WILLIAM ALLCOCK

Programming with the GridFTP Client Library

GridFTP is a fast, efficient, secure, and robust protocol for data transfer. This protocol is in wide use in Grid applications, primarily through an implementation available in the Globus Toolkit™. The Globus Toolkit provides a server, a script-capable command-line interface client called *globus-url-copy*, and a set of development libraries. While *globus-url-copy* will meet the needs of most data movement tasks, at times custom code is the only solution. You may wish to enable your application to access remote files stored behind a GridFTP server, or you may prefer to process the data before local storage.

The Globus Toolkit provides two primary libraries for GridFTP: the control library and the client library. The control library provides very low-level primitives for command processing, parallel stream (multiple TCP streams) I/O, and so forth. This library gives you extreme flexibility, but it requires a deep understanding of the GridFTP protocol because you must implement the state machine yourself. The client library is built on top of the control library and hides this complexity. Each of the GridFTP operations (discussed below) represents a complete session. The protocol and state machine are completely hidden. Here we introduce the use of the Globus GridFTP client libraries; the control library is beyond the scope of this article.

Client Library Overview

The client library API is fairly straightforward, and the online API documentation is quite good. Check out the resources sidebar for further information. The client library API consists of seven basic categories of functions. Each of these categories is discussed below.

Activation/Deactivation: All modules in the Globus Toolkit require a call to the module activation/deactivation. This call initializes certain data structures and allocates/deallocates memory. Reference counts are kept for each activation/deactivation.

Handle Management: A handle is equivalent to a file descriptor. It is the "name" of each connection or session you establish. A handle is essentially the same as a control channel and associated data channels.

Handle Attributes: Attributes are data structures that pass information that modify the behavior of their associated object. Attributes are never directly accessed. In all cases, functions are provided that set the values or query their current values. Handle attributes are available related to channel caching and plug-ins. Channel caching refers to holding the socket connections open after an operation is complete and reusing them if possible. You can either cache all the connections or cache them on a URL-by-URL basis. Plug-ins are functions that can be called when certain predefined events occur. For instance, a debug plug-in provided with the Globus Toolkit is triggered every time a command or response is sent over the wire and prints each command or response to stdout, so that you can troubleshoot your session. Use of plug-ins is not covered in this article.

Operation Attributes: Operation attributes are similar to handle attributes, but rather than applying to an entire session, they apply only to a specific operation. Operation attributes include the file type, the transfer mode, number of streams, TCP buffer size, and security options.

FTP Operations: FTP operations can be subdivided into two subcategories: remote file system operations and data movement operations. The remote file system operations are fairly typical: You can check existence of files and directories; make a directory; remove a directory; delete a file; move a file; check the modification time, size, and checksum of a file; change permissions on a file; and get directory listings. Note that *chown* (change the file ownership) is not provided because it requires root privileges and the server runs as an unprivileged user for security reasons.

Two data movement commands are provided — put (send data to a remote host) and get (retrieve data from a remote host) — and multiple variations on those two commands. The general syntax is as follows:

```
globus_ftp_[partial|extended]_put\
  |get|third_party_transfer
```

The extended variations allow the invocation of server-side processing routines, if present. Typically, some sort of data reduction is involved, such as sub-sampling, to reduce the amount of data transmitted over the network. Although the extended variants are not often used, in certain circumstances they can be incredibly powerful. The partial variants add an offset and length to the parameter list and allow a portion of the file to be retrieved. The third-party transfer variants invoke a server-to-server transfer, rather than a client-to-server transfer. This requires a little explanation.

The GridFTP protocol, like FTP, is a split-channel protocol. Commands are sent over one socket connection (the control channel) and data over another (the data channel). In a typical FTP client-server transfer scenario (see *Figure One*), both the control and data channels are opened between the client and the server, and the client is involved in the data transfer. In a third-party transfer, however (see *Figure Two*), the client opens two control channels — one to the source server and one to the destination server — but the data channel is opened between the two servers and the client is not involved in the actual movement of the data.

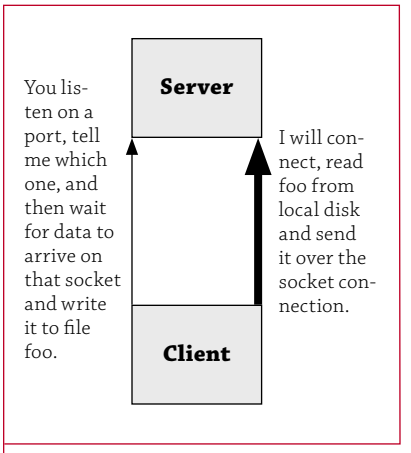

FIGURE ONE Standard FTP Client Server Model

Reading and Writing Data: The FTP operation commands handle the protocol exchange required. For many of the commands, such as making a directory, this is sufficient. No data movement occurs, merely a success or failure response over the control channel. However, the data movement commands also need to have code that actually drives the data movement. These functions handle this task. Note that they are asynchronous functions and that iteration is normally handled through callbacks. In other words, the callback function called at the completion of a write is normally a variation on "Are we at end of file? If not, register another write."

Restart Marker Handling: One of the significant improvements of GridFTP over FTP is its robustness to failure. GridFTP has the writing server periodically send back restart markers, or acknowledgments, of which blocks of bytes have been written. The helper functions ease the handling and consolidation of these markers.

The Globus Toolkit Programming Model

We turn next to how the library functions are actually used. The fact that we use an asynchronous programming model often seems strange to people developing C applications with the Globus Toolkit, We briefly explain this model as we go through the basic steps required for using the GridFTP client library. We examine code snippets and provide a small example of a whole file put. An in-depth explanation is beyond the scope of this article; check the resource side bar for more information.

Include headers: All code that makes calls to functions in the GridFTP client library must have the following include:

```
#include "globus_ftp_client.h"
```

Module Activation/ Initialization: In any Globus Toolkit C code, you must call the module activate/deactivate for any module that you make direct calls to. The module will activate and deactivate its own dependencies. For instance, Globus XIO, Globus GSI, and so forth used by the Globus GridFTP client library will automatically be activated when you call activate on the client library. However, if you also make calls directly to those modules — for instance, you also use XIO for doing the file IO — then you need to call activate/deactivate on XIO as well. The rule is, if you make a direct call to a module function, you must activate and deactivate. This is relatively simple:

```
result =
  globus_module_activate(
  GLOBUS_FTP_CLIENT_MODULE);
```

Handle Setup: Every function call is a completely encapsulated GridFTP session: a control connection is formed, authentication is done, if necessary a data channel is established, the necessary data is transferred, and then everything is torn down — unless you turn on channel caching. Basically, we recommend you always enable channel caching. It can result in a substantial performance gain; and even in a simple program you will end up with at least two connections, one to check the features the server supports (discussed below) and one to move the data. If you check the size of the file or do other file system operations, each of them is also a connection. Below we use the `cache_all` call. Most of the time, this will be all you need. However, if you have a long-running program that will access many different sites, you will need to do explicit caching by URL or to cache all and implement some algorithm for periodically flushing URLs from the cache, for instance a pool to limit the maximum number of connections and a timeout to keep connections from hanging around indefinitely. See the online documentation for descriptions of the other caching functions.

```
globus_ftp_client_handleattr_init(
  &handle_attr);
globus_ftp_client_handleattr_\
  set_cache_all(GLOBUS_TRUE);
globus_ftp_client_handle_init(
  &handle, &handle_attr);
```

Check Features: A good practice is to always verify that the server supports any features you intend to ex-

ploit. Since the minimum implementation of a GridFTP server is basically standard FTP with the gss security (RFC 2228), the server you are talking to may not support setting the buffer size or parallelism. Checking for features comprises four steps. First, you call `init`. Second, you call the features function, which sends the `FTP FEAT` command do the server, then parses the response, and loads the structure with the results. Third, to access the results, you call `is_feature_supported` listing the feature you are interested in; the features are an enumerated type. Fourth, when you are finished, you call `features_destroy` to free the memory for the structure. As mentioned above, checking features encapsulates an entire GridFTP session. Therefore you have to specify a URL (all you need is the protocol, host, and port in this case) and a callback function. Again, everything in the Globus Toolkit is asynchronous, so the call will return immediately, but you have to wait for the callback to know that the structure is populated and you can access it. In code it looks like *Listing One*, where the complete callback would simply set done to TRUE. Note that answer can be `GLOBUS_FTP_CLIENT_TRUE`, `FALSE`, or `MAYBE`. `MAYBE` means that particular feature was not probed; it does not necessarily indicate an error

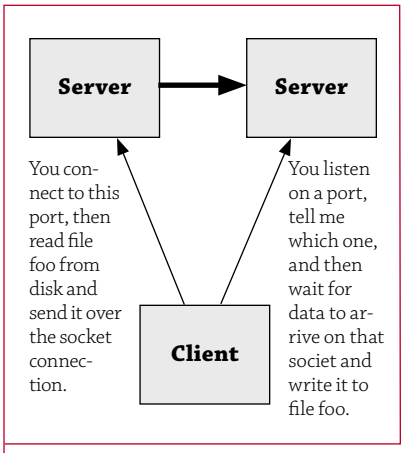

FIGURE TWO GridFTP Third-party Transfer Model

Set Operation Attributes: Once you know what features are supported by the servers you are using, you can configure any necessary attributes. The file system operations often may not use attributes, but the data movement operations often will. For a data movement operation, if you don't specify any attributes, the default will be standard RFC 959 stream mode (normal FTP), using RFC 2228 gss authentication via the Grid Security Infrastructure (GSI), an encrypted control channel and an authenticated data channel. We will briefly discuss the various attributes and then show a coding example of how they are used.

The operation attributes can be separated into two rough categories: data movement options and security options. All the function calls have the same form:

```
globus_ftp_client_operationattr_\
  [set|get]_"attribute".
```

The set variant changes the value of the attribute, and the get variant returns the current value of the attribute. The data movement attributes are as follows:

type: This attribute sets the file type to either ASCII or Image (binary).

mode: GridFTP supports two modes: stream (MODE S) and extended block (MODE E). Stream mode is what a normal FTP server uses; the file is moved by sending the bytes as an ordered sequence of bytes over the wire. Extended block mode is a GridFTP extension. This mode sends the data in blocks with eight bits of flags and a 64-bit offset and length *prepended*. This format allows out-of-order reception of the data because the offset in the transmission is specified. With out-of-order data possible, we can now exploit sending the data over multiple paths. In the current version of the server, we call this "parallelism." It involves only a single source and destination network endpoint (IP address or NIC) but has multiple TCP streams. Version 4 of the Globus Toolkit will also have a striping capability that will allow multiple endpoints, such as a cluster with a shared, parallel file system. Parallelism provides better performance by basically working around TCP limitations. Striping, when available, will allow users to break the single host performance bottleneck on very large files.

parallelism: This specifies the number of TCP streams that should be opened between each network endpoint (see the discussion of parallelism above).

LISTING ONE

Using the callback for asynchronous communication

```
result = globus_ftp_client_features_init(features);
result = globus_ftp_client_feat(handle,
          url, operation_attr,
          features, complete_cb,
          callback_args);
while(!done) {
  globus_cond_wait(&cond, &lock);
}
result = globus_ftp_client_is_feature_supported(features,
          answer,
          GLOBUS_FTP_CLIENT_FEATURE_PARALLELISM)
```

LISTING TWO

Example MODE E with four streams

```
globus_ftp_client_operationattr_init(&attr);
parallelism.mode = GLOBUS_FTP_CONTROL_PARALLELISM_FIXED;
parallelism.fixed.size = 4;
globus_ftp_client_operationattr_set_mode(
  &attr,
  GLOBUS_FTP_CONTROL_MODE_EXTENDED_BLOCK);
globus_ftp_client_operationattr_set_parallelism(
  &attr,
  &parallelism);
```

LISTING THREE

Starting the Protocol Exchange

```
result = globus_ftp_client_put(
   &handle,     /* the handle we initialized above */
   dst,         /* the URL of the destination */
   GLOBUS_NULL, /* Restart Markers, if any */
   done_cb,     /* callback when the transfer is complete */
   0);          /* an optional argument to the callback */
```

tcp_buffer: This is another important performance attribute. If the TCP buffer size is not sufficiently large, it will limit your performance. For an explanation of why that is so, please see last month's "On the Grid" article on GridFTP performance tuning.

The remaining operation attributes relate to security. Security is a critical aspect of any Grid application. GridFTP provides a wide range of security options. Both the authorization and the protection level may be set on the control channel and the data channel. Username/Password authentication is available only if the protocol specified is FTP. Since the username and password are sent in clear text, this is an extremely bad idea. GridFTP (which uses the *gsiftp protocol specifier* for historical reasons), offers three mode of authentication through *gss*: `NONE`, `SELF`, and `SUBJECT`. `NONE` means no authentication is performed, `SELF` indicates that the server should be running under your credentials, and `SUBJECT` allows you to specify the expected subject name that the server will authenticate with. The default mode has the server present a subject name containing the hostname on which it is running.

Protection relates to verification of the data. Standard security measures define four levels of protection: `CLEAR`, `SAFE`, `CONFIDENTIAL`, and `PRIVATE`. `CLEAR` indicates no protection of any kind; `SAFE` indicates that the data is integrity checked (i.e., a checksum is performed); `CONFIDENTIAL` means that the data is encrypted; and `PRIVATE` means that the data is both encrypted and integrity checked. The definitions above are from `RFC 2228`, the `PROT` command. However, we do not support the `CONFIDENTIAL` mode; it is actually treated as `PRIVATE`. In other words, you cannot encrypt without also getting integrity checking. The reason stems from a limitation in our underlying libraries (SSL).

For example, if you wish to set `MODE E` (required for parallelism), the parallelism to four streams, and your TCP buffer size to 2 MB per stream (not a bad group of rule-of-thumb defaults for gigabit transfers across the United States), the code would look like *Listing Two*.

Execute the Operation: We are now ready to move some data. The asynchronous nature of the Globus Toolkit comes into play here again. *Listing Three* shows how to start the protocol exchange.

However, all you have done is initiated the control channel protocol exchange. This command sends all of the necessary commands to the server; and since this is a put, the server waits for data to come down the data channel. However, since this is a client-server interaction and you are the client, you have to actually read the data off the disk and send it down the wire to the waiting server. This process is handled by the `register_read` or in this case, the `register_write` command (see *Listing Four*). Once again, the `register_write` is asynchronous, so it returns immediately. One of the parameters to `register_write` is the function (callback) to call when it has completed writing the data. This callback normally checks to see whether the file is at EOF; if not, it reads another block and calls `register_write` again. The process continues until the entire file has been moved.

A few notes are in order here. First, it is allowed — and, in fact, necessary for performance — to have multiple `register_write` calls outstanding. As a rule of thumb, we recommend 2 times the number of streams you are using. This method helps ensure that a stream is never

sitting idle. Second, since this is asynchronous, there is no guarantee of the order of arrival of the data at the server; hence, you should never have logic depending on the order.

Third, shared values, such as the current offset in the file, will need to have appropriate mutexes placed around them to avoid data corruption.

Module Deactivation and Cleanup: After the work is done, cleanup remains. Don't forget to free any buffers that you allocated or were allocated for you by calls such as `globus_error_get()` and `globus_print_friendly()`. Also, destroy anything that you initialized, and then deactivate the module as shown in *Listing Five*.

Or, if you are done with all your Globus Toolkit code and you want a short cut, you can call:

```
globus_module_deactivate_all()
```

That's it. You can now move data quickly and efficiently from a broad range of resources on the Grid. Besides the code snippets included inline, there is also a complete, though very simple, example that can be downloaded. See the resources sidebar for details.

Summary

We have provided a brief introduction to the Globus Toolkit GridFTP client library. This library provides very high-level commands that implement the protocol without requiring the developer to have an in-depth knowledge of the protocol. The toolkit mainly provides remote file system commands and data movement commands, as well as a range of other helper functions to assist in executing these operations. In Grid environments, GridFTP is the de facto standard for data access. A significant number of data sets are made accessible via a GridFTP protocol-compliant server. By adding support for the GridFTP protocol to your application, you gain access to these data sets. The Globus Toolkit GridFTP client allows you to do this with relative ease. The information presented here is intended to familiarize you with the asynchronous programming model that the GridFTP client library uses.

This work was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract W-31-109-ENG-38, by the National Science Foundation, by Los Alamos National Laboratory and by IBM.

Bill Allcock is the Technology Coordinator for GridFTP at Argonne National Laboratory and is a member of the Globus Alliance. His work centers on data-intensive applications, particularly transport and management of large datasets.

LISTING FIVE

Deactivating Modules

```
globus_ftp_client_operationattr_destroy(operation_attr);
globus_ftp_client_handleattr_destroy(handle_attr);
globus_ftp_client_handle_destroy(handle);
globus_module_deactivate(GLOBUS_FTP_CLIENT_MODULE);
```

Resources

Globus Toolkit API Documentation
- www.globus.org/developer/api-reference.html

Asynchronous Programming
- www-unix.globus.org/toolkit/docs/3.2/developer/globus-async.html

Example Codes
- www.clusterworld.com/codes/Oct04/grid

LISTING FOUR

Writing Data to the Destination

```
globus_ftp_client_register_write(
 handle,         /* the handle to our session */
  buffer,        /* the data to send to the server */
  length,        /* length of the data */
  global_offset, /* offset in the file where this should be written */
  feof(fd),      /* are we at EOF? */
  data_cb,       /* function to call when the write is complete */
  GLOBUS_NULL); /* argument to the callback */
```