PSFC/JA-13-22

Input/Output Plugin Architecture for MDSplus

Stillerman, J.A., Fredian, T.W., Manduchi, G.*

* Consorzio RFX, Euratom-ENEA Association,
Corso Stati Uniti 4, Padova 35127, Italy

June, 2013

Plasma Science and Fusion Center
Massachusetts Institute of Technology
Cambridge MA 02139 USA

This work was supported by the U.S. Department of Energy, Grant No. DE-FC02-99ER54512. Reproduction, translation, publication, use and disposal, in whole or in part, by or for the United States government is permitted.

Input/Output Plugin Architecture for MDSplus

Joshua Stillerman[1], Thomas Fredian[1], Gabriele Manduchi[2]

[1] *Massachusetts Institute of Technology, 175 Albany Street, Cambridge, MA 02139, USA*
jas@psfc.mit.edu, twf@psfc.mit.edu

[2] *Consorzio RFX, Euratom-ENEA Association, Corso Stati Uniti 4, Padova 35127, Italy,*
mailto:gabriele.manduchi@igi.cnr.it

The first version of MDSplus was released in 1991 for VAX/VMS. Since that time the underlying file formats have remained constant. The software however has evolved, it was ported to unix, linux, Windows, and Macintosh. In 1997 a TCP based protocol, mdsip, was added to provide network access to MDSplus data. In 2011 a mechanism was added to allow protocol plugins to permit the use of other transport mechanisms such as ssh to access data. users. This paper describes a similar design which permits the insertion of plugins to handle the reading and writing of MDSplus data at the data storage level. Tree paths become URIs which specify the protocol, host, and protocol specific information. The protocol is provided by a dynamically activated shared library that can provide any consistent subset of the data store access API, treeshr. The existing low level network protocol called mdsip, is activated by defining tree paths like "host::/directory". Using the new plugin mechanism this is re-implemented as an instance of the general plugin that replaces the low level treeshr input / output routines. It is specified by using a path like "mdsip://host/directory".

This architecture will make it possible to adapt the MDSplus data organization and analysis tools to other underlying data storage. The first new application of this, after the existing network protocol is implemented, will be a plugin based on a key value store. Key value stores, can provide inexpensive scalable, redundant data storage. An example of this might be an Amazon G3 plugin which would let you specify a tree path such as "AG3://container" to access MDSplus data stored in the cloud.

1. Introduction

MDSplus[1] is a suite of tools for data acquisition, data management, and remote data access. It was developed by a collaboration between MIT/PSFC, IGI/RFX, and LANL/Zth. Work began in 1988 and it has been in use at fusion experiments around the world since 1991. Currently more than forty sites use some or all of the tools provided. At its core are a hierarchical record store and a vector/matrix interpreter. Data access, both local and remote can be achieved through the evaluation of expressions in this interpreter that, in turn, refer to records accessed by functions in a library called *treeshr*.

This interpreter based data access defines MDSplus as 'data as a service'. The TDI language serves both to specify the contents of records and to specify requests for data from the system. For example in Alcator C-Mod plasma current is defined as:

```
    Build_With_Units(
        \MAG_ROGOWSKI.SIGNALS:ROG_FG + 2100F0 * \BTOR,
        "ampere")
```

and could be referenced as:

```
    \IP
```

or

```
    \IP*-1E-6.
```

These expressions can be evaluated either locally on the user's computer, or remotely by a data server. In addition the records they refer to: \IP, \MAG_ROGOWSKI.SIGNALS:ROG_FG and \BTOR can be read from the record store either locally or over the network.

Over its 20 year history the underlying file format is essentially unchanged. The software has been ported across platforms and computer languages, never the less the original data files from 1991 can still be read, and updated. Over the years we have added new record types to implement new features such as segmented records and extended node attributes, and to relax record length restrictions imposed by earlier operating systems. We have also added new APIs that expose the inherently object oriented nature of the system. When the project was started, object oriented compilers and tools were not mature enough for production software. Self descriptive data, hierarchical data classes, introspection and dynamic image activation/call by name, were all implemented in traditional languages.

MDSplus data transport has evolved over time. The system started with local file access only. Later, network protocols at three different levels were added. All of these are still in use.

The system originally relied on the VMS shared cluster file system for storage, concurrency and transport. In 1996 mdsip thin client access was added. This allowed remote users to request data servers to evaluate expressions on their behalfs. The servers received expressions from network clients, and did local input/output to read records, and evaluated the expressions returning answers over the network.

In 1998 a thick client protocol was added. In this scenario, clients request record level input/output, which the server performs on their behalf. The client then uses local compute resources to evaluate any expressions involved in the request. The server's tree definitions are used, and the server assembles multi-part records for the client.

Distributed trees were added to MDSplus in 2002. Trees are defined as a list of possible specifications, each of which has an optional host followed by a partial file specification to locate the files. If the host is provided, the mdsip network protocol is used to request that record level input and output is performed on behalf of the client. All data decompression, record assembly and expression evaluation is done by the client computer.

For 20 years MDSplus relied on these mechanisms to provide data access. Pluggable transports for for mdsip were recently added, so that the above scenarios could be done over the user's choice of network protocols. Instead of communications between client and server using simple TCP packets, plugin capability allows developers to add new protocols to be used in the client/server communications. Today, there are plugins implemented to enable the use of protocols such as ssh, http and gsi (Globus Security Infrastructure).

2. Motivation

Data storage is undergoing a revolution in recent years. There are new storage technologies, and new APIs. Performance constraints are decreasing in importance. Site specific policies and preferences need to be taken into account. In addition, we would like to have a convenient way to benchmark new I/O and network caching schemes.

In recent years, there has been an explosion of data, and commensurate technologies to deal with it. Starting with google indexing, and caching the entire crawlable web. People have begun to leverage large amounts of social media, and marketing data. Underpinning these applications are powerful distributed key value stores[2][3] and NoSQL databases. These modern systems have not been employed for science data in general, and fusion energy data in specific. This project provides a framework for experimenting with these systems.

When MDSplus was first developed, fusion experiments were producing on the order of 1 MegaByte of data per shot, and computers and disks were large expensive objects in air conditioned rooms. This data size has been doubling every two years, and at Alcator C-Mod shots from the FY12 campaign exceeded 15 Gigabytes in size (See Figure 1). The initial design and implementation decisions were made in a compute and I/O constrained environment. Today, computers and storage are inexpensive and fast, and it is now feasible to store redundant copies of data, to index it with strings, and to create local data caches.

Some existing fusion experiments have local constraints on their underlying data stores. These may be motivated by the existence of data analysis applications, large amounts of existing data, contractual obligations, or simply site preferences. In the past MDSplus has accommodated them in one of two ways. In some cases (DIIID, IPP, Tore Supra) an MDSplus thin client server is run at the site, evaluating expressions that access the local data store.

When one site (JET) wanted to use the MDSplus hierarchical view to their existing data they replaced the treeshr API with a custom implementation that provided a bridge between their storage format and MDSplus.

This project provides a platform to compare new data storage schemes under the MDSplus framework. Different approaches can be easily benchmarked against each other. Local data systems and preferences can be easily accommodated.

3. Implementation

This project involved extending the current treepath specifications to optionally include a protocol followed by a protocol specific string. This extends the originally available local partial file specifications, remote partial file specifications, and thick client hosts. Trees are defined by creating an environment variables (treename_path) containing a semicolon separated list of ways to look for the files which hold it. Treeshr looks for files that it is trying to access in each of the elements in this path until it either succeeds or exhausts the elements of the path. There is special '~' syntax in the filename which can be used to substitute digits of the shot number or the name of the tree into the partial file specification. For example, at C-Mod the cmod tree (cmod_path) is defined as:

```
alcdata-test::/cmod/trees/test/~t/;
alcdata-new::/cmod/trees/new/~t/;
alcdata-models::/cmod/trees/models/~t/;
alcdata-archives::/cmod/trees/archives/~i~h/~g~f/~e~d/~t;
alcdata-saved::/cmod/trees/saved/~t/
```

When a tree is opened each of these servers are asked in turn if they can find its files using the mdsip protocol. Note the ~t is replaced with the tree name, and the ~d - ~I are replaced with digits of the shot number automatically in each of the above clauses. The new system introduces a more general url syntax for the clauses. A protocol name, followed by a colon followed by two forward slashes and then a protocol specific string. For example:

```
mdsip://alcata.psfc.mit.edu/cmod/trees/test/~t/
```

specifies the mdsip protocol, which in turn parses the remainder of the string for the host and partial file specification.

Each protocol is implemented by a shared library with the same name as the protocol. This library must have an entry point named 'load_protocol', that fills in a TREE_PROTOCOL structure defined in treeshr's private include, with the addresses of the routines that this protocol implements. Throughout treeshr calls to any routines that might be redefined by a plugin, are called using macros which conditionally call the native treeshr versions or the plugin defined ones. Each plugin impliments a subset of the potentially redefinable entry points.

Each plugin can replace treeshr routines at different levels of its call stack. Some like the mdsip plugin replace the atomic file input / outout routines. Others like the mongodb one, which is under development, replace higher level treeshr calls such as TreeOpen, TreeGetRecord, or TreeGetNci.   Mongodb is an open source NOSQL database. It provides tables with heterogenous rows, and includes replication, sharding and network access[4]. The requirement is that they implement a functionally complete set of routines..

This project required much of the code in treeshr to be refactored. Tree paths, and parts of tree paths were parsed in several different routines, extracting information from them incrementally as required. For this scheme to work, the tree paths had to be broken into terms, very early, so that the needed plugins could be loaded before the entry points are called. Parsing the path's terms is relatively inexpensive compared to loading the plugins. In order to load a plug in the operating system has to activate the image, fill in the addresses of all of the defined entry points, and allocate any plugin specific data structures. This is only done on an as needed basis. Treeshr loads plugins in order until the IO or entry point request can be successfully executed. Subsequent calls for this tree/shot must be satisfied by that plugin's routines, and later plugins in the tree's path are not loaded.

4. Current Status

Work on this project is ongoing. The existing mdsip code that implements distributed trees, has been repackaged as a treeshr input / output plugin called mdsip. The mdsip plugin and its url syntax can be used interchangeably with the original distributed trees implementation and tree path syntaxes. This plugin is a file input/output level package. Standard posix file operations are replaced with network requests for remote file operations. As described in section 3, this involved significant re-engineering of treeshr. Other file level plugins can now be easily produced using this as a template. All of the mechanics of parsing the tree paths and dynamically activating the code have been implemented.

Work has begun on a mongodb (nosql) plugin. Rather than doing this at the same level as the mdsip plugin, this is being written to intercept treeshr calls higher up in the call stack. Since the database provides higher level functions than raw file input / output, it makes sense to leverage these capabilities. In addition this choice provides a good test case for the underlying idea that plugins can be provided at various levels of the API. This plug in has been started, but is still under development. Writing it has exposed some unanticipated complexities.

5. Next Steps

Work on treeshr plugins is continuing. This work will take one of two directions. Original plans called for completion and testing of plugins for mongodb, local file input / output, and mdsip thick client. A new idea, which we intend to explore first, is to create a generic python plug in, which would allow a python module which implements the input / output functionality to be loaded. This would greatly ease future plugin development. Many databases and key-value stores have very good python interfaces. For some, like mongodb, the python interface is superior to the interfaces in compliled languages. The tree path clauses would specify the python module to be loaded and any additional information needed by these routines to access the tree records. For example:

```
    pyplug://pymongo:ip.address.of.database/collection-name
```

would activate the generic python plugin, tell it to load the pymongo.py module. The rest of the string would be used by the python code to access the database.

These approaches will be pursued in order to have examples of a variety of plugins, both at the file and at the record input / output layers of treeshr. These can then be used for testing and benchmarking. Further, we will be able to completely remove the input / output routines from treeshr, which should significantly simplify it.

6. Conclusions

Several conclusions can be drawn from this ongoing project. The original treeshr code was not designed to accommodate different input and output schemes. They were written over a long period of time and spread throughout the code. Consolidating them, and removing all of the conditional compilation greatly simplifies things.

Plugins provide an easy mechanism for testing and benchmarking new input / output schemes.

Sites with proprietary, or site specific input / output needs will have a documented method to layer them under MDSplus. Site specific code can be called without recompiling or relinking any of the distributed MDSplus software.

6. References

[1]Stillerman, J.A.; Fredian, T.W.; Klare, K.A.; Manduchi, G., "MDSplus data acquisition system," Review of Scientific Instruments , vol.68, no.1, pp.939,942, Jan 1997
[2]Rick Cattell. 2011. Scalable SQL and NoSQL data stores. SIGMOD Rec. 39, 4 (May 2011), 12-27
[3]Avinash Lakshman and Prashant Malik. 2010. Cassandra: a decentralized structured storage system. SIGOPS Oper. Syst. Rev. 44, 2 (April 2010), 35-40
[4]MongoDB.
http://www.mongodb.org/, Retrieved September 2012.