A Parallel I/O Mechanism for Distributed Systems

Troy Baer and Pete Wyckoff
Ohio Supercomputer Center
1224 Kinnear Road
Columbus, OH 43212
{troy,pw}@osc.edu

Abstract

Access to shared data is critical to the long term success of grids of distributed systems. As more parallel applications are being used on these grids, the need for some kind of parallel I/O facility across distributed systems increases. However, grid middleware has thus far had only limited support for distributed parallel I/O.

In this paper, we present an implementation of the MPI-2 I/O interface using the Globus GridFTP client API. MPI is widely used for parallel computing, and its I/O interface maps onto a large variety of storage systems. The limitations of using GridFTP as an MPI-I/O transport mechanism are described, as well as support for parallel access to scientific data formats such as HDF and NetCDF. We compare the performance of GridFTP to that of NFS on the same network using several parallel I/O benchmarks. Our tests indicate that GridFTP can be a workable transport for parallel I/O, particularly for distributed read-only access to shared data sets.

1. Introduction

As grids of distributed systems become more commonly used for parallel computations, shared access to data becomes increasingly problematic. Most of the solutions used for shared data access on parallel systems, such as cluster or parallel file systems, are not appropriate for use across wide-area networks. However, grid data services can be accessed in ways very similar to parallel I/O systems, so extending parallel I/O capabilities to grid applications is a natural extension of existing practice.

1.1 MPI-2 Parallel I/O

The Message Passing Interface, or simply MPI, has long been the standard interface for developing parallel applications, particularly on distributed-memory architectures such as MPP supercomputers and clusters. MPI has been implemented on a large number of communication technologies, ranging from TCP/IP, to InfiniBand [13], to Globus [10]. However, MPI-2, the second major version of the MPI standard [9], also describes an interface for performing I/O operations on files in parallel, including interleaved individual or collective access to shared files. While the ROMIO implementation of the MPI-2 I/O interface [21, 22] has been implemented on a large number of storage systems thanks to its ADIO abstract device interface [20], the only implementation using a grid data management protocol is not widely available and relies upon a long deprecated protocol [3]. However, an ADIO driver using a widely available grid data service would immediately allow a large base of MPI programs to be used more effectively in grid environments.

1.2 Globus and GridFTP

The Globus Grid Toolkit [1] is a widely used set of tools and libraries for grid computing, including certificate-based authentication and data management services. The lowest level of the Globus data management services is GridFTP [7, 8], an extension of the venerable File Transfer Protocol [18] using the Globus Grid Security Infrastructure (GSI) [2]. The client interface to GridFTP supplies almost all of the operations needed to implement the MPI-2 I/O interface. Furthermore, since several MPI implementations already support authentication and communication using Globus protocols, building an MPI-2 I/O interface atop GridFTP allows for secure, shared access to files in those same environments.

1.3 Goals

In this paper, we will describe the design and then the implementation of a driver for the ADIO component of

ROMIO using the GridFTP client interface. We will also discuss the capabilities and limitations of this software, including functionality and performance. Finally, we will discuss future directions and improvements that could be made to the current implementation.

2. Design Considerations

Since a GridFTP "file system" does not implement all of the functionality of a POSIX file system, it requires some special care in the semantics of certain operations. As shown in Table 1, the GridFTP client interface has equivalents to most (though not all) POSIX and MPI-2 I/O operations, and in cases where there is not a one-to-one correspondence in functionality, the actions taken by the GridFTP driver for ROMIO must endeavor to conform as closely to the requirements of the MPI-2 specification as possible. In a few cases however, this proved extremely difficult.

2.1 GridFTP Namespace

The first consideration in designing an I/O storage system is its namespace. Files in GridFTP are referenced using a URL, of the forms `ftp://host/path/file` or `gsiftp://host/path/file`. The difference between these is the protocol used; `ftp://` URLs use the traditional FTP protocol, while `gsiftp://` URLs may also use GridFTP extensions such as GSI authentication, extended block mode, and parallel data transfers. Happily, ROMIO has support for a file system prefix with a URL-like syntax in its file name handling routines (eg. `pvfs:/pvfs/dir/file`), so extending them to handle GridFTP URLs was straightforward.

2.2 Basic I/O Operations Using GridFTP

The I/O operations in the GridFTP client interface, like most Globus routines, are designed to be called asynchronously. A callback routine supplied at invocation is executed when the operation completes. A mutual exclusion (mutex) lock is used to ensure that only one thread may signal the completion of a data transfer. A further complication is that GridFTP follows the FTP model of client-server operation, where there are separate control and data transfer channels. Thus, the algorithm for initiating a blocking read or write transaction is as follows:

1. Initiate I/O transaction on control channel
2. Initiate one or more data transfers on data channel
3. Acquire lock
4. Wait for control channel transaction to complete; this implies data transfer completion as well
5. Release lock

2.3 Limitations of the GridFTP Client Interface

While the asynchronous design of the GridFTP client interface makes it seem natural for use in implementing MPI-2 nonblocking I/O operations such as `MPI_File_iread()`, it unfortunately has a significant limitation: a GridFTP file handle may have only one control channel operation in progress at any time per client process [6]. MPI-2 I/O semantics, on the other hand, allow multiple asynchronous I/O operations to be in progress at the same time. This semantic mismatch can be addressed in two ways. The first and simpler approach is treat **all** I/O operations as blocking, which is the approach used by the version of the ROMIO GridFTP driver described here. This requires that MPI-2 nonblocking I/O operations block silently, which implies waits issued against these nonblocking operations return immediately. A more elegant but complex solution to this problem would be to maintain a FIFO queue of I/O operations and have the completion of the top operation on the queue initiate the operation following it.

Another significant limitation of the GridFTP client interface is that is it has no concept of access modes, such as read-only or write-only. The MPI-2 I/O interface, on the other hand, has a large set of access modes derived largely from POSIX. As a result, read and write operations at the ADIO driver level for GridFTP must check the access modes and set error conditions appropriately. For instance, if an `MPI_File_write()` is issued against a GridFTP URL opened read-only, the ADIO driver for GridFTP must abort the operation and issue an error before any GridFTP client routines are invoked.

The data transfer routines in the GridFTP client interface, `globus_ftp_client_register_read()` and `globus_ftp_client_register_write()`, have slightly different interfaces; the write routine has an offset parameter that the read routine lacks [5]. This affects the implementation of strided reads, since the read cannot specify an offset. There are at least two possible solutions to this. One is to implement a strided read as a sequence of separate invocations of the algorithm shown in the previous section; however, this has the additional overhead of doing multiple control channel operations and mutex lock/release sequences for a single read operation from the user's point of view. An alternative to this approach is to read the entire extent of the remote file containing the desired data to the client and then copy out

I/O Operation	POSIX	MPI-2 I/O	GridFTP
Create	`creat()` or `open(...,O_CREAT)`	`MPI_File_open(..., MPI_MODE_CREATE,...)`	none
Seek	`lseek()`	`MPI_File_seek()`	none
Read (non-blocking)	`aio_read()`	`MPI_File_iread()`	`globus_ftp_client_partial_get()` `globus_ftp_client_register_read()`
Read (blocking)	`read()`	`MPI_File_read()`	`globus_ftp_client_partial_get()` `globus_ftp_client_register_read()` `globus_cond_wait()`
Write (non-blocking)	`aio_write()`	`MPI_File_iwrite()`	`globus_ftp_client_partial_put()` `globus_ftp_client_register_write()`
Write (blocking)	`write()`	`MPI_File_write()`	`globus_ftp_client_partial_put()` `globus_ftp_client_register_write()` `globus_cond_wait()`
Sync	`fsync()`	`MPI_File_sync()`	none
Delete	`unlink()`	`MPI_File_delete()`	`globus_ftp_client_delete()`

Table 1. Comparison of I/O Operations

the desired portions to the user's buffer; this alternative requires additional memory but only one control channel operation and mutex lock/release. The second approach effectively trades bandwidth for server-side latency, which is appropriate for relatively dense accesses over small extents; it was the approach used in the implementation discussed below.

The GridFTP client interface also lacks some fundamental operations that makes certain MPI-2 I/O features difficult to implement. For instance, the GridFTP client interface has no equivalent to the POSIX `creat()` system call; there is no way to create a file that does not already exist without writing data to it. To work around this, the client can write a single byte to the beginning of the file to create it, then overwrite that byte with the first user-initiated write transaction. The GridFTP interface also lacks a mechanism to tell the remote server to flush its buffers to disk, which effectively makes `MPI_File_sync()` impossible to implement in a way that preserves MPI semantics. Finally, the GridFTP interface lacks a way of explicitly requesting locks or atomic updates on a file, which makes the implementation of `MPI_File_set_atomicity()` problematic.

3. Implementation Details

An ADIO driver atop the GridFTP client interface has been implemented and tested using the design described in Section 2. In the process of doing this, several issues were uncovered that had not been expected during the design phase of the project. In most cases, these issues could be worked around; however, in one case the issue caused a serious limitation in functionality.

3.1 Difficulties with Firewalls and Network Address Translation

The Ohio Supercomputer Center (OSC), like many sites, keeps the vast majority of its cluster compute nodes on private networks with network address translation (NAT) gateways to bridge to the public Internet and to filter incoming traffic. This is not recommended by the Globus developers, but the fact of the matter is that it is often necessary for both security and lack of sufficient IP address space. However, this caused significant problems with writing files to remote GridFTP servers; the behavior observed was that reading a file through a NAT firewall would succeed, but writes to the same file would fail. It was hoped initially that a NAT proxy for GridFTP similar to those used for traditional FTP could be developed, but those hopes faded upon further observation of how GridFTP allocates ephemeral ports on the client side. The European Data Grid community has observed similar behavior [14]. The Globus developers have a set of recommendations for how to configure a firewall to permit Globus services such as GridFTP through it [25]. However, these recommendations include assigning a static GridFTP client port range to each compute node, which is inherently unscalable and difficult to manage in systems with more than a few tens of compute nodes.

3.2 Limitations on MPI-I/O Functionality

As mentioned previously, limitations in the GridFTP client interface make it extremely difficult to implement `MPI_MODE_CREATE`, `MPI_File_sync()` and `MPI_File_set_atomic()`. Unexpectedly, the lack of atomic operations made it difficult to implement shared file pointers in the way used by other ROMIO drivers. The natural way to implement a shared file pointer is to store the current shared file pointer location in a sepa-

rate file and force atomicity on it. This approach has two drawbacks for GridFTP: first, it requires read/write access to the directory, which may not be available for remote GridFTP files; and second, it requires atomicity to work. GridFTP is not alone in lacking shared file pointers under ROMIO; a number of parallel file systems, including PVFS, also lack shared file pointers.

3.3 MPI-2 File Hints

The GridFTP client interface has a number of features that are potentially be useful to certain applications but are disabled by default, such as striping and multiple data transfer threads. To allow access to these features from MPI programs, a number of GridFTP file hints were implemented as part of the driver. The hints are stored as (*key*, *value*) pairs in an `MPI_Info` object, which is then used as an argument to `MPI_File_open()`. The hints implemented for GridFTP are summarized in Table 2. The three that may not be familiar to users of the original FTP protocol are `ftp_control_mode`, `parallelism`, and `striped_ftp`. The `ftp_control_mode` hint sets the FTP file transfer mode to either `stream` (traditional passive FTP) or `extended` (allowing parallel and partial transfers), the latter of which is the default. The `parallelism` hint sets the maximum number of threads that can be spawned by an individual client process to transfer data to and from the FTP server, with a default of one. The `striped_ftp` hint enables striped transfers when the FTP service is striped across multiple data servers.

4. Results

The GridFTP driver for ROMIO was developed and tested on the BALE cluster at OSC; this is a 51-node cluster of dual-processor Athlon systems used for instruction and research. The GridFTP driver has also been used in parallel grid applications on OSC's Cluster Ohio grid testbed [17]. The driver passes all but two of the test programs included with ROMIO; however, the two tests that fail require atomic updates and/or shared file pointers, so their failures were not unexpected.

4.1 Performance

Figures 1 through 4 compare the read and write performance (without and with buffer flushes) of the ROMIO test program `perf` on a file accessed via GridFTP and NFS on the OSC BALE cluster using 100 Mbit/s Ethernet. This application has an extremely simple parallel I/O

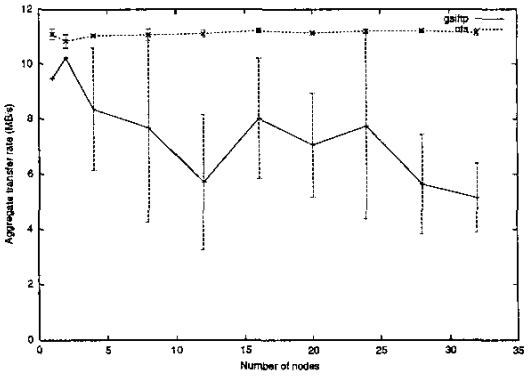

Figure 1. ROMIO `perf` Read Performance without Sync

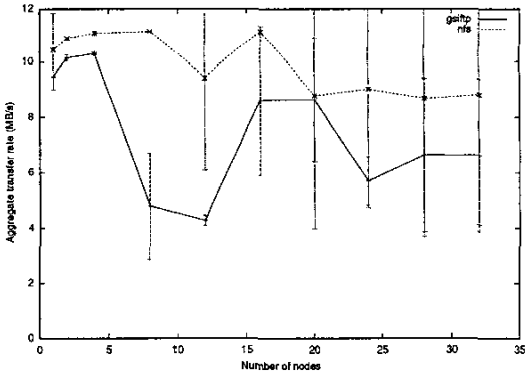

Figure 2. ROMIO `perf` Read Performance with Sync

Hint Key	Value Type	Possible Values
`ftp_control_mode`	string	`extended` (default) or `stream`
`parallelism`	integer	number of threads connecting to ftp server (default 1)
`striped_ftp`	string	`true`|`false` or `enable`|`disable` (default `false`)
`tcp_buffer`	integer	size of TCP buffer in bytes
`transfer_type`	string	`ascii` or `binary` (default)

Table 2. Hints for GridFTP Files in ROMIO

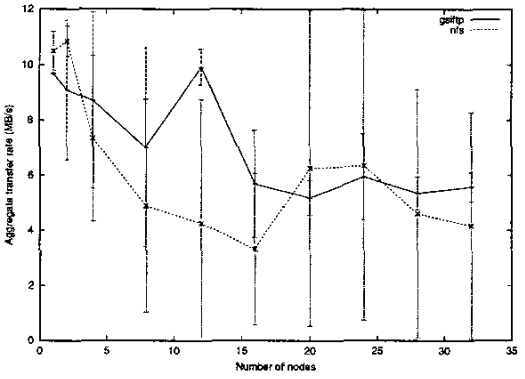

Figure 3. ROMIO `perf` Write Performance without Sync

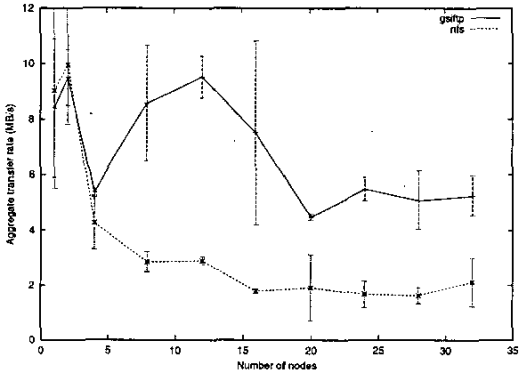

Figure 4. ROMIO `perf` Write Performance with Sync

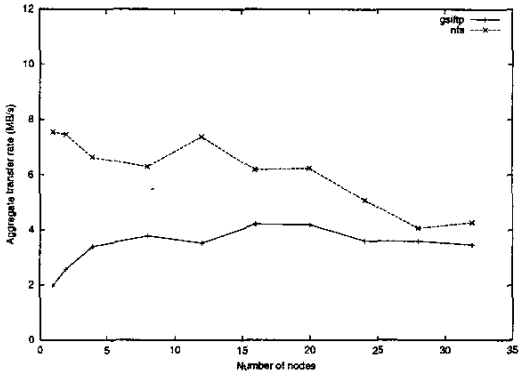

Figure 5. FLASH I/O Benchmark Checkpoint Performance using NetCDF

pattern: each MPI process independently writes and reads a chunk of data (four megabytes by default) at an offset of `chunksize * MPI_rank`. As can be seen from the plots, GridFTP is typically about two-thirds as fast as NFS for reads, but as fast as or faster than NFS for writes. The poor write performance on NFS can be attributed to its locking protocol. The large amount of variability observed is a result of network contention for the single network interface on the server side.

4.2 Parallel Access to Scientific Data Formats over the Grid

There are two widely used "standard" formats for portable, annotatable scientific data storage: HDF [16] and NetCDF [23]. Both of these have parallel I/O interfaces built on top of MPI-I/O; the HDF5 library has an MPI-I/O interface included with it [15], while a separate library for parallel access to Netcdf files has been developed by Argonne National Laboratory and Noprthwestern University [12]. Experiences with using these parallel interfaces over GridFTP were mixed. The HDF5 parallel interface relies heavily on `MPI_File_set_atomic()`,

and as a result it does not work well with GridFTP (or NFS, for that matter). Parallel NetCDF, on the other hand, works quite well over GridFTP.

A parallel application that uses both HDF5 and parallel NetCDF is the ASCI FLASH code from the University of Chicago [24]. This application has such heavy parallel I/O requirements that its I/O behavior has been made into a benchmark [26, 19]. The original FLASH I/O benchmark used HDF5's parallel interface; however, recent versions have used parallel NetCDF as well [4]. Figure 5 compares the checkpoint performance of the parallel NetCDF version of the FLASH I/O benchmark on the BALE cluster using GridFTP and NFS over 100 Mbit/s Ethernet. As in the case of ROMIO `perf` writes with buffer sync, NFS performance drops owing to lock contention; GridFTP performance is lower overall but more consistent.

4.3 An Application in Wide-Area Parallel Computing

Despite the limitations on writing files through NAT firewalls, using MPI-I/O over GridFTP on compute nodes in private networks such as those used by OSC and the Cluster Ohio grid still has significant applications. For instance, to facilitate using the Cluster Ohio grid environment for genome sequence alignment calculations, researchers at the Ohio State University wrote a wrapper around the popular tool BLAST that performs queries in parallel. MpiBLAST [11] uses MPICH-G2 with a worker-slave paradigm to farm out a set of queries against the same search database across multiple worker nodes. Both the input queries and the databases are accessed by the nodes using MPI-I/O. In several cases, the input and database files were referenced using GridFTP URLs to enable transparent remote access of a central data repository. Thus, MpiBLAST users could run the code anywhere in the grid using any sequence database without needing to first pre-stage the data to each potential compute node.

5. Conclusions and Future Directions

We have demonstrated that GridFTP can be used to implement the majority of the MPI-2 I/O interface. By using MPI-I/O for data access, developers of grid applications can reuse significant amounts of existing code, especially for shared read-only access to standardized scientific data formats like NetCDF. While GridFTP performance is often lower than that of NFS on the same network, it has the added benefits of better security (using GSI) and a lower performance penalty for simultaneous shared access from multiple clients. We are submitting a patch for this functionality to the ROMIO developers, so that it may be included in future version of ROMIO.

Future work will focus primarily on two areas: strided reads and non-blocking I/O operations. The current implementation's approach to strided reads does not scale to applications that need small, discontiguous portions of large data sets; a set of heuristics needs to be added to switch to doing multiple GridFTP read transactions, based on the ratio of the size of the data being read to the size of the extent in which that data lies. We also need to better support non-blocking I/O operations using the operation queue concept described earlier.

References

[1] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. 11(2):115–128, 1997.

[2] I. Foster, C. Kesselman, G. Tsudik, and S. Tuecke. A security architecture for computational grids. In *ACM Conference on Computers and Security*, pages 83–91. ACM Press, 1998.

[3] I. Foster, D. Kohr, Jr., R. Krishnaiyer, and J. Mogill. Remote I/O: Fast access to distant storage. In *Proceedings of the Fifth Workshop on Input/Output in Parallel and Distributed Systems*, pages 14–25, San Jose, CA, November 1997. ACM Press.

[4] B. Gallagher. HDF5/parallel-netCDF FLASH I/O benchmarks. `http://flash.uchicago.edu/~jbgallag/io_bench/`, 2003.

[5] Globus Project. Globus reference manual: globus_ftp_client data operations. `http://www.globus.org/api/c/globus_ftp_client/html/group__globus__ftp__client__data.html`.

[6] Globus Project. Globus reference manual: globus_ftp_client operations. `http://www.globus.org/api/c/globus_ftp_client/html/group__globus__ftp__client__operations.html`.

[7] Globus Project. GridFTP: Universal data transfer for the grid. `http://www.globus.org/datagrid/deliverables/C2WPdraft3.pdf`.

[8] Globus Project. GridFTP update January 2002. `http://www.globus.org/datagrid/deliverables/GridFTP-Overview-200201.pdf`, 2002.

[9] W. Gropp, E. Lusk, and R. Thakur. *Using MPI-2: Advanced Features of the Message-Passing Interface*. MIT Press, Cambridge, MA, 1999.

[10] N. Karonis, B. Toonen, and I. Foster. MPICH-G2: A grid-enabled implementation of the Message Passing Interface. *Journal of Parallel and Distributed Computing*, 63(5):551 – 563, 2003.

[11] M. Lauria. MpiBLAST parallel sequence alignment, 2003.

[12] J. Li, W. keng Laio, A. Choudhary, R. Ross, R. Thakur, W. Gropp, R. Latham, A. Siegel, B. Gallagher, and M. Zingale. Parallel netCDF: A high-performance scientifi c I/O interface. In *Proc. SC 2003*, 2003.

[13] J. Liu, J. Wu, S. P. Kini, P. Wyckoff, and D. K. Panda. High performance RDMA-based MPI implementation over InfiniBand. In *Proc. Int'l. Conf. on Supercomputing 2003*, 2003.

[14] A. McNab. Firewall issues for Globus 2 and EDG. `http://www.hep.man.ac.uk/u/mcnab/grid/firewalls05nov02.ppt`, Nov. 2002.

[15] National Center for Supercomputing Applications. HDF5 – a new generation of HDF. `http://hdf.ncsa.uiuc.edu/HDF5/`.

[16] National Center for Supercomputing Applications. Hierarchical Data Format. `http://hdf.ncsa.uiuc.edu/`.

[17] Ohio Supercomputer Center. The Cluster Ohio project. `http://oscinfo.osc.edu/clusterohio/`, 2004.

[18] J. Postel and J. K. Reynolds. RFC 959: File transfer protocol, Oct. 1985.

[19] R. Ross, D. Nurmi, A. Cheng, and M. Zingale. A case study in application I/O on Linux clusters. In *Proc. SC 2001*, 2001.

[20] R. Thakur, W. Gropp, and E. Lusk. An abstract-device interface for implementing portable parallel-I/O interfaces. In *Proceedings of the Sixth Symposium on the Frontiers of Massively Parallel Computation*, pages 180–187, October 1996.

[21] R. Thakur, W. Gropp, and E. Lusk. On implementing MPI-IO portably and with high performance. In *Proceedings of the Sixth Workshop on Input/Output in Parallel and Distributed Systems*, pages 23–32, May 1999.

[22] R. Thakur, E. Lusk, and W. Gropp. Users guide for ROMIO: A high-performance, portable MPI-IO implementation. Technical Report ANL/MCS-TM-234, Mathematics and Computer Science Division, Argonne National Laboratory, October 1997.

[23] University Corporation for Atmospheric Research. UniData NetCDF. `http://www.unidata.ucar.edu/packages/netcdf/`.

[24] University of Chicago. ASCI/Alliances center for astrophysical thermonuclear flashes. `http://flash.uchicago.edu/`.

[25] V. Welch. Globus toolkit firewall requirements. `http://www.globus.org/security/v2.0/firewalls.html`, July 2003.

[26] M. Zingale. FLASH I/O benchmark routine – parallel HDF5. `http://flash.uchicago.edu/~zingale/flash_benchmark_io/`, 2002.