WILLIAM ALLCOCK AND JOHN BRESNAHAN

The Globus™ eXtensible Input/Output (XIO) System

A New I/O System: So What?

I was once told that everything new should be put to the "who cares" test. This statement is especially true for software components.

The Globus Toolkit™ (GT) includes a new extensible I/O library called Globus XIO. So, who cares? Chances are if you have a mature code base using only run-of-the-mill I/O characteristics in a static environment, then you don't care. If, however, you are creating applications that need high-performance I/O leveraging the advances in network infrastructure and protocols, or if you want to access nonstandard storage systems or a range of storage systems or applications that need to interact with the volatile service architecture on today's clusters, then you should care.

Why? Because the Globus eXtensible Input Output system offers many features that can help in such tasks. To computer scientists, Globus XIO is an API that provides an abstraction layer to transport protocols. To application developers, it is an API that enables different I/O problems to be presented uniformly as a simple open/close/read/write (OCRW) interface. To network protocol developers, it is a support framework for developing communication protocols. To mass storage vendors, it is an interface that enables an existing application written with XIO to access their hardware.

So, is Globus XIO all things to all people or the solution to all I/O problems in cluster computing? The answer is obviously no. Globus XIO does not try to solve every I/O problem by mapping every type of I/O to a single API; that is clearly not possible. Nor does Globus XIO attempt to hide all details of the underlying protocol; that is clearly undesirable. While it is nice to think that all I/O can be lumped into a single abstraction, protocols are different for a reason. They solve different problems and do different jobs. Often a user must be able to control these differences to properly take full advantage of the I/O protocol involved.

Globus XIO has two main objectives. The first is to present a simple OCRW API to all *byte stream* I/O. In other words, whether you are reading from a file or a network or an electron microscope, whether it is running TCP, UDP, or some custom protocol, to the application it should simply be OCRW. We can't hide all the differences, but we can minimize the impact on the application code and isolate the differences. The second objective is to minimize the development time for creating and prototyping new I/O protocols and new device interfaces. This article focuses on the first objective.

An Example

It is amazing — or frustrating, depending on your point of view — how many I/O problems appear to be solvable with OCRW functionality. The devil is in the details, however, and the solutions to these problems are just different enough to result in a variety of API models with a variety of semantics. From a high-level perspective, perhaps they all look alike. But to the developer who has just learned to use an asynchronous push model API to send messages via FTP, the synchronous multithreaded API for HTTP messaging that must be integrated into an application is quite different.

A common usage scenario in the Grid involves obtaining data from a remote source (a remote GridFTP server, a mass storage system, a specialized instrument such as an electron microscope, or perhaps the output of a simulation program), performing a computation on that data, and then transporting the results to another remote site (perhaps a proprietary display, another mass storage device, or another computation).

Our example has multiple I/O types: I/O written to the network protocol API; I/O written to the proprietary API of the remote data source(s); posix I/O to the local disk; if the analysis codes are interactive, I/O with the user; and I/O to the visualization system. In the top of *Figure One* is a graphical representation of these APIs. Each can have different programming semantics: threaded vs. non-threaded, synchronous vs. asynchronous, different error handling systems, and so forth.

At the bottom of *Figure One* is a representation of Globus XIO. Now the differences are all hidden from the application. There is a uniform programming model, so the application developer can design a clean logical flow. Furthermore, if at a later date the application needs to be aware of a new protocol, little to no new development is required. The application can simply select a new driver stack for data transfer. These advantages can dramatically reduce development time.

Situations do arise when an application needs to interact directly with the driver, typically in the case of optimization. For example, in TCP an application may wish to set the

buffer size for performance reasons. While Globus XIO is an abstraction layer, it does provide a means to interact directly with a driver when needed. The code to do this is, of course, driver specific. In other words, the code required to alter the TCP buffer size will work only with the TCP driver. Most often, such cases can be taken care of when an application is initializing its data structures. The most complicated code with regard to I/O in a typical application centers on the reading and writing of data. The logic required to open streams of data is most often a single straight down thread of execution; therefore, when driver-specific code is needed, the vast majority of the advantages provided by Globus XIO can be preserved.

Earlier we said that Globus XIO is not a panacea and it is appropriate for byte stream-oriented I/O. If we examine the I/O types in our scenario, Globus XIO would likely be appropriate for the remote data accesses, the local disk I/O, and the network I/O. The interaction with the user would depend on whether it was a simple terminal/text interaction or a GUI. The former might make sense in XIO, the latter almost certainly not. The display I/O would likely not be appropriate. It is said that when all you have is a hammer, everything looks like a nail. Well, most of us have a variety of tools, and we should use the tool that is the most natural fit.

XIO and the Globus Toolkit

Globus XIO is available now in alpha version and will be released with version 3.2 of the Globus Toolkit. With that release, all C-based code in the toolkit will use Globus XIO via a globus_io wrapper. Previously, we used the Globus IO library for I/O. This library has now been reimplemented with Globus XIO underneath. The new implementation allows Globus XIO to be used without modifying the toolkit. In version 4.0 of the Globus Toolkit, we will enable GridFTP — our wide-area data movement tool and the principal driver for Globus XIO — to use Globus XIO directly. We have three primary usage scenarios in mind for Globus XIO:

1 Independence from the Transport Control Protocol

GridFTP was designed for and is often used in the highest-performance networks in the world. For instance, during the SC2003 supercomputing conference, GridFTP achieved nearly 9 gigabits per second on the NSF TeraGrid. At SC2003 we also demonstrated GridFTP running over UDT with minimal changes to the application. UDT is a transport protocol being developed at the University of Illinois-Chicago. It is based on UDP but has reliability added over UDP and backs off in the face of congestion. A number of such transport protocols are being developed for Grid environments, since the Transport Control Protocol, which has served the Internet well for many years, was not designed for the high-speed, high-latency, bulk transfers that are becoming common in Grid applications today. (As an aside, we note that use of highly aggressive protocols, such as UDT, is not recommended on the general Internet.)

2 Ease of Adding GridFTP Support to Third-Party Applications

As the use of GridFTP becomes commonplace in Grid applications, many are finding that the data they want is stored behind a GridFTP server. Many applications want to be able to read and write files stored behind such a server just as they do local files. Enter the GridFTP XIO driver. This driver speaks the GridFTP protocol and can communicate with a server, but it allows the application to use the familiar OCRW interface. Note that XIO driver will end up making calls to other (TCP and GSI) XIO drivers. This practice is acceptable and indeed encouraged.

3 Ease of Providing GridFTP Access to Data Storage

Since many applications are adding the ability to access GridFTP-enabled data stores, you may want to make your data store accessible via GridFTP. The "standard" GridFTP server XIO stack would have a posix file driver as its transport driver. With the flexible Globus XIO architecture, however, you will find it relatively easy to write a driver that accesses a proprietary storage system.

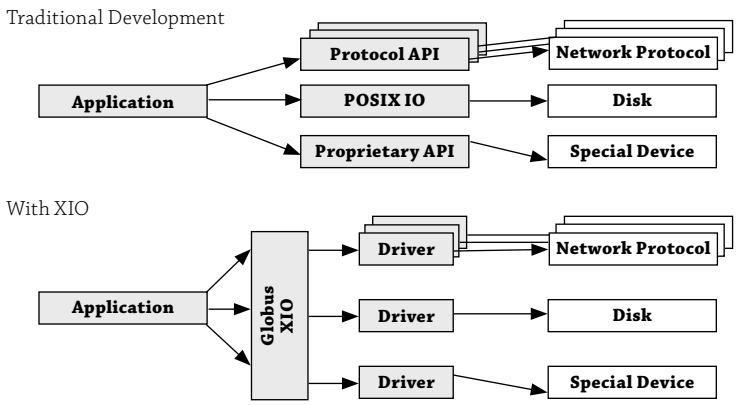

FIGURE ONE Traditional APIs and the XIO model

Technical Details

Globus XIO is a C library. Since the core API is simply OCRW, the user API is fairly straightforward. The Globus XIO Web site, shown in the resources sidebar, has complete API documentation available. We provide synchronous calls, `globus_xio_read()` and asynch calls, `globus_xio_register_read()` and have vector variants of each `globus_xio_[register]_readv()`. The best way to become familiar with Globus XIO is by looking at an example. Let's examine the case in which a user wishes to use Globus XIO to read data from a file. The code for globus_xio_example.c is available on the ClusterWorld Web site (www.clusterworld.com/code/Apro4/grid/globus_xio_example.c).

STEP 1 Activate Globus

XIO was developed by the Globus Alliance and so inherits some Globus Toolkit semantics. Accordingly, as with all Globus Toolkit programs, you must first activate the globus_module. Until then, no globus_xio function calls can be successfully executed. The module is activated with the following line:

```
globus_module_activate
  (GLOBUS_XIO_MODULE);
```

STEP 2 Load Driver

The next step is to load all the drivers needed to complete the I/O operations in which you are interested. The function `globus_xio_load_driver()` is used to load a driver. To successfully call this function, you must know the name of all the drivers you wish to load. For this example we want to load only the file io driver. The prepackaged file io drivers name is "file." This driver is loaded with the following code:

```
globus_result_t res;
globus_xio_driver_t driver;
res = globus_xio_load_
  driver(&driver, "file");
```

If upon completion of the above function call, you receive the response `res == GLOBUS_SUCCESS`, then the driver was successfully loaded and can be referenced with the variable "driver."

STEP 3 Create Stack

Once globus_xio is activated and a driver loaded, you need to build a driver stack. In our example the stack consists of only one driver, the file driver. The stack is established with the following code (building from the above code snips):

```
globus_xio_stack_t stack;
globus_xio_stack_
  init(&stack);
globus_xio_stack_push_
  driver(stack,driver);
```

STEP 4 Opening the Handle

Once the stack is created, you can open a handle to the file, in one of two ways. The first way is a passive open. An example is a TCP listener. The open is performed passively by waiting for some other entity to act on it.

The second way is an active open. An example is a TCP connect. The open is performed actively by the user, who initiates the open. Our example has an active open.

To open a handle, you must first establish a target object. The target object tells globus_xio what it wants to open by associating it with a contact string. The following code illustrates this:

```
globus_xio_target_t  target;
globus_xio_handle_t  handle;
globus_xio_target_
  init(&target,"/tmp/
  junk.txt",stack,NULL);
res = globus_xio_open
  (&handle,target,NULL);
```

STEP 5 The Payoff

Now that you have an open handle to a file, you can read or write data to it with either `globus_xio_read()` or `globus_xio_write()`. Once you are finished performing I/O operations on the handle, you should call `globus_xio_close(handle)`.

All this may seem like a bit of effort for simply reading a file. The advantages become clear, however, when you wish to swap other drivers. In the above example, it would be trivial to change the I/O operations from file I/O to TCP, HTTP, or FTP. All you would need to do is change the driver name string passed to globus_xio_load_driver() and change the contact string passed to globus_xio_target_init(). Both can be done easily at runtime, as the program globus_xio_example.c demonstrates.

Resources

Globus XIO Web Site
- www.globus.org/xio

Globus Alliance Web Site
- www.globus.org

Teragrid Website
- www.teragrid.org

UDT (formerly SABUL)
- www.evl.uic.edu/eric/atp/sabul.pdf

Bill Allcock is the Technology Coordinator for GridFTP at Argonne National Laboratory and is a member of the Globus Alliance.

John Bresnahan is a senior scientific programmer with the Mathematics and Computer Science Division at Argonne National Laboratory.