

Distributed Technologies for Remote Access of HDF Data

Svetlana G. Shasharina, Chuang Li, Nanbor Wang, Rooparani Pundaleeka, David Wade-Stein
Tech-X Corporation, 5621 Arapahoe Ave, Suite A, Boulder 80303
{sveta, chuli, nanbor, roopa, dws}@txcorp.com

Abstract

Scientific simulations and experiments use sophisticated data formats to store and access their data. An example of such a format is the Hierarchical Data Format (HDF), commonly used in fusion and plasma physics, geosciences, astronomy and medical research. Most HDF data gets generated remotely (at a remote supercomputer or an experimental site) and is not readily available to the scientists. Transferring the whole data to the users' machines for analysis and visualization might be prohibitive because of the size of the data, bandwidth limitations or local storage limitations. In addition, different subsets of data may be interesting for analysis at different times. Thus, scientists need a solution for querying and accessing subsets of remote data. This paper describes several approaches to provide an access to remote HDF data and compares their performances. It also gives some details about the solution that we consider the winner – a Globus-based Web Service.

Index Terms—HDF5, Web Service, Globus, GridFTP, CORBA

1. INTRODUCTION

As the scale of both experiments and simulations grows, the size of generated data is becoming more and more challenging. For example, in a typical plasma fluid simulation of a three-dimensional grid with 2,000x200x200 points, one must save at least 10 values (three components of electric and magnetic fields along with density and flow velocity for a single fluid) per grid cell. This implies that the size of the simulation data in a single time slice can easily reach 6.4 GB. In particle simulations, the problem is magnified – one can easily have 5 particles per cell, with each particle described by six or more variables. For such simulations, the size of the data in a single time slice is now closer to 26 GB, and one can generate multiple time slices (easily 30 or more) per run. Hence, the data produced by a single 3-D simulation easily approaches a terabyte. Examples of 3-dimensional plasma physics and fusion codes producing large datasets at remote location include hybrid PIC-fluid simulations, such as VORPAL [1], and large MHD codes, such as Nimrod [2] and M3D [3].

Such data-intensive and computation-intensive applications require supercomputers, while the analysis and visualization of the data produced are typically performed on local clusters

and desktops. The data is commonly stored in the Hierarchical Data Format (HDF) [4], which has a convenient API to query and access data of interest through methods that transfer data into local memory directly.

It would be very convenient for scientists to be able to use a similar API that would work on remote (rather than local) data. Once the remote dataset gets into the client's local memory for analysis and visualization, it can be cached for further reuse. Thus the main challenge here would be to determine the most useful functions, use them as a foundation for the remote API, and provide the remote API implementation that would be efficient and allow for the “to-memory” access. These challenges directed the work presented in this paper.

In the remainder of this paper, we first provide some background on technologies used in our work: the HDF data format, the C/C++ Web Services—as provided by the Globus toolkit [5] and gSoap [6], and CORBA [7] in section 2. Next we present our prototype solutions using these technologies for the remote HDF data access in section 3 and evaluate their performance in section 4. We then describe the capabilities of the chosen and fully implemented solution in section 5. We also briefly discuss related projects in section 6. Finally in section 7, we provide our concluding remarks and future directions.

2. BACKGROUND

2.1. HDF Data Format

Self-describing data formats are commonly used for the storage of simulation data. Historically, simulation data was stored using some ad hoc format. For example, an electromagnetic simulation might first store the dimensions of the data, then the first field, the second, etc. This method of data storage presents difficulties especially when new versions of a simulation programs are released. Self-describing data formats eliminate this problem by allowing the code to store data by name. The file storage software then takes care of developing an index for the data storage. In addition, the data can be assigned attributes, so that the consumer of the data is able to obtain information about the units, dimensions and other metadata for a particular variable.

Additionally, the self-describing formats now in use avoid binary incompatibilities. Since different architectures and platforms use different binary representations for data, a binary file written in one, may not be readable by another.

The self-describing data file is written in a universal binary format, which can be read back correctly on any platform.

The Hierarchical Data Format (current version is HDF5) and the Network Common Data Form (NetCDF) [8] are commonly used in multiple scientific communities including fusion and plasma physics, astronomy and medicine. HDF5 stores two primary objects: datasets and groups. A dataset is essentially a multidimensional array of data elements, and a group is a structure for organizing objects. One can create hierarchical data groups to store scientific data structures such as images, arrays of vectors, and structured and unstructured grids. Additionally, HDF5 also allows for parallel I/O. The NetCDF does not allow for arbitrary depth structure, nor does it allow for parallel I/O.

The full HDF5 C API is very powerful and detailed, but somewhat low-level. As a result, the HDF5 community developed the HDF5 Lite API [9], which consists of a set of higher-level functions encapsulating a series of commonly performed lower-level HDF5 operations into a few common tasks. For example, instead of using low-level HDF5 API functions to traverse the HDF5 hierarchy to locate a dataset, the HDF5 Lite API allows users to specify a dataset using a complete path name and will locate the dataset using this name. This higher level API is being used in this work as a model for the remote HSF5 data access.

2.2. Middleware Technologies

The following subsections present the middleware technologies used for the prototype solutions for providing access to remote HDF5 data.

2.2.1. CORBA. The Common Object Request Broker Architecture (CORBA [7, 12]) is an open standard for distributed object interaction defined by the Object Management Group (OMG). CORBA standardizes many common network-programming tasks such as object implementation, registration, instantiation, and, location and language transparency. CORBA also defines language mapping specifications for mapping the Object Request Broker (ORB) interfaces/constructs to various popular languages. ORB is the basic mechanism by which objects transparently make requests to, and receive responses from other objects on the same machine or across a network.

By ensuring the quality and the interoperability to the standards, ORB vendors are able to continuously evolve and optimize ORB implementations while allowing users to switch between ORB products freely and with minimal effort. The General Inter-ORB Protocol (GIOP) is CORBA's standard wire protocol for message exchange. The Internet Inter-ORB Protocol (IIOP) is a concrete realization of GIOP over TCP/IP. GIOP adopts a binary data representation format called the Common Data Representation (CDR) that allows standardized and efficient message exchange between ORBs.

Unlike Web Services, which use XML documents as the common format for message exchange, CORBA uses CDR. This allows messages to be exchanged in binary format and is more efficient both in terms of processing overhead and protocol overhead. Despite the maturity of CORBA, it is not commonly used in scientific applications. It is widely used, however, in real-time and embedded applications as well as high-performance enterprise applications.

This project used a CORBA implementation called TAO [13], developed by the Distributed Object Computing Group of Washington University in St. Louis, the University of California, Irvine, and Vanderbilt University. TAO is an open-source, high performance and highly configurable ORB implementation of CORBA specifications. TAO supports the standard OMG CORBA object reference model and real-time CORBA specification and includes numerous enhancements designed to ensure efficient, predictable, and scalable QoS behavior for high-performance and real-time applications.

2.2.2. Globus and gSoap Frameworks. Globus has become the de facto standard Grid software used in multiple data intensive scientific applications. This toolkit went through multiple changes and in its recent reincarnation, became a framework for creating stateful Web Services with bindings in Java, C++ and Python. Because we aim to use the C HDF5 API on the back end of the services, we have based our work on the WS C Core from Globus-4.0.2.

As a standard WS compiler, Globus takes a WSDL document as its input and generates the stubs and skeletons, as well as libraries for developing SOAP/XML Web Services clients and servers. The implementation is then added to be used by the generated stubs and skeletons, to complete the service.

We also evaluated the use of gSoap, which is an open source project developed by Dr. Robert A. van Engelen and his team at Florida State University. In addition to being a C/C++ WS compiler, gSoap includes a highly optimized implementation of the SOAP engine. Additionally, gSOAP supports many performance enhancement mechanisms, such as connection keep-alive, HTTP chunking, Base64-encoded message compression, and Direct Internet Message Encapsulation (DIME) streaming [10]. Due to the existence of these features, and because of its open licensing terms, we chose gSOAP as one of the candidate solutions for the remote HDF data access.

We used DIME to move data in binary format in our gSoap test because DIME messages do not have the 33% overhead which is associated with the Base64-encoding [11]. DIME specification allows inclusion of multiple binary files in a single package. A typical DIME package contains a *Message Begin* flag, multiple binary records, and a *Message End* flag. Each record contains a header, which includes information such as the size of the record, which is not known at the beginning of the transfer. When the records are transmitted,

DIME simply adds the end-record flag, resulting in much faster and more flexible processing.

3. PROTOTYPE SOLUTIONS

3.1. CORBA System

The system implemented using TAO-1.5.2 was a classic CORBA system with a C++ client and server. For data retrieval service using CORBA, we implemented the operation as defined in the following CORBA IDL definition:

```
typedef sequence <octet> OctetSeq;
OctetSeq getDataset (in string dataset);
```

The implementation of this method on the server side, delegates to a set of HDF5 C language API functions which extract the specified dataset from a given HDF5 file. In addition, there was also a method for querying the dataset's metadata, which has information about the dataset such as rank, dimensions and data type. This information is needed to perform memory allocation on the client. Although CORBA provides a very efficient mechanism for marshaling and transmitting most data types, the getDataset operation still returns the retrieved array as a block of binary data in our implementation to avoid even this minimal overhead.

3.2. H5WS System

This system was built using the C WS Core of Globus (Figure 1). The interface of the service prototype has methods for retrieving the metadata of a dataset as well as extracting a dataset. The C++ client uses the pointer reference of the remote service obtained through the stubs generated by Globus to delegate all the calls to the remote service. The client class also contains a pointer to a data-fetching class over which type the client service is templated, allowing us to substitute the data-fetching mechanism as needed. Only one data-fetching class is implemented at the moment (GridFTP [14] client) and it uses the GridFTP C client API to get data into its memory from a remote file.

The server implementation uses the relevant methods of HDF5 C API and performs actions such as retrieving metadata and extracting a dataset, and writes into a temporary file, which can then be accessed by the GridFTP client.

The service is built on the C WS core of GT4.0.2 and uses HDF5 library 1.6.4 and 1.6.5 for HDF5 related operations. The service implementation has been fully tested with the gcc 3.2, 3.3, and 3.4 compilers.

3.3. gSoap System

The gSoap-2.7 system was implemented similar to the CORBA system, although its definition uses C syntax: The interface is defined in a C header file, which is used by gSoap

to generate the stubs and skeletons for the metadata query and dataset retrieval methods. Developing server implementations using the HDF C API concluded the implementation of this prototype system. Enabling DIME is done using a macro definition for CXXFLAGS while compiling the server and client sources.

The CORBA and gSoap systems are similar in the sense that the data is transferred from the remote host to the client memory as a block of binary data.

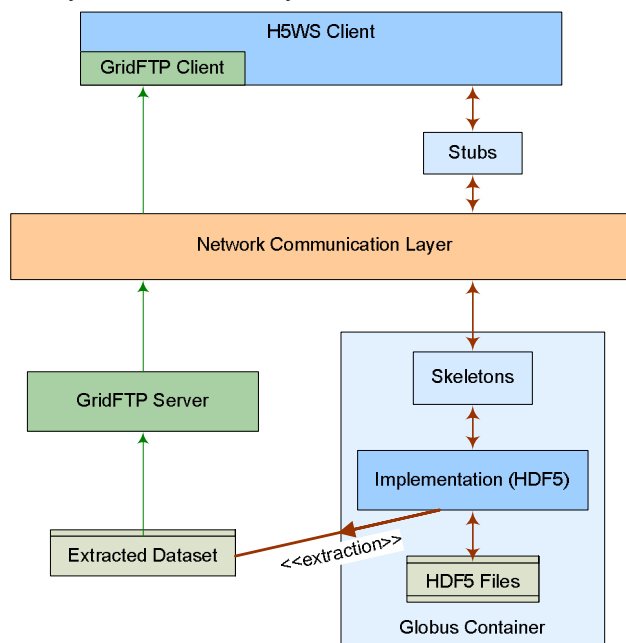


Figure 1. The architecture of H5WS.

4. PERFORMANCE TESTS

4.1. Testing Setups

Setup	Bottleneck bandwidth (Mbyte/sec)	RTT (msec)	BDP
LAN	12.5	0.27	3.4 KB
ESnet	125.0	72.0	9.0 MB
WAN	0.19	162.0	31.4 KB

Table 1. Network parameters for three connection scenarios. RTT is the Round Trip Time used to calculate the Bandwidth Delay Product (BDP).

In order to evaluate the systems described, we performed tests devised from a typical usage scenario. This scenario assumes that the remote data is located on a well-connected host with large storage capacity but having different client bandwidths. The three connection setups (summarized in Table 1) are as follows:

1. LAN: the server and the client are connected via a

100Base-T switch. For these tests we used two computers in our facility connected via a 100Mbps Ethernet switch.

2. **WAN:** the server and the client are connected via the “regular” Internet backbone. We performed this test using a server at NERSC, and a client at our facility which has an external connection to the Internet through a T1 (1.54 Mbps) connection.
3. **ESnet:** the server and the client are connected via a high speed wide-area network, such as the ESnet (OC-3: 155 Mbps) [15], sponsored by the US Department of Energy. For this configuration, we used a server at National Energy Research Scientific Computing Center, NERSC and a client at the Princeton Plasma Physics Laboratory. The Computer at NERSC is connected to the ESnet through a 10Gb interface, while the machine at PPPL uses 1Gb interface.

The benchmark test is designed to accurately measure the performance using three transport mechanisms adhering to a typical usage scenario of data retrieval services consisting of the following steps:

1. Locate the server and establish connection.
2. Query the metadata of target dataset.
3. Using the metadata, implicitly or explicitly allocate dynamic client memory for accommodating the dataset.
4. Retrieve the dataset into the client memory.
5. Disconnect from server.
6. Release the memory and clean all the resources.

Regardless of the actual mechanism used, the remote data retrieval service models the remote operations following the style of the HDF5 high-level API. This allows clients to identify the target dataset by specifying the HDF5 file name and the full pathname of the dataset in the HDF data hierarchy. Two operations are provided: query the metadata of the dataset and retrieve the dataset. Three different systems described in section 3 were evaluated here—the gSoap Web Service, the CORBA system and H5WS. P1 and P2 for the ESnet setup indicate 1 and 2 streams respectively.

For the H5WS implementation (based on GT4 and GridFTP), to locate the server and establish a connection, we create the client class, load in necessary GT4 client modules, and initialize the client handles. Once the dataset is located, it is extracted and dumped to a temporary file on a server machine. On the other side, the client allocates a local buffer for the dataset based on its metadata. The GridFTP client is then initialized, and activated to transfer the temporary file from remote server to the local buffer. After the data is obtained, we destroy the GT4 based client and GridFTP client.

4.2. Testing Results

We collected a set of HDF5 files from the VORPAL and M3D codes, and performed 25 iterations of the described usage scenario for extraction and transfer of datasets ranging

in size from 0.5MB to roughly 40MB. The average performance results are presented in Figure 2.

This figure shows that in the LAN setting, H5WS has too much overhead and is less efficient than the other solutions. In fact, the CORBA system saturates the connection in this case, while gSoap saturates about 50% of it. The H5WS overhead is mostly due to the fact that the connection is secure and also because the extracted data is stored into a temporary file on the servant machine. This overhead becomes negligible in the other scenarios, though.

In the WAN scenario, H5WS gives the best results and uses

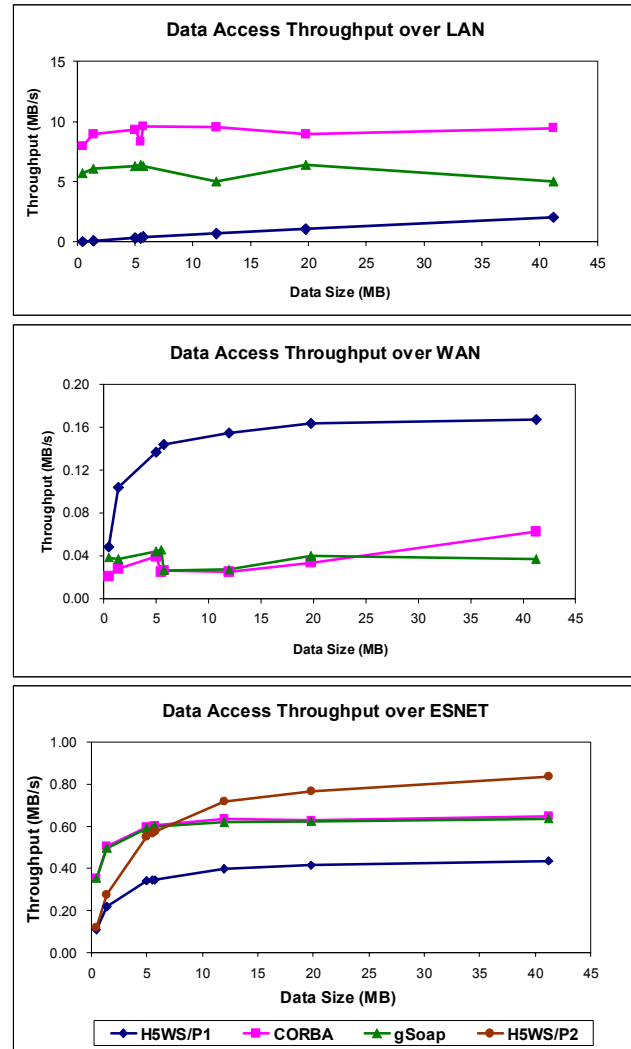


Figure 2. Data access throughput versus the size of transferred data for LAN, ESnet and WAN setups.

the bandwidth well, while other systems underperform.

In the ESnet scenario, H5WS seemingly loses if a single stream is used for GridFTP. But with an addition of just one more stream, it fares better than CORBA and gSoap.

4.3. Using Multiple Streams of GridFTP

In our tests using multiple streams of GridFTP of the H5WS system, we collected data for all three scenarios again. In LAN and WAN settings multiple streams did not provide any advantage and we do not show these results. This was expected, since multi-streaming works best for large bandwidth and large latency connections, as is the case for ESnet. This is demonstrated on Figure 3. The amount of the gained advantage depends on the data size (starts degrading at some point), though, as demonstrated in Figure 4.

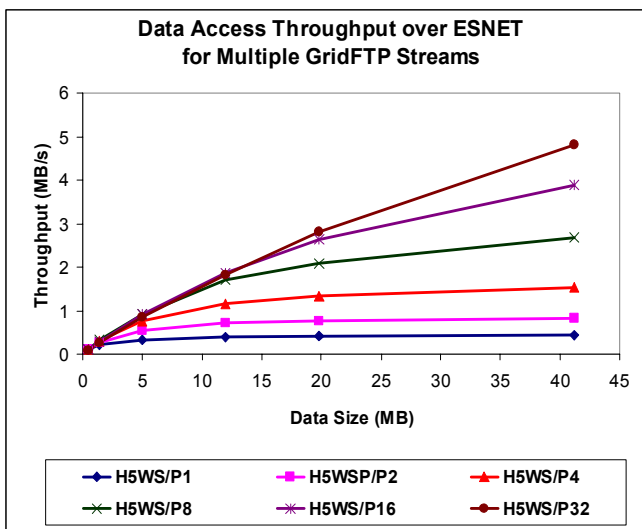


Figure 3. Throughput as a function of the dataset size for various numbers of GridFTP streams for ESnet setup.

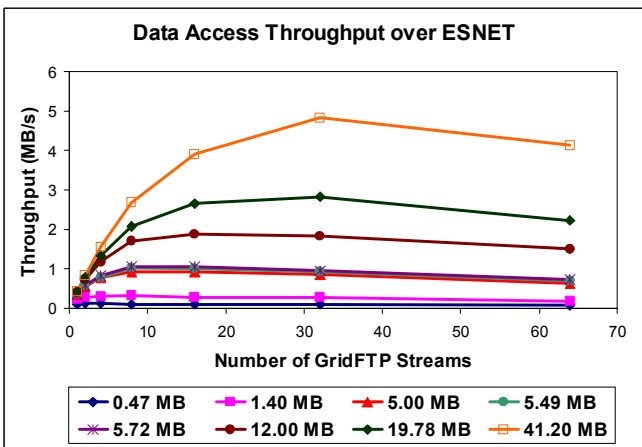


Figure 4. Throughput as a function on the number of GridFTP streams for datasets of various sizes for the ESNET setup.

In terms of saturating the connections, CORBA did this for the LAN scenario, while H5WS had too much overhead. gSoap's overhead was smaller and it filled the pipe close to 50%. In the ESnet scenario, none of the solutions was close to the bottleneck, because the BDP is much larger than the TCP window (64KB in all cases) and the data could not possibly fill up the pipe. In the case of WAN, only H5WS managed to be close to the bottleneck connection although the window sizes were appropriate for all systems.

The main conclusion of our tests is that the H5WS prototype provides overall best performance for large datasets and has advantages in the ESnet and WAN scenarios, which are the situations typically faced by researchers accessing their remote data. In addition, Globus is widely accepted in the scientific data intensive applications, while gSoap and CORBA are not very familiar to most of scientists. Hence we proceeded with the full development of H5WS. The capabilities of this service are described in the next section.

5. H5WS STATUS

Once the evaluation of the prototype systems was completed, we concluded the implementation of the H5WS solution [16]. We faced a couple of issues during the implementation:

Firstly, Globus generates Makefiles for client and server, but does not provide means to include external files and libraries prior to the generation of Makefiles. We had to overwrite the Globus configuration files, and create out custom shell scripts to generate the final Makefiles. Secondly, only the person who starts the service (Globus container) could use the service. We discovered that this is present only in the C Core, and not in the Java Core. We obtained a patch from the Globus developers which fixed this problem, the service could be accessed by multiple clients.

Our current H5WS implementation allows the user to open a remote HDF5 file, query remote HDF5 data to determine if a dataset of interest exists, find its metadata (rank, dimensions and data type), bring the dataset into local client memory, extract a hyperslab (an arbitrary cube of data) of interest, or retrieve the entire file if desired. The API was designed in such a way that it has a feel and look of the local HDF5 C API so that users do not have to switch to new types of functions.

In addition to the C client API, we provided a convenient command line interface allowing users to perform the same actions without becoming familiar with the Globus API. This interface allows users to choose the server (URL and port number), type and name of dataset to be extracted, and number of the GridFTP streams.

The H5WS system has been installed at Tech-X, Princeton Plasma Physics Laboratory, and NERSC and is being evaluated and tested by the members of VORPAL and NIMROD collaborations.

6. RELATED RESEARCH

The importance of remote access of HDF5 data was recognized a while ago and gave rise to the RemoteHDF5 [17, 18] project at the Zuse Institute Berlin, which started approximately at the same time as our project. The Zuse team implemented an HDF5 plugin to the stripped GridFTP server. The main difference of their approach is that it used the pre-Web Service Globus and thus provided a solution, which does not have a Web Service interface. Having a Web Service interface is important given the fact that Web Service is rapidly becoming the standard way to express remote capabilities.

Other possible approaches could involve experimentation with the Logistic Networks [19] and OpenDAP [20], but this work is out of the current scope of the project that concentrated on services allowing custom interfaces.

Alternative to our approach would be to do data visualization and analysis remotely and bring in only the visualization objects locally. Such approach is implemented in some visualization tools like VisIt [21]. Our approach is more low-level but allows for further data transformation on a client side and combination with multiple visualization engines.

7. CONCLUDING REMARKS AND FUTURE DIRECTIONS

This paper outlines the challenges facing the scientific community when dealing with ever-growing remote data, and describes and evaluates several solutions allowing for querying and accessing remote HDF5 data. In addition, we have described our solution, which uses the WS C Core from the Globus Toolkit and GridFTP as the underlying data transfer.

In future work we will consider extending the service with the capability of indexing HDF5 data so that only data manifesting particular features would be transferred. In addition, we plan to provide visualization clients for such systems as AVS/Express and VisIt.

8. ACKNOWLEDGEMENT

The work on this project is funded by DOE under contract # DE-FG03-01ER83842 and by Tech-X Corporation.

9. REFERENCES

- [1] <http://www-beams.colorado.edu/vorpal/>.
- [2] <http://www.nimrodteam.org/>.
- [3] <http://w3.pppl.gov/~jchen/>.
- [4] <http://hdf.ncsa.uiuc.edu/HDF5/>.
- [5] <http://www.globus.org>.
- [6] Robert A. van Engelen and Kyle Gallivan, "The gSOAP Toolkit for Web Services and Peer-To-Peer Computing Networks", in

the proceedings of the 2nd IEEE International Symposium on Cluster Computing and the Grid (CCGrid2002), pages 128-135, May 21-24, 2002, Berlin, Germany.

- [7] <http://www.corba.org>.
- [8] <http://www.unidata.ucar.edu/packages/netcdf/>.
- [9] http://hdf.ncsa.uiuc.edu/HDF5/hdf5_hl/.
- [10] http://gotdotnet.com/team/xml_wsspecs/dime/WSDL-Extension-for-DIME.htm.
- [11] Svetlana G. Shasharina, Nanbor Wang, John R. Cary, "Grid Service for Visualization and Analysis of Remote Fusion Data", In Proceedings of the International Workshop on Challenges of Large Applications in Distributed Environments (CLADE 2004) pages 34-43 Honolulu, Hawaii 2004.
- [12] Michi Henning and Steve Vinoski, *Advanced CORBA Programming in C++*, Addison-Wesley, Reading, MA, 1999.
- [13] <http://www.cs.wustl.edu/~schmidt/TAO.html>
- [14] <http://www.globus.org/datagrid/gridftp.html>.
- [15] <http://www.es.net>.
- [16] <https://collaborate.txcorp.com/collaborate/distributed-technologies/fusion-data-grids-project/>.
- [17] <http://www.zib.de/visual/projects/gridlab/hdf5/>.
- [18] Hans-Christian Hege, Andrei Hutanu, Ralf Kähler, André Merzky, Thomas Radke, Edward Seidel, Brygg Ullmer. Progressive Retrieval and Hierarchical Visualization of Large Remote Data. Proceedings of the 2003 Workshop on Adaptive Grid Middleware, p. 60-72, 2003.
- [19] <http://loci.cs.utk.edu/>.
- [20] <http://www.opendap.org/>.
- [21] <http://www.llnl.gov/visit/>.