

Grid Service for Visualization and Analysis of Remote Fusion Data

Svetlana G. Shasharina, Nanbor Wang, John R. Cary
Tech-X Corporation, 5621 Arapahoe Ave, Suite A, Boulder 80303
{sveta,nanbor,cary}@txcorp.com

Abstract

Simulations and experiments in the fusion and plasma physics community generate large datasets at remote sites. Visualization and analysis of these datasets are difficult because of the incompatibility among the various data formats adopted by simulation, experiments, and analysis tools, and the large sizes of analyzed data. Grids and Web Services technologies are capable of providing solutions for such heterogeneous settings, but need to be customized to the field-specific needs and merged with distributed technologies currently used by the community. This paper describes how we are addressing these issues in the Fusion Grid Service under development. We also present performance results of relevant data transfer mechanisms.

Index Terms—Fusion, Grid, Web Service, MDSplus, CORBA, HDF5, Data Analysis

1. Introduction

The fusion and plasma physics community, along with other scientific communities, is facing increasing challenges in the analysis of remote datasets due to (1) the sheer size of these datasets; (2) the diverse formats used to store the data, and; (3) heterogeneous data analysis and visualization tools adopted by different research project teams.

As the scale of both experiments and simulations grow, the size of generated data has becoming more and more challenging. For example, in a now common 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 3D 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, like VORPAL [1], and large MHD codes, such as Nimrod [2] and M3D [3]. Such data and compute intensive applications require supercomputers, while the analysis and visualization are typically performed on local clusters and desktops.

Large fusion experiments in the US also generate a lot of raw and analyzed data. The National Spherical Torus Experiment (NSTX) fires on the order of 3000 times per year [4], generating roughly 0.5 TB of data in a year. The DIII-D tokamak [5] at General Atomics fires approximately 2,000 times in a year with about 0.25 GB of unprocessed data stored for each shot. This corresponds to roughly 0.5 TB of data per year. The Alcator C-Mod tokamak [6] at the Plasma Fusion Science Center at MIT fires roughly 2,000 times each year, with approximately 0.2 to 0.4 GB of data per shot. The total archive contains between 2 and 3 TB of unprocessed data. The Tokamak Fusion Test Reactor [7] fires over 60,000 times with the data taken for each shot sometimes exceeding 1GB, resulting in a total dataset exceeding 10 TB. Finally, the International Thermonuclear Experimental Reactor (ITER) [8], a proposed international fusion reactor, will likely generate more than 1GB of data in every shot.

Because of the size of these datasets, it is important for physicists to access remote data efficiently. One way to approach the problem of analyzing remote and bulky data is to create tools for accessing the selected datasets from the remote files for analysis and visualization using local software without caching the data locally. Another way is to use Grid Replica Service to create data replicas on a more cost effective location on the network, and then access the selected data from a replica for analysis and visualization. Creating data replicas make accessing the data more cost effectively the second time it is needed. In either case, an efficient “file-to-memory” data access mechanism should be implemented that can select and utilize appropriate transfer mechanisms depending on the size and the format of the accessed data.

The diversity and the incompatibility in data formats adopted by different programs, however, make accessing data file difficult. For example, NSTX, DIII-D and Alcator C-Mod use MDSplus [9] as a data storage system and data format, while many simulations produce data in HDF5 [10] format. This makes integration of simulations and experiments and

comparison between different simulations problematic if one uses conventional approaches of providing a monolithic data access implementation.

Moreover, different research teams often prefer heterogeneous data analysis tools. For example, Princeton Plasma Physics Laboratory uses AVS/Express [11] and Interactive Data Language (IDL) [12]. General Atomics analyzes its data using IDL. In addition, packages like OpenDX [13] and SciRun [14] are used in NIMROD and VORPAL collaborations.

The problem of providing efficient transfer of large datasets in highly heterogeneous environments is addressed by Grid technologies [15]-[17]. However, problems related to mixing particular data formats and data analysis tools are specific to the fusion community and cannot be resolved without the active participation of fusion scientists.

The idea of applying Grid technologies to fusion and plasma physics data leads us to start the Fusion Grid Service project. This project seeks to provide a Grid Service framework for connecting together heterogeneous datasets coming from various fusion simulations and experiments and bringing the selected data into well-established analysis and visualization tools. It will also provide a way to select the optimal data-transport protocol automatically depending on size and type of data and the network and platform configuration. These protocols might include MDSip (the native MDSplus protocol), GridFTP [18] (from Globus), SOAP with binary encoding, and/or DIME [19] and CORBA [20]. The proposed Fusion Grid Service will include both HDF5 and MDSplus servers specialized for accessing data in the corresponding format. The client interface will provide uniform, transparent, and easy access to data of either format. The C++ client interface will be integrated into tools such as Interactive Data Language and AVS/Express for easy visualization and analysis of remote data.

Implementing the fusion grid as a Grid Service has the advantage of getting a standard set of Quality of Service features, provided by Open Grid Service Infrastructure, including lifecycle control, sophisticated security, information discovery and registry mechanisms. A typical use of the Fusion Grid Service will be when participants of the M3D project request remote HDF5 datasets generated by M3D, from their AVS/Express visualization applications as if they were local and compare them with experimental data originating in a different site in MDSplus format. A second example would be when a General Atomics physicist accesses remote MDSplus signals from her IDL program, analyzes it and compares the results with the analysis using HDF5 data coming from the NIMROD code.

The rest of this paper describes our work on the Fusion Data Grid project and is organized as follows. Section 2 provides background information for technologies on which the Fusion Grid Service is based. Section 3 illustrates the architectural

overview of the Fusion Grid Service. Section 4 presents performance comparison results of several data transporting mechanisms we plan to use in the project and discusses how further optimizations can be applied. Finally, Section 5 provides concluding remarks and future directions of this project.

2. Background

In this section we provide a brief description of technologies that will be used in the implementation of the Fusion Grid Service.

Grids and Web Services

The Open Grid Services Architecture (OGSA) has evolved from Computational Grid technologies and Web services. Exposing Grid technologies as Web services has several advantages. First of all, Web services intentionally accommodate extreme heterogeneity including applications, network protocols, and operating systems. In order to manage this heterogeneity effectively, Web services use a ubiquitous protocol infrastructure (such as TCP/IP, HTTP and SMTP) as well as XML-based messages and metadata, such as Simple Object Access Protocol (SOAP) [21]. SOAP is a simple enveloping mechanism for XML that can support message passing or RPC-style communication models. Web services are described in the XML-based Web Services Description Language (WSDL) [22].

Another advantage of using Web services is that they support dynamic discovery and composition of services. Finally, industrial-wide support of Web services ensures that Web services can exploit numerous tools provided by industry, such as WSDL processors generating various bindings and hosting environments from the specified interfaces.

As specified today, however, Web services are not suitable for distributed scientific computation. They do not address the Quality of Service (QoS) required by Grid applications, such as a common security policy, distributed workflow, resource management, and real-time and hardware constraints. Most importantly, Web services suggest a persistent service model, while Grid computing has often to deal with transient instances of services. The goal of OGSA is to enhance Web services with the requisite service management, QoS, dynamic deployment, soft-state management, and standardization.

An OGSA development framework, such as Globus Toolkit 3.0, bridges the Grid Service interface and the run-time hosting environment. It provides infrastructure and libraries that support security, discovery, resource management, invocation, communication, exception handling and data management and transfer. For example, GridFTP is a powerful extension to FTP providing rapid and secure transfers of files. GridFTP implements several transfer rate enhancements, including use of multiple TCP streams and

striped data transfer, which increases parallelism by allowing data to be striped across multiple hosts. Additionally, GridFTP allows for third-party control of data transfer, so that one host can initiate and monitor a transfer between two other sites. At this point, GridFTP has not exposed APIs for extracting a specific dataset from a file. However, there is ongoing work to expose API to allow external plug-ins for data access and filtering be installed into GridFTP. The Globus XIO [23] currently under development will be able to incorporate this extended GridFTP features, as well as other capabilities such as parallel data streaming.

The current release of Globus provides only Java hosting environment with the C/C++ hosting environment now expected in spring of 2004. For this reason, we are evaluating generic C/C++ Web Services tools, such as gSOAP. gSOAP [24] is an open-source project developed by Dr. Robert A. van Engelen and his team at Florida State University. The gSOAP toolkit includes the compiler tool that generates C/C++ stubs and skeletons for creating Web Services, and the libraries for developing SOAP/XML Web Services clients and servers. In addition, gSOAP includes a highly optimized implementation [25] of the SOAP engine. Finally, gSOAP supports many performance enhancement mechanisms, such as connection keep-alive, HTTP chunking, Base64-encoded arrays message compression, and Direct Internet Message Encapsulation (DIME) streaming. For these reasons, and because of its open licensing terms, we chose gSOAP for implementing the performance tests required to determine the optimal Fusion Grid Service configuration.

Data Transfer Mechanisms and Fusion Data Formats

In this section we describe candidates for data transfer mechanisms for the use in the Fusion Grid Service, a means to combine them in this service and switch them on the fly depending on the data type size, and data formats, which we intend to support in our service.

MDSplus

MDSplus [9] is a client-server system for acquiring, storing and analyzing of data that is used for all major fusion experiments (including D3D, NSTX, CMOD) in the US and many other international systems. It is simultaneously a data format and a transfer mechanism highly suited for fusion experiments. The MDSplus data transfer is performed through a server application called MDSip on a single socket TCP/IP connection. During experimental operation, data is gathered from experiment signals and placed in the MDSplus server as data trees. Subsequently, the data is accessed through the MDSplus interfaced by data analysis code, which then puts the analyzed results back into MDSplus. An example of this is the TRANSP code [26] that is heavily used at Princeton Plasma Physics Laboratory and other USA labs.

Use of MDSplus has begun to extend beyond experiments to simulation codes. An example is the NIMROD project, which has the principal goals of developing a modern computer code suitable for the study of long-wavelength, low-frequency, nonlinear phenomena in realistic toroidal geometries, making use of a multi-disciplinary, multi-institutional team, and making the final code available to the whole fusion community. Data from the NIMROD project has been written to an MDSplus tree and is available for subsequent visualization. Because visualization tools are being developed for NIMROD data in MDSplus trees, it would be advantageous to have a client that supports access to simulation data through the MDSplus API.

Access to MDSplus is built into many software tools used at fusion experimental sites. For example, the DIII-D experiment stores all analyzed data in MDSplus trees and provides access to it from ReviewPlus [27] and EfitTools [28].

Recent development of MDSplus includes the adoption of Globus I/O, a server that uses X.509 certificates for client/server authentication and authorization while a client will be required to do a grid-proxy-init before connecting to a "grid enabled" server.

HDF5 Data Format

Self-describing data formats are increasingly being used for the storage of data generated by simulations. Historically simulation data was stored in a location dependent manner. That is, 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 program are released. Self-describing data formats eliminate the 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. The data can then also be accessed by name. In addition, the data can be assigned attributes, so that the reader of the data can query about the units, dimensions and other metadata for a particular variable.

Additionally, the self-describing formats now in use take care of binary incompatibilities. Because different processors use different binary representations for numbers, a binary file written by one processor may not be readable by another processor. The self-describing data file reading and writing software ensures that the data is written in a universal binary format on all processors, and the reading software translates that to the format appropriate on input.

In common use in the fusion community are the Hierarchical Data Format [10] (with HDF5 being the current version) and the NetCDF [29] format. HDF allows one to create a structure inside of the HDF file, so that one can create data groups, inside of which one can have data and data groups, etc. The NetCDF format does not allow for arbitrary depth structure in the same way. Additionally, HDF5 has parallel I/O.

The full HDF5 API is very powerful and detailed but somewhat low-level. That is why the HDF5 community came up with the HDF5 Lite API [30] which consists of a set of higher-level functions, which encapsulate 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 and locate the dataset, HDF5 Lite API allow users to specify the dataset using complete path name and will locate the dataset from the name.

HDF5 has become very popular within the fusion community. Examples are 3D Gyrokinetic Toroidal Code (GTC) [31] and M3D [3] code used for studying neoclassical and turbulent transport and Magneto-Hydrodynamics effects in tokamaks and stellarators. Recently, NIMROD also started supporting HDF5. Finally, VORPAL [1], a versatile N-dimensional hybrid plasma simulation code developed at the University of Colorado, Boulder, and Tech-X Corporation, dumps data in HDF5 format. That is why, in addition to MDSplus, HDF5 is the format for which we have decided to support in our project.

Binary Data and SOAP

As HDF5 provides only API to access local data files, our Fusion Data Grid framework will provide a distributed HDF5 API. A straightforward solution is to implement the API using Web Services. However, as was mentioned above, Web Services use SOAP for message and data passing. SOAP is XML based, and binary data has to be converted into plain text before it can be sent over the wire.

Therefore, we will encode the scientific data as binary chunks using Base64-encoded arrays in our benchmark tests. There is also a DIME specification that allows inclusion of multiple binary files in a single package with files preserved in their original state. A typical DIME package contains a *Message Begin* flag, multiple binary records, and a *Message End* flag. Each record contains a header that 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. This results in much faster and more flexible processing.

CORBA

The Common Object Request Broker Architecture (CORBA) is an open distributed object infrastructure defined by the Object Management Group (OMG). OMG is an industrial consortium that, among other things, oversees the development and evolution of CORBA standards and their related service standards through a formal adoption process. The process has proven to be highly effective in ensuring the quality, the interoperability, and the implementability of newly adopted standards. CORBA standardizes and automates many common network programming tasks such as object

implementation, registration, and location transparency. CORBA also defines standard language mappings of most popular languages for the programming interfaces [32] to services provided by the Object Request Broker (ORB). An 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 of the standards, ORB vendors are able to continuously evolve and optimize ORB implementations and users can freely switch between ORB products with minimal efforts. The General Inter-ORB Protocol (GIOP) is CORBA's standard message exchange protocol over the wire. The Internet Inter-ORB Protocol (IIOP) is a concrete realization of GIOP using 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 much more efficient both in terms of processing overhead and protocol overhead.

Our benchmark tests use TAO [33], 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 reference model and real-time CORBA specification with many enhancements designed to ensure efficient, predictable, and scalable QoS behavior for high-performance and real-time applications.

Visualization and Data Analysis Tools

In this section we describe the data analysis and visualization tools that we intend to use on the client side of the Fusion Grid Service and the means to call C code from these tools. We need this capability in order to incorporate them into C/C++ client bindings, typically provided by Web/Grid Services. We chose to support IDL and AVS/Express because these tools can be used to analyze data from the majority fusion simulations and experiments.

Interactive Data Language

Interactive Data Language (IDL) [12] is extremely popular in physics, mathematics and medical applications. There are several reasons for this. To start, IDL is available on almost all platforms. IDL is a scripting language, which allows for rapid prototyping and immediate debugging. IDL has many built-in capabilities for data analysis, as well as read and write capabilities for the most widely used scientific data formats (netCDF and HDF).

IDL also provides many built-in tools for visualization of

3D data, including both iso-surfacing and volume rendering. The iso-surfacing tools include a shaded volume tool and an interactive GUI for extracting slices. The volume rendering tools use voxels (pixels with opacity). Several voxel rendering methods are provided in IDL, including control over lighting, interpolation and quality options.

One of the features of IDL that makes it so flexible and useful is the ability to extend its capability with custom code, written in a high-performance compiled language like C using dynamically loadable modules (DLM). A dynamically loadable module is code that has been compiled into a shared object. A DLM can be loaded automatically upon starting IDL and therefore appears much more transparent and integrated to the user. This capability will be used in the project to wrap the C++ HDF5 client of the Grid Service and effectively build an IDL Grid Service client.

AVS/Express

AVS/Express [11] is a visual and command-line object-oriented development tool that enables building reusable objects, application components, and sophisticated data visualization applications. The visual development environ-

ment of AVS/Express is the Network Editor. It is used to connect, define, assemble, and manipulate objects through mouse-driven operations. AVS/Express provides hundreds of predefined objects to process, display, and manipulate data. The objects and application components connected and assembled in the Network Editor control how data is processed and how it is displayed.

AVS/Express extends the obvious advantages of simple visualizations by making it easy to develop sophisticated three-dimensional representations of data. It is more accurate to model 3D objects in a 3D view. Actual manipulation of 3D objects, for example rotating of a model, is clearly more intuitively performed in 3D space.

Like IDL, AVS/Express is extensible by adding new modules. A module is an AVS/Express object that has parameters and methods. Each method corresponds to a C, C++, or FORTRAN subroutine that is called when the object is created or destroyed or a parameter's value changes. Creating a new module with C++ methods of the C++ client proxies will be our approach to creating an AVS client module. This module will allow users to request and query

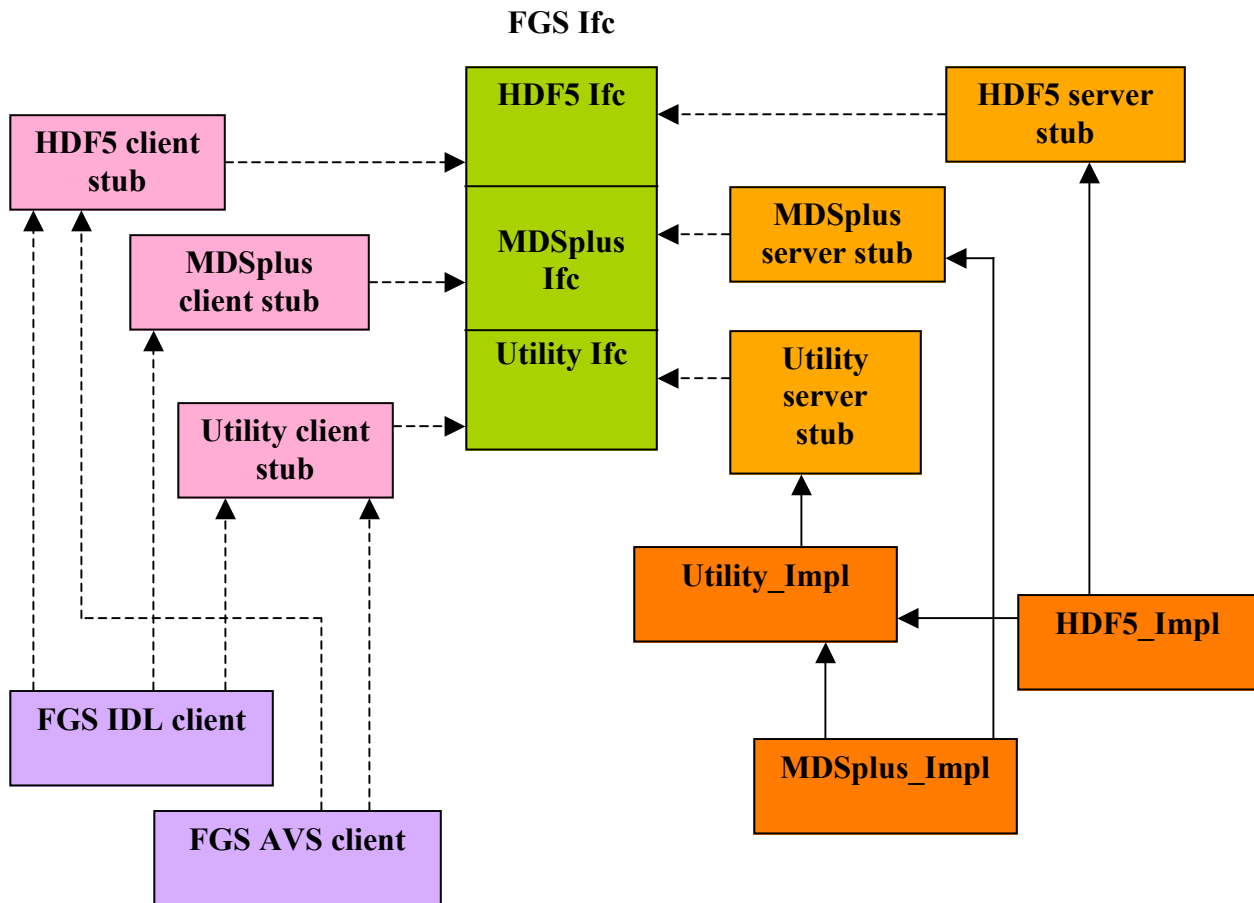


Figure 1. The Fusion Grid Service (FGS) Architecture. Dashed arrows mean “generated from”, solid arrow mean “derived from.” “Ifc” means interface and “Impl” means the implementation objects (servants).

remote HDF5 data and use them within local AVS/Express visualizations.

3. Architecture of the Fusion Grid Service

The fully developed Fusion Grid Service will allow users to access and transfer data from HDF5 files and MDSplus trees and use them on the client side in mixed applications. Web Services will be used to send commands from the client to the server, while the data can be transferred via alternative mechanisms. The transfers of HDF5 data can be performed via CORBA, GridFTP, or Web Services using binary SOAP encoding, and the transfer of MDSplus data can be done via three other protocols: MDSip, or GridFTP, depending on the size of the data and preferences of users. MDSip is “native” to MDSplus, while GridFTP will be integrated with this application.

In accordance with this plan, the WSDL interface of the service will then consists of three types of ports. The first one will deal with the get/query methods of the HDF5 API, while the second one will describe the get/query part of the MDSplus API using MDSip, GridFTP or Binary SOAP. A separate Utility interface will contain methods to deal with querying the status of jobs, canceling jobs, changing network protocols, requesting data cached on the server and choosing a particular pre-fetching scheme. Specifically,

- The HDF5 WSDL will mirror part of the HDF5 Lite API [34] methods that deal with the reading of specific datasets and the querying of dataset attributes. In addition, we will also provide high-level methods for reading hyperslabs [35] and slices (slabs of thickness 1) that are not included in HDF5 Lite, and define them in the HDF5 WSDL.
- The MDSplus WSDL document will contain methods that need a different implementation from the one provided by MDSplus. An example of this is the `gft_mdsvalue` function, which provides the same functionality as the local version of `mdsvalue` and uses GridFTP/Binary SOAP to send the data to the client.
- A Utility WSDL interface will define common operations such as querying the status of jobs, canceling jobs, changing the default network protocols, requesting data buffered on the server and pre-fetching data.

The development of the Fusion Grid Service (see Figure 1) will start with creating these WSDL documents. Client and server stubs will be generated from the WSDL specifications. Integrating with the Globus runtime relies on the OGSA team implementing the tools for generating C++ bindings. The OGSA team has partially completed this work (generation of C client stubs), but has not fully provided the C/C++ hosting environment yet. Therefore, we are currently conducting our tests and prototyping using the gSOAP Web service toolkit.

The implementation objects (`Utility_Impl`, `HDF5_Impl` and `MDSplus_Impl`) will derive from the server stubs generated from the WSDL documents described above. Both `MDSplus_Impl` and `HDF5_Impl` will derive from `Utility_Impl` class and inherit functions for general tasks and override them as necessary. For example, they will be able to pre-fetch data using a simple pre-fetching of data (specialized to be signals in `MDSplus_Impl` and hyperslabs of thickness one, or *slices*, in `HDF5_Impl`). The mode of selection of the data to pre-fetch will be specified by the user and based on the currently referenced data. There will be several schemes from which to choose. These schemes will be specified in a suite of tests, determining the high access probability data, or *neighborhood*. The neighborhood is computed using the previous access patterns for the specific dataset and slices, known characteristics of data (e.g., dimensionality), and general behavior/access patterns (for example, if a user asks for some coordinate E_x , she will typically also ask for corresponding E_y and E_z). Use of pre-fetching will significantly speed up the data analysis and visualization and reduce the client latency.

While implementation of the `HDF5_Impl` class is straightforward (simply delegating all actions to existing HDF5 Lite API), the `MDSplus_Impl` class will be implemented in such a way that the default transfer mechanism (MDSip) could be overwritten. The constructor of the server object will use system calls to start MDSip with a particular local configuration file. In addition, the `MDSplus_Impl` object will have a method for getting and evaluating data and sending it via alternative methods, such as Binary SOAP, DIME, GridFTP or CORBA, underneath. This implementation will first delegate implementation to the native `mdsvalue` function and then use the alternative methods to put the extracted data to the client machine. Finally, in order to provide users a means to import either type of data into either type of application, we will eventually need to implement an importer of MDSplus data into HDF5 files and back.

All client stubs will further be wrapped into a single IDL client module, so that HDF5 and MDSplus remote data could be conveniently requested from IDL data analysis and visualization applications. Two client stubs (HDF5 and Utility) will also be wrapped into an AVS/Express module, so that HDF5 remote data could be called directly from AVS/Express applications. Because MDSplus users rarely use AVS/Express, we will not consider implementing an AVS/Express wrapper for the MDSplus client.

4. Performance Tests

To provide a robust implementation of the Fusion Grid Service and to minimize the latency of data retrieval, we must be able to select appropriate data transport mechanisms that

are most efficient for the specific run-time environment. For this purpose we have benchmarked several data transfer mechanisms. The tests are devised from typical usage scenarios, all assuming that the remote data is located on a well-connected host with large storage capacity serving but having different client bandwidths. The three connection setups are identified as the following:

1. **LAN:** the server and the client are connected via a 100Base-T switch. For these tests we used two computers in Tech-X connected via a 100Mbps Ethernet switch.
2. **ESnet:** the server and the client are connected via a high speed wide-area network, such as the ESnet [36] sponsored by the US Department of Energy. For this configuration, we had a server at General Atomic and a client at the Princeton Plasma Physics Laboratory. These institutions connect to the ESnet via an OC-3 (155 Mbps) connection. However, since all these computers use 100base-T switches, the connection bandwidth cannot exceed 100Mbps.
3. **WAN:** the server and the client are connected via the “regular” Internet backbone. We performed this test using a server at General Atomic and a client at an internal machine at Tech-X Corp. Tech-X has an external connection to the Internet through a T1 (1.54 Mbps) connection.

Table 1: Characteristics of Experiment Network Connections

Connection Scenarios	Max. Measured TCP Bandwidth (Mbytes/sec)	Network Latency (msec)	Bandwidth Delay Product
LAN	11.2	0.13	1.6 Kbytes
ESnet	0.69	83	1 Mbytes
WAN	0.17	112	21 Kbytes

Table 1 summarizes the characteristics of all the experiment connections. The maximum measured TCP bandwidths in Mbytes/sec using iperf [37] for the connection scenarios in our benchmark tests. As shown in the table, the measured bandwidths for both LAN and WAN matches the bandwidth of the slowest links in respective connection setups. However, the measured bandwidth for the ESnet is significantly slower than the minimum bandwidth of 100Mbps because of the high round-trip latency (~850ms) we observed. This is caused by the high-latency characteristic of ESnet. The TCP window size can impose significant effect on the performance of the data transfer.

All tests programs are compiled with GNU g++ v3.3 on x86 Linux using the highest optimization level (-O3) without debug information. The benchmark test is designed to accurately measure the performance of using four transport mechanisms, according 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. Implicitly or explicitly allocate dynamic memory for receiving dataset.
4. Retrieve the data into the client memory.
5. Disconnect from server.
6. Use the retrieved data (not timed).
7. Release the memory.
8. Optional writing data into local files (not timed).

For the obvious reasons our timing programs do not include step 6. We also did not time writing into files (step 8), since this step is optional and is assumed to take approximately the same time for all the mechanisms. This means that we measured only “file-to-memory” performances.

The four data retrieving mechanisms we have tested so far use a mixture of data storage formats and data transport mechanisms. This mixture is specific for the fusion simulations and experiments. The first mechanism is MDSplus version 1.5-0, as described in Section 0, which is a remote data distribution mechanism and data format in its own right. We use the server program, MDSip, that comes with the package and implement our own client benchmarking program using MDSplus’ C interface. The client program allows users to specify the tree, the shot number and the name of the signals it should retrieve and repeats retrieving the signal data using the defined steps for specified iterations.

The remaining data retrieval mechanisms allow retrieving of HDF5 datasets remotely using various data transport mechanisms and different optimization methods. Regardless of the actual mechanism used, the remote data retrieval service models the remote operations following the style of HDF5 high-level API. They allow clients to identify the target dataset by specifying the HDF5 file name and the full pathname of the dataset in the data hierarchy. Two operations are provided to query the metadata of the dataset and to retrieve the dataset. The client side benchmarking program uses the same steps as that of MDSplus benchmarking client. Two different transporting mechanisms were tested for remote HDF5 service, namely, Web Service using gSOAP 2.3.1 and CORBA using TAO 1.3.

However, for Web Service implementations, plain SOAP using text-based XML encoding is not efficient enough for transmitting large scientific data [38]. In order to avoid the unnecessary overhead of transmitting large datasets in SOAP encoding, we adopted two optimization methods in the distributed HDF5 service. The first optimization adopted by

both Web Service implementations is to return requested dataset as raw binary data using Base64 encoding. Note, however, that transmitting data in binary form implies the application is taking up the responsibility to ensure the data is in a format that is understandable by the client. Our HDF5 server implementations take advantage of HDF5 APIs to extract data directly in the target machine's format – specifically, byte ordering before transmitting the binary data over to the client side. This alleviates us from having to implement our own custom data marshaling routines.

This optimization is applied in both Web Service-based implementations, where the data retrieval operation always returns the retrieved array as a block of binary data using the type “xsd:base64Binary”. We call the two implementations “binary SOAP” and “binary SOAP-DIME” respectively. As their names imply, “binary SOAP” returns the retrieved data as part of the response envelop, while “binary SOAP-DIME” returns the retrieved data as a WS-attachment using DIME.

For data retrieval service using CORBA, we implement the operation as defined in the following definition:

```
typedef sequence<octet> OctetSeq;
OctetSeq getDataset (...);
```

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.

As previously mentioned in this section, the high latency of the ESnet is a significant factor in determining the throughput of pumping data through the connection which was observed in our earlier experiments. The experiments conducted in this test, therefore, make sure that all the test programs that we have control of use the same TCP window size of 64 Kbytes. Especially, gSOAP uses a window size of 32 K by default and we had to modify the source code and recompile the library to adjust the window size. The CORBA implementation, TAO, used in the experiments, however, provides command line switches to adjust the window size.

We compared the performance of both extracting a set of datasets in various sizes (ranging from 0.5 MB to 19 MB) remotely using MDSplus and our distributed HDF5 services, including binary SOAP, binary SOAP-DIME, and CORBA, over different network connections. We collected a set of HDF5 files from coming from the VORPAL and M3D fusion codes. We then converted these data files into similarly structured MDSplus format using a converter developed by Thomas Fredian from MIT. This approach ensured that our benchmark results compare data not only of the same size but also from the similar hierarchical structures.

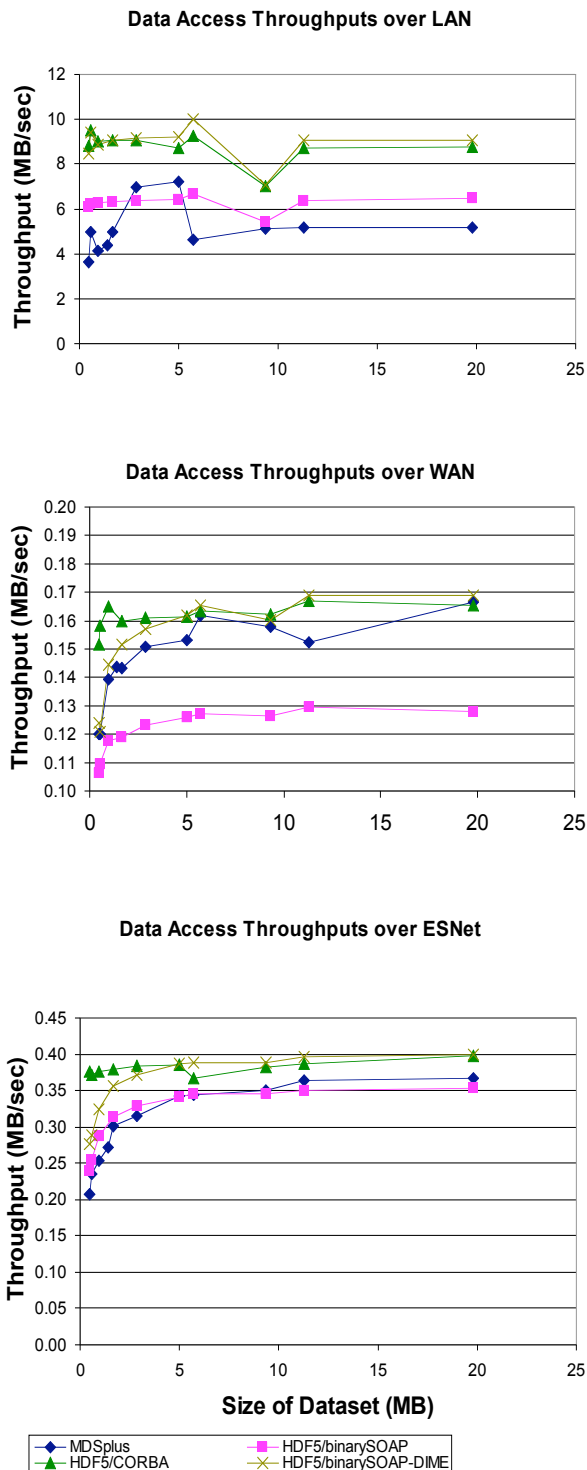


Figure 2. Throughput versus the size of the transferred data in LAN, WAN and ESnet configurations.

Figure 2 shows the comparison results of running the tests over the LAN, ESnet, and WAN configurations. Comparing the results between MDSplus and HDF5/binarySOAP reveals that an increase in bandwidth can make the overhead in transfer mechanisms more significant. This is because when running over a higher bandwidth connection, much more data could have been transmitted than when running over a lower bandwidth network during the same amount of idle time. This can be seen from the fact that MDSplus provides 10~25% better throughput than does HDF5/binarySOAP when running over WAN. The performance difference was almost reversed when the test was performed over LAN. This seem to imply that MDSplus imposes a bit higher mechanical overhead, such as file access, whereas HDF5/binarySOAP imposes more protocol overheads, such as the 33% payload overhead caused by the Base64 binary encoding.

The figures also show that CORBA and binary SOAP-DIME both provide throughputs close to bandwidth capacity in all cases. In the case of transmitting the data using CORBA, the performance gain can be contributed to both the mature middleware specification and the near 0 overhead for encoding binary data in CORBA. On the other hand, comparing to sending large data as Base64 binary in Web Service, using DIME attachments circumvents not only the 33% increase in payload but also the need to parse over the large block of data that is required when returning the data in the response envelope.

It is also obvious from observing the results that the location of a dataset inside the file hierarchy affects the time required to locate and retrieve the data from a file. This is especially obvious when running the test over LAN configuration where there is a big decrease in throughput of various HDF5 tests. Likewise, throughput results for MDSplus show much more fluctuation when the test was run over LAN configuration.

5. Concluding Remarks and Future Directions

This paper outlines the challenges facing the fusion and plasma community in accessing and sharing experimental and simulation data. It also presents the Fusion Data Grid service framework under development at Tech-X Corporation for addressing these challenges. Our initial performance tests show that sending bulk datasets using CORBA or as DIME attachments in binary SOAP encoding yield better performances in most configurations.

Other future performance tests will look into improving data transfer performance by establishing multiple parallel data streams over the network [14, 39]. This approach has limited applicability over less than reliable networks as the exponential-back off algorithm employed by TCP may cause

these streams to run out of sync and degrade the throughput even more than using a single data stream. When running over a more reliable network connection, such as ESnet, and over conventional routers that treat all data streams equally, we can more easily harvest the full bandwidth of the network with multiple streams. We plan to integrate these multi-stream data transport mechanisms, such as GridFTP, into our implementations and tests in the future so that the service could intelligently switch to the most appropriate protocol depending on the size and type of data.

The proposed Grid Service should allow for switching data transfer mechanisms on the fly, depending on the type, size and location of data. There are two candidates for a solution that might provide this capability and which we are planning to evaluate: the Proteus and the Globus XIO libraries. The Proteus library [40], under development at Indiana University, allows for integrating multiple message protocols within one system. Proteus decouples application code from protocols at run-time, thus allowing clients to incorporate new protocols without recompiling. The current Proteus implementation has experimented with SOAP and JMS support. We will need to extend it to include GridFTP, and MDSip – the protocols we intend to use for different types and sizes of data.

Globus XIO, under development at Argonne National Laboratory, is a library with a uniform API allowing for swappable Grid I/O implementations. Although one needs to compile the application using XIO with all transfer drivers included, XIO promises an easily extensible and well-documented solution for bringing multiple protocols into the Fusion Grid Service.

ACKNOWLEDGMENT

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

REFERENCES

- [1] *Vorpal: a Versatile Plasma Simulation Code*, [Online]. Available: <http://www-beams.colorado.edu/vorpal/>.
- [2] *Non-Ideal MHD with Radiation Open Discussion (NIMROD) Project*, [Online]. Available: <http://www.nimrodteam.org/>.
- [3] M3D, [Online]. Available: <http://w3.pppl.gov/~jchen/>.
- [4] M. Ono, *et al.*, Nucl. Fusion **40**, 557 (2000).
- [5] D. P. Schissel, *et al.*, Phys. Fluids **31**, 3738 (1988).
- [6] *The Alcator C-Mod Tokamak Fusion Research Project*, [Online]. Available: <http://www.psf.mit.edu/cmof/>.
- [7] Strachan, J.D., *et al.*, Phys. Rev. Lett. **72** 3526 (1994).
- [8] <http://www.iter.org/>.
- [9] T. Fredian and J. Stillerman, *MDSplus Remote Collaboration Support – Internet and the World Wide Web*, Fusion Engineering and Design **43**, (1999).
- [10] <http://hdf.ncsa.uiuc.edu/HDF5/>.

- [11] <http://www.av5.com/>.
- [12] <http://www.rsinc.com/idl/>.
- [13] <http://www.opendx.org/index2.php>.
- [14] <http://software.sci.utah.edu/scirun.html>.
- [15] I. Foster, C. Kesselman, editors, *The Grid: Blueprint for a Future Computing Infrastructure*, (1999).
- [16] <http://www.globus.org/>.
- [17] Global Grid Forum, *Open Grid Service Infrastructure (OGSI), Version 1.0 draft*, [Online]. Available: http://www.gridforum.org/ogsiwg/drafts/draft-ggf-ogsi-gridservice-29_2003-04-05.pdf.
- [18] *The GridFTP protocol and software*, [Online]. Available: <http://www.globus.org/datagrid/gridftp.html>.
- [19] http://gotdotnet.com/team/xml_wsspecs/dime/WSDL-Extension-for-DIME.htm.
- [20] <http://www.omg.org/corba/>.
- [21] <http://www.w3.org/TR/SOAP/>.
- [22] <http://www.w3.org/TR/wsdl>.
- [23] <http://www-unix.globus.org/developer/xio/>.
- [24] 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.
- [25] Robert A. van Engelen, "Pushing the SOAP Envelope with Web Services for Scientific Computing," *the proceedings of the International Conference on Web Services (ICWS)*, 2003.
- [26] <http://w3.pppl.gov/transp/>.
- [27] <http://web.gat.com/comp/analysis/uwpc/docs/reviewplus.shtml>.
- [28] <http://web.gat.com/pubs-ext/MISCONF99/A23153.pdf>.
- [29] http://hdf.ncsa.uiuc.edu/HDF5/hdf5_hl/.
- [30] <http://www.unidata.ucar.edu/packages/netcdf/>.
- [31] W. W. Lee, *Gyrokinetic approach in particle simulation*, Phys. Fluid 26, 556 (1983).
- [32] Michi Henning and Steve Vinoski, *Advanced CORBA Programming in C++*, Addison-Wesley, Reading, MA, 1999.
- [33] <http://www.cs.wustl.edu/~schmidt/TAO.html>.
- [34] http://hdf.ncsa.uiuc.edu/HDF5/hdf5_hl/doc/RM_hdf5hl.html.
- [35] http://hdf.ncsa.uiuc.edu/HDF5/doc/Tutor/examples/C/h5_hyperslab.c.
- [36] <http://www.es.net/>
- [37] Y. Thomas Hou, Yingfei Dong, and Zhi-Li Zhang, "Network Performance Measurement and Analysis Part 1: A Server-Based Measurement Infrastructure" [online]. Available: <http://www-users.cs.umn.edu/~dong/papers/framework.ps>.
- [38] K. Chiu, M. Govindaraju, and R. Bramley. "Investigating the Limits of SOAP Performance for Scientific Computing",. In *Proceedings of the Eleventh IEEE International Symposium on High Performance Distributed Computing (HPDC'02)*, July 2002.
- [39] S. Klasky, S. Ethier, Z. Lin, K. Martins, D. McCune, and R. Samtaney, "Grid -Based Parallel Data Streaming implemented for the Gyrokinetic Toroidal Code", in *Proceedings of the 2003 Conference on Supercomputing*, Phoenix, AZ, November 2003.
- [40] K. Chiu, M. Govindaraju, and D. Gannon. "The Proteus Multiprotocol Library", in *Proceedings of the 2002 Conference on Supercomputing*, Baltimore, MD, November 2002.