accessible streaming query and management abstraction, named Virtual Stream Store. In this paper, we describe our realization of the interfaces specified in the GGF DAIS Data Service proposal for streaming resources.

I. INTRODUCTION

In this technical report, we describe a realization of the DAIS Data Service specification [7] for a novel data resource. The resource is an abstract resource that is a collection of data streams that taken together define a body of information meaningful to a community. We follow the framework provided by OGSA Data Service proposal [4]. It is recommended that the reader be familiar with the OGSA Data Services proposal [4] and the more general Grid Data Service Specification [7]. The latter defines various portTypes that are extended in this document. At the same time, this document can be viewed as a parallel document to other realization documents, including the Relational Realization [5] and XML Realization [6].

The rest of this technical report is organized as follows. We provide an overview of the specification in the section II. In section III, we show how the system works using four use-cases. In section IV, we discuss the interface realization in detail. Conclusion is given in section V.

II. SPECIFICATION OVERVIEW

Virtual Stream Store (**VSS**) is an abstract collection of streams and computational resources [1]. More specifically, it has two major components. First, it contains a collection of distributed data streams that are related by coherence and meaning, and share a common communication substructure. Second, the virtual stream store has access to a set of computational resources located in physical proximity to data streams. The computational resources are where query processing is carried out.

The Grid Data Service (GDS) is a web service that provides a document-based access to a data resource. It adheres to the OGSA Data Services proposal [4] and as such is the interface to the GDS is through three general portTypes: Data Access,

Data Factory, and Data Management. The GDS used in our work is based on the GDS of OGSA-DAI [10] with a few extensions and exceptions necessitated by the needs of the virtual stream store model. A GDS exists in a one-to-one relationship with a Virtual Stream Store. With the exception of a plug-in that talks to a specific stream system, the GDS is agnostic to any specific implementation of the stream system. The GDS tracks all active queries and computational resources that belong to its virtual stream store.

dQUOB [11] is an example of a back-end continuous query stream processing system. The **dQUOB stream system provides dynamic query execution over data as the data moves from providers to consumers in event channels. The VSS is a logical collection of streams that share meaning and coherence.** dQUOB server shown in Figure 1, is a persistent service that controls access to the dQUOB stream system [2]. A stream query processing system can generate streams that belong to one or more VSS; hence a VSS is in a many-to-one relationship with the backend dQUOB system. More generally, dQUOB can be replaced by any stream processing system implementation, for example, the Stanford Stream server [8] as shown in Figure 1.

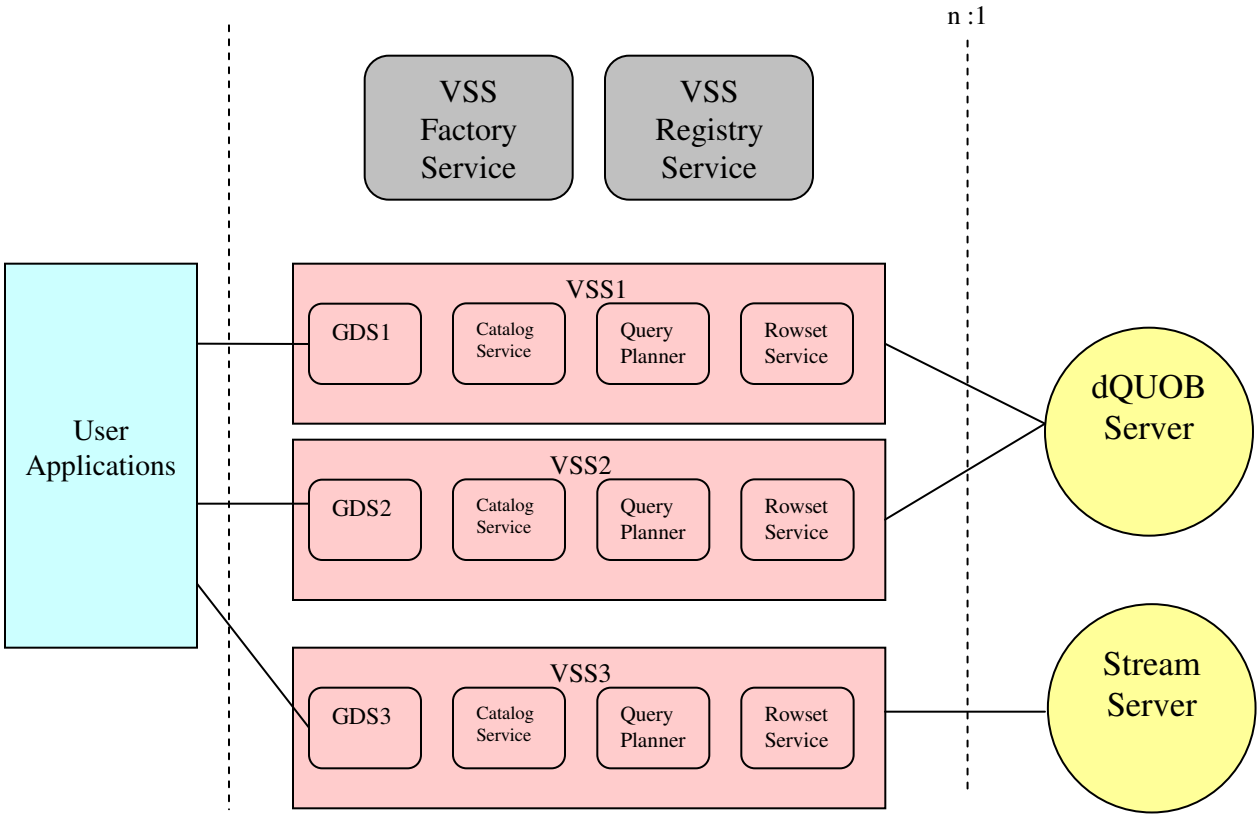

Figure 1 Virtual Stream Store components and interaction with users and backend stream processing systems.

The streaming realization is a specification of data services and interfaces for a streaming data resource. We define Grid Data Service as a persistent service that serves a single Virtual Stream Store [1]. As defined before, a **Virtual Stream Store is an abstract collection of streams and computational resources**. The Grid Data Service depicted in **Figure 2** is a grid service composed of four sub-service categories. A sub-service category can be viewed as a functional unit that exports a unique set of interfaces from the other functional units.

The various functionalities that must be supported in the Virtual Stream Store can be viewed as falling into four sub-service categories, where each virtual interface is composed of methods that cross three general port types:

I. **SQL Logical Interface**: the data resource is a stream processing system, hence is accessed by means of a continuous query language. This language is typically an extension of SQL. There is a single persistent instance of the SQL Data Service.

II. **Rowset Logical Interface**: exports a set of interfaces that enable a user to manipulate the results of a continuous query. Each Rowset Data Service can serve for several user submitted queries. The Rowset Data Service is in existence for the duration of the often long-lived continuous query.

III. **Stream Publish Interface**: We view a data resource as a set of streams. The users of this service are stream providers. Through this service a stream provider can add or remove a stream and also change its characteristics.

IV. **Admin Logical Interface:** Provides grid access for administrator to manage the streams and the computational resources in the system.

A virtual interface exports the standard set of interfaces as defined in [7], DataAccess, DataFactory, and DataManagement. The DataAccess interface allows access to the data resource itself. This could be a query issued to a DBMS, to update a set of rows, or retrieve a tuple from a result set. The DataFactory interface allows for the instantiation of a new instance of a service. The DataManagement interface allows management activities on the data service itself. The interfaces and operations of the GDS are summarized in Figure 2. The details are explained section IV.

SubService	Operation
SQL Logical Interface	SQLAccess:sqlQuery
	SQLAccess:dropQuery
	SQLFactory:createService
	SQLMgt:listQueryStatus
Rowset Log. Inter.	RowsetAccess:getTupleTi
	RowsetAccess:getTuples
	RowsetFactory:createService
Stream Pub. Logical Interface	PublishAccess:createChannel
	PublishAccess:registerStream
	PublishAccess:publishData
	PublishAccess:unregisterStream
	PublishAccess:removeChannel
Admin Log. Inter.	AdminAccess:addComputationalElement
	AdminAccess:dropComputationalElement

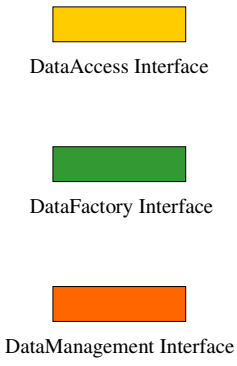

Figure 2 Four sub-services of GDS

III. TOUR OF SPECIFICATION BY EXAMPLE

In this section, we demonstrate functionality of VSS through four use-cases.

Case I. *Administrator Alice wants to set up a new instance of a VSS named VSS1. How does she go about doing this?*

VSS Factory service shown in Figure 1, is a persistent service that responds to request for creating a new instance of Virtual Stream Store. To instantiate a new instance of VSS means to instantiate the GDS, catalog service, query planning service and the rowset service. GDS is the gateway to the Virtual Stream Store and it calls other services internally. The API discussed in technical report is implemented in the GDS. A persistent VSS Registry service shown in Figure 1, keeps track of the GDS handles for different VSS instances. The information of all input data streams and computation nodes registered to a specific VSS instance is maintained by its catalog service. The query planner records information of all the queries and based on this information and generates a query plan for the incoming query. The rowset service helps users manage streaming query results. All above services are built on Open Grid Services Architecture (OGSA) model [3, 11] and implemented according to the OGSI specification [9]. Besides starting these persistent services, Alice also needs to specify the publish/subscribe protocol used by the backend stream processing system and its query language. Then a new VSS1 is created and it is ready to take new streams, computation resources, continuous queries and follow-up rowset queries.

Case II: *How does Stream provider Bob, publish his streams into an existing VSS1?*

Bob first obtains the handler of the GDS (say 'GDS1') corresponding to VSS1 by searching through the VSS Registry service. Then he publishes streams into VSS1 using GDS1's create_channel() interface. Since each GDS is associated with only one VSS, Bob need not specify the name of the VSS instance to which he publishes streams. But he needs to specify the name of the channel along with data format information of be added to the underlying publish/subscribe system.

create_channel (channelName, dataType);

The first parameter to the operation is a channelName which is meaningful logical name for Bob to refer this channel later. The second is type definition for this channel. After this, VSS will register this new channel and its type (format) information and return a global unique channel identifier for the new channel. Later Bob uses channelName to refer this channel and publish data to it as shown below.

publish_data (channelName, streamName);

But the VSS system maps (Bob, channelName) to the system-wide unique channelId for reference.

Case III*: User David submits 3 long running queries (q1, q2,q3) on VSS1 through GDS1 and is handed a pointer to three RingBuffers (one for each query) on the Rowset Data Services, from which he can retrieve results.*

David must first contact the registry to obtain the handle for GDS1 corresponding to VSS1. Then he submits the queries through GDS1's interface SQLAccess:sqlQuery(). To use this interface, David needs to specify the query string and query name. Associating a name with the query enables easy identification of the query. VSS1 creates a new RingBuffer on the rowset service for each query submitted. The pointers (IDs) to the newly created RingBuffers are returned to David.

RingBufferIDList = sqlQuery(qName1,qExp1,qName2,qExp2,qName3,qExp3);

The streamed results are returned and stored in the corresponding ringbuffers.

Case IV*: User Mary submits one long running query to VSS1 through GDS1 that is exactly the same as one of the queries that David submitted.*

This case is similar to the previous one from the user point of view. But the system notices that Mary's query is exactly the same as an executing query. To save system resources, we will let Mary and David share the same RingBuffer. However, for security reasons, if either of them specify that they do not want to share their RingBuffers, VSS will create a new RingBuffer for Mary and David each.

IV. DATA SERVICE INTERFACE

The individual interfaces are defined as follows:

A. SQL Logical Interface

SQL Data Service is a sub-service which virtualizes the data resource as a SQL database. User communicates with a data resource through SQL-based continuous query language. The service interfaces can be classified into DataAccess interface, DataFactory interface and DataManagement interface as shown in Figure 2.

1 **SQLAccess:sqlQuery** operation is used to direct pre-defined continuous query to data resources. The continuous query is expressed in SQL like language. And user can give his continuous query a logical name, which can be used to refer to this query later. We overload this operation with two different parameter sets. One is for a single query submission; the other is used to submitting several queries at a time. Both operations are listed below.

RingBufferID = sqlQuery(queryName, queryString);

This operation accepts a single continuous query. It has two inputs, queryString and queryName. queryString is defined in the continuous query language accepted by the backend stream processing system. queryName is the logical name of SQL query that will be used as the handle for this query. The output is a rowsetPtr, which is the pointer to the rowset data service. The rowset data service is where the result is stored. The rowset data service is explained in detail a little later.

RingBufferIDList = sqlQuery(qName1,qExp1,qName2,qExp2……qNamen,qExpn) ;

This operation overloads the previous one with different input, output parameters. User can submit several queries at a time with this operation. The input can have several different pairs of (qName, qExp), each of them specifies one submitted query. Accordingly the output is a list of pointers to the ring buffers created in the rowset set service that holds results for these queries.

Possible exceptions from these operations are InvalidQuery, InvalidQueryName and ConflictQueryNumber. InvalidQuery means a query either semantically incorrect or fails during the execution. InvalidQueryName means that query name either violates naming scheme or already exists. ConflictQueryNumber points out that there is conflict between the number of queries submitted and the number of names user specified.

- **SQLAccess:dropQuery** operation can stop/drop a particular continuous query from the system.

  True/False = dropQuery (queryName);

  The input of this operation is queryName. It is the logical name of the query to be dropped from VSS. The output is a boolean value indicating success or failure of the operation. A possible exception message is invalidQueryName, indicating the specified query name does not exist in the system for this user.

- **SQLFactory:createService** operation belongs to DataFactory interface. It is used to create a new data service, which can be an empty service (no resources associated), a service with associated resources or a session between client and user. Based on different requirements, we overload this operation with different input parameters.

  True/False = createService(VSSName);

  This operation has a single parameter, VSSName. The new created service is has no associated resources initially. The input parameter is VSSName, which is the name of VSS. The output indicates whether the operation succeeds or not.

  True/False = createService(VSSName, resourceConfig);

  This operation has two input parameters. VSSName is the logical name for new VSS and resourceConfig is the file where the resource configuration is described.

  Possible exceptions are invalidVSSName and invalidConfig. invalidVSSName can be caused by the violation of the naming scheme or the specified name already being used. invalidConfig indicates that either the configuration file in the wrong format or it specifies some resources which do not exist.

- **SQLManagement:listQueryStatus**
  This operation is used to check the status of specific query.

queryStatus = listQueryStatus (queryName);

The input parameter is the queryName, which is the name of specified query. The output, queryStatus, is the status of specified query, which can be PLAN, DEPLOY, WAIT and ACTIVE. The status indicates the different stages for the query: under the plan generation stage, under deployment, waiting for the incoming data or under execution. Possible exception is invalidQuery, which can be caused by specifying a non-existing query.

B. Rowset Data Service

- **RowsetAccess:getTupleTi** operation is used to access tuple received at timestamp Ti in the ring buffer.

  tuplePtr = getTupleTi(timestamp, ringBufferID);

  The inputs of this operation are timestamp and ringBufferName. timestamp specify the time when the interested tuple is received. ringBufferID indicates the targeted ring buffer. Possible exceptions are invalidRingBuffer and tuplesNotAvailable. invalidRingBuffer indicates that the specified ring buffer does not exist. tuplesNotAvailable means that the user specified tuples are not available until the query fired time.

- **RowsetAccess:getTuples:** the operation is used to access tuples falling into a specific time period.

  resultPtr = getTuples(startTime, endTime, returnDataMode, ringBufferID);

  There are four inputs: startTime and endTime are timestamps to define the interested time period. returnDataMode indicates how the result data will be returned. There are two modes: BULK_RETURN and STREAM_RETURN. The former chunks all the data together and return the bulk. STREAM_RETURN mode returns data continuously as a stream. ringBufferID specifies the ringBuffer where the tuples are cached. Under the BULK_RETURN mode, the output is the pointer to the storage where all the chunked data is stored. Under the STREAM_RETURN mode, the output is the pointer to the channel where the stream comes.

  Possible exceptions are invalidTimeRange, invalidRingBuffer, tupleNotAvailable. invalidTimeRange means startTime is larger than endTime. invalidRingBuffer indicates that specified Ringbuffer does not exist on the service. tupleNotAvailable indicates satisfied tuples are still not available in the ringBuffer.

- **RowsetDataFactory:createService** operation is used to create a new Rowset Data Service. In a VSS, there must be at least one Rowset Data Service as it is a persistent service component for VSS. Typically an instance of Rowset Service is started when the VSS is created. At the beginning, all the RingBuffers are created on that Rowset Service. Later, for system's scalability, new Rowset Service can be created. This createService call is issued by the system without user

invocation. When the user submits a query into system and the maximum number of ringbuffers is reached on the main Rowset Data Service, system will automatically create an auxiliary Rowset Data Service.

rowsetPtr = createService();

The operation has no input parameter and the output is the pointer to the new created Rowset Data Service.

C. Stream Publishing Service

- **PublishingAccess:createChannel** operation is used to create new publishing channel in the system. This operation is overloaded with two separate parameter sets. One is to create a new channel at a time. The other enables to create several channels once.

  True/False = createChannel (channelName, dataType);

  The inputs to this operation are dataType and channelName. dataType is the data type or dataformat definition for the new channel. channelName is reference name for the new created channel, which must be unique in the whole system. The output indicates whether the operation succeeds or not.

  True/False = createChannel (channelNameList, dataTypeDefFile);

  This operation can create several channels at a time. The input parameters are channelNameList and dataTypeDefFile. dataTypeDefFile is a formatted file which includes a list of data type definitions**.** ChannelNameList is the list of referenced names of new created channels. Output indicates the operation status. There is not partial success.

  Possible exception is conflictName which means that the specified name is already taken by some channels. Another possible exception is that invalidDataType which indicates the dataType definition file does not follow the pre-defined formats.

- **PublishAccess: registerStream** operation is used to register a new stream resource to an existing channel.

  streameId = registerStream(channelName, streamFormat);

  The inputs are channelName and streamFormat. channelName specify where the streaming data will go. streamFormat defines the format of the data to be published. Output is the system-wide unique stream ID that will be used later as a reference for the stream.

  Possible exception is that nonExistChannel and conflictFormat. nonExistChannel means the specified channel does not exist in the system. conflictFormat means the stream has different dataformat from the channel's format.

- **PublishAccess: publishData** operation can be used to publish the stream to a specific channel.

  True/False = publishData (streamId, streamData);

  The inputs are streameId and streameName. streamId is the system-wide identifier got from the stream registration operation. The streamData points to the data to be published to the system.

  Possible exceptions are nonExistStreamId. nonExistStreamId means that no such stream has been registerd.

- **PublishAccess: unregisterStream** operation is to unregister the stream from the channel.

  True/False = unregisterStream (channelName, streamId);

  Inputs are channelName and streamId. channelName specifies the channel where the stream will un-register from. streamId specifies which stream will be un-registered.

  Possible exceptions are nonExistChannel, nonExistStream and invalidOperation. nonExistChannel means the specified channel does not exist, nonExistStream means no such stream is registered to the channel. invalidOperation means this operation is not allowed to be issued by the user, as VSS only allows the stream provider to un-register the stream from the channel.

- **PublishAccess: removeChannel** operation is used to remove channels from the VSS.

  True/False = removeChannel(channelName);

  The input is channelName which is the logical name of channel to be removed. Only the provider who creates the channel can request to remove that channel.

  A possible exception is nonExistChannel, which means the channel does not exist. Another exception is invalidOperation which means this channel is not allowed to be removed as there are still some streams attached to it.

D. Admin Service

- **AdminAccess:addComputationNode** operation adds a new Computational Node in a Virtual Stream Store.

  nodeId = addComputationNode(hostName, configFile);

The inputs are hostname and configFile. hostName can be used to identify the contributing computer on the Internet. The configFile is the name of a formatted file which provides some information for this host. The information can be the memory size, cpu speed, location and etc. The resources provider can even specify how much they want their resources to be shared in the configFile. The output is the global unique indentifier for this computation node.

Possible exceptions are existHost and invalidConfig. existHost means that this host has been already registered as computation node in this VSS. invalidConfig indicates the configuration file violates the pre-defined format.

l **AdminAccess:removeComputationNode** operation aims to remove a computation node from the Virtual Stream Store.

True/False = removeComputationNode(nodeId);

The input of this operation is nodeId, which is the global unique identifier for the node, generated when the node is added into VSS. The output shows the execution status.

Possible exceptions are invalidNode and invalidAccess. invalidNode means the specified node does not exist. And invalidAccess indicates that user don't have permission to remove this node because only the resource provider can do this.

V. CONCLUSION

Data streams are main data resource for many applications, such as Weather forecasting, Stock market financial analysis, Network traffic monitoring and so on. The management of streaming resources plays a key role in these applications. At the same time, Open Grid Service Architecture (OGSA) [3, 11], aims to facilitate the sharing of data and computing resources, which is needed by these streaming applications. In this paper, we described the realization of OGSA Grid Data Service in Virtual Stream Store, a stream query and management system. By providing grid access to streams, Virtual Stream Store enables sharing of data and computation resources among different streaming applications.

ACKNOWLEDGMENT

The authors would like to thank Mario Antonioletti and Neil Chue Hong at EPCC for their helpful discussion and support.

REFERENCES

1. B. Plale, Architecture for Accessing Data Streams on the Grid, *In Proceedings of 2nd European Across Grids Conference (AxGrids),* 2004.

2. B. Plale, K. Schwan, Dynamic Querying of Streaming Data with the dQOUB System, *In Journal IEEE Transactions on Parallel and Distributed Systems,* 14(4), 2003.

3. I. Foster, H. Kishimoto, A. Savva, D. Berry, A. Djaoui, A. Grimshaw, B. Horn, F. Maciel, F. Siebenlist, R. Subramaniam, J. Treadwell, J. Von Reich, Open Grid Services Architecture, Version 1.0, *Global Grid Forum Informational Document,* Continuously updated.

4. I. Foster, S. Tuecke., J. Unger, OGSA Data Services, *Global Grid Forum 9,* 2003.

5. M. Antonioletti, A. Krause, S. Hastings, S. Langella, S. Malaika, J. Magowan, S. Laws, N. W. Paton, Grid Data Service Specification: The Relational Realization, *Global Grid Forum 9*, 2003.

6. M. Antonioletti, A. Krause, S. Hastings, S. Langella, S. Malaika, S. Laws, N. W. Paton, Grid Data Service Specification: The XML Realization, *Global Grid Forum 9*, 2003.

7. M. Antonioletti, M. Atkinson, S. Malaika, N. W. Paton, D. Pearson, G. Riccardi, Grid Data Service Specification, *Global Grid Forum 9*, 2003.

8. R. Motwani, J. Widom, A. Arasu, B. Babcock, S. Babu, M. Datar, G. Manku, C. Olston, J. Rosensteain, R. Varma, Query Processing, Resource Management, and Approximation in a Data Stream Management System, *In Proceedings of Conference on Innovative Data Systems Research (CIDR)*, 2003.

9. S. Tuecke, K. Czajkowski., I. Foster, J. Frey, S. Graham, C. Kesselman, T. Maquire, T. Sandholm, D. Snelling, P. Vanderpilt, Open Grid Services Infrastructure, *Global Grid Forum 8*, 2003.

10. M. Antonioletti, M. Atkinson, R. Baxter, A. Borley, N. P. Chue Hong, B. Collins, N. Hardman, A. C. Hume, Alan Knox, M. Jackson, A. Krause, S. Laws, J. Magowan, N. W. Paton, D. Pearson, T. Sugden, P. Watson, M. Westhead, Design and implementation of Grid database services in OGSA-DAI, *In Journal Concurrency and Computation: Practice and Experience,* 17 (2-4, 357-376), 2005.

11. I. Foster, D. Gannon, The Open Grid Services Architecture Platform, *Global Grid Forum 7, Working Draft*, 2003.