The Globus Striped GridFTP Framework and Server

William Allcock[1] John Bresnahan[1] Rajkumar Kettimuthu[1] Michael Link[1]
Catalin Dumitrescu[2] Ioan Raicu[2] Ian Foster[1,2]
[1] *Math & Computer Science Division, Argonne National Laboratory, Argonne, IL 60439, U.S.A.*
[2] *Dept of Computer Science, University of Chicago, Chicago, IL 60615, U.S.A.*
allcock@mcs.anl.gov

Abstract

The GridFTP extensions to the File Transfer Protocol define a general-purpose mechanism for secure, reliable, high-performance data movement. We report here on the Globus striped GridFTP framework, a set of client and server libraries designed to support the construction of data-intensive tools and applications. We describe the design of both this framework and a striped GridFTP server constructed within the framework. We show that this server is faster than other FTP servers in both single-process and striped configurations, achieving, for example, speeds of 27.3 Gbit/s memory-to-memory and 17 Gbit/s disk-to-disk over a 60 millisecond round trip time, 30 Gbit/s network. In another experiment, we show that the server can support 1800 concurrent clients without excessive load. We argue that this combination of performance and modular structure make the Globus GridFTP framework both a good foundation on which to build tools and applications, and a unique testbed for the study of innovative data management techniques and network protocols.

1 Introduction

Rapid increases in both the quantity and diversity of data stored on secondary and tertiary storage systems, and in the raw capacity of wide area networks, make it both desirable and feasible, in principle at least, to move large amounts of data across wide area networks. For example, the NSF TeraGrid network links large clusters and storage systems at nine sites with a network providing up to 30 Gbit/s end-to-end. In principle, we should be able to move data across this network at more than 3 Gbyte/s, or 10 Tbyte/hr.

In practice, the orchestration of such transfers is technically challenging. One key issue is the frequent need to exploit parallelism in multiple dimensions, including (depending on context) storage systems, network interfaces, and backbone network trunks. Another is dealing with failures of various sorts. Firewalls, parallel file systems, and other specialized devices can also cause difficulties, as can the need to transform data before and/or after transfer. For these and other reasons, rapid, efficient, and robust wide area end-to-end transport requires the management of complex systems at multiple levels. For example, in recent work, we required 32 hosts connected at 1 Gbit/s to drive a 30 Gbit/s connection.

Effective end-to-end data transfers thus demand a systems approach in which file systems, computers, network interfaces, and network protocols are managed in an integrated fashion to meet performance and robustness goals. Furthermore, unless such approaches are encapsulated in software that is both easily usable (by end users and higher-level tools) and portable across different end system and network architectures, they will not be widely used.

These considerations motivate the work that we describe here, which concerns the design, implementation, and evaluation of a modular and extensible data transfer system architecture suitable for wide area and high-performance environments. This *Globus striped GridFTP framework* implements the GridFTP extensions [7] to the File Transfer Protocol (FTP) [45], which provide support for striped transfers from multiple data sources, failure detection, and other features. Both the framework and a high-performance striped server constructed within the framework form part of the Globus Toolkit [23] version 4 (GT4), and leverage Globus components for security and I/O functions.

The Globus GridFTP framework has a modular structure that allows for the coordination of multiple data streams, the substitution of alternative transport protocols, and other desirable features. These features

allow us to achieve a high fraction of end-to-end bandwidth over both local and wide area networks.

The rest of this paper is as follows. After discussing related work, we introduce in Section 3 the requirements that we seek to address, and in Section 4 review the GridFTP protocol. In Section 5, we describe the design of our framework and server, and in Section 6, we present experimental results. We conclude in Section 7.

2 Related Work

The efficient movement of distributed data is not a new problem. Parallel I/O systems commonly treat access to distributed data as a collective operation [49], and collective communication operations seek to optimize data transformations and transfers by coordinating related activities [35, 38]. In two-phase I/O [48] and in Remote I/O [25], data is read and then reorganized via interprocess communication prior to transfer. HPF/MPI [21] used the FALLS (FAmiLy of Line Segments) representation [47] to compute efficient inter-cluster communication schedules.

Researchers have come up with numerous solutions to address limitations of TCP's [44] AIMD-based congestion control mechanism [8]. These solutions include improvements to TCP [19, 33, 37], new transport protocols such as XCP [36], XTP [52] and reliable layers on top of UDP [5, 13, 14, 27, 30, 33, 51]. Our system is designed to interface to such high-performance communication protocols and to quality of service negotiation systems [22]. To date, our work has focused on the efficient use of TCP or other transport protocols on a per-stream basis. Our system could also manage all streams associated with a single transfer in a coordinated manner.

The Distributed Parallel Storage System (DPSS) [34] is a dynamically configurable collection of widely distributed disk servers that operate in parallel to provide high-speed random access to large data sets. Beck et al.'s logistical networking [10] also enables wide distribution (and replication) of data. Our system can make use of such systems when single node or site performance is the bottleneck.

Thain et al. [54] and Swany [53] describe data movement systems that make opportunistic use of disks in intermediate nodes to improve end-to-end performance. Our system can be used to transfer data between nodes in the end-to-end path.

BitTorrent [15] and Slurpie [50] allow clients to upload pieces of a file from multiple sources when multiple people are downloading the same file at the same time. As our system supports striping and partial file transfer, it could be used to good effect as a data transfer tool in these systems.

Distributed file systems [31, 43] can be used to enable access to remote data while maintaining file system semantics. The General Parallel File System (GPFS) [3] has achieved performance comparable to that of our system across wide area networks. GridFTP, and our system, are intended for use in less tightly coupled environments, in which file system semantics may be neither achievable nor desirable.

The work of Weigle and Chien [57] is perhaps closest to ours in terms of goals and approach. They conceptualize the M-to-N communication problem in terms of sets of nodes termed *composite endpoints*. They define an API for defining sender and receiver data distributions within composite endpoints, and introduce and evaluate algorithms for computing efficient communication schedules. Their techniques can integrate naturally with GridFTP.

3 Problem Statement

We review the requirements that motivated our design.

Striping. Continued commoditization of end system devices means that data sources and sinks are often clusters. Whether data is obtained from disk, sensors, or computation, the "end system" that drives a wide area link may involve many physical devices and considerable internal parallelism. This parallelism may also extend to the external network interface: a common configuration might have individual nodes connected by 1 Gbit/s Ethernet connections to a switch that is itself connected to the external network at 10 Gbit/s or faster. Thus, we wish to support *striped* data movement operations, in which data distributed across, or generated by, a set of computers or storage systems at one end of a network is transferred to another remote set of storage systems or computers.

Collective operations. While one can in principle express a data transfer between two clusters as a set of independent point-to-point transfers, it can be valuable to express such transfers as a single "collective" operation. Such an expression can permit a more concise description of the data transfer and provide a convenient logical unit for monitoring and management. Such an expression can also expose opportunities for optimization that might not be apparent in a set of point-to-point transfers. Thus, we wish to treat striped transfers as collective operations.

Uniform interfaces. Data sources and sinks come in many shapes and sizes, and may include clusters with local disks, clusters with parallel file systems, archival

storage systems (with or without parallel data mover support), and geographically distributed data sources. We want to make it possible for clients to access such sources and sinks via a uniform interface. We also want to make it easy to adapt our system to support different sinks and sources.

Network protocol issues. The standard protocol for network data transfer remains TCP. However, TCP's congestion avoidance algorithm can lead to poor performance, particularly in default configurations and on paths with high round trip times. Solutions to this problem include careful (ideally automated) tuning of TCP parameters [18], TCP protocol improvements [19, 33, 37], multiple "parallel" TCP connections [28, 46], and the substitution of alternative protocols [13, 14, 27, 33]. We want to support such alternatives.

End-to-end performance. Depending on context, high end-to-end performance can require the integrated management of many different devices, including storage systems, computers used to transform data, network interfaces, and network paths, and also perhaps other devices such as computers and storage systems located at intermediate points in a network. We would like to provide a framework within which a range of such end-to-end management approaches can be applied in a convenient manner.

Diverse failure modes. Collective operations, striped transfers, and end-to-end management offer opportunities for enhanced performance, but also introduce new failure modes. Our design must address robustness and fault tolerance.

4 GridFTP Protocol

We adopt the GridFTP data transfer protocol, rather than alternatives such as WebDAV [56], for five reasons. First, the FTP protocol [45] on which GridFTP is based separates control and data channels, enabling third-party transfers, that is, the transfer of data between two end hosts, mediated by a third host. Second, FTP is a widely implemented and well-understood IETF-standard protocol with a large base of code and expertise from which to build. Third, FTP provides a well-defined architecture for protocol extensions and supports dynamic discovery of the extensions supported by a particular implementation. Fourth, many extensions have been defined through the IETF, some of which are useful in the current context. Fifth, GridFTP adds new features that are relevant to our concerns.

The following is a summary of key GridFTP features.

Third-party control of data transfer. To manage large datasets for distributed communities, we must provide authenticated third-party control of data transfers between storage servers. A third-party operation allows a user or application at one site to initiate, monitor and control a data transfer operation between two other sites: the source and destination for the data transfer.

Authentication, data integrity, data confidentiality. GridFTP supports Generic Security Services (GSS)-API authentication of the control channel (RFC 2228) and data channel (GridFTP extensions), and supports user-controlled levels of data integrity and/or confidentiality. Data channel authentication is of particular importance in third party transfers since the IP address of the host connecting for the data channel will be different than that of the host connected on the control channel, and there must be some way to verify that it is the intended party.

Striped data transfer. Data may be striped or interleaved across multiple servers, as in a parallel file system or DPSS disk cache [34]. Thus, GridFTP defines protocol extensions that support the transfer of data partitioned among multiple servers.

Parallel data transfer. On wide-area links, using multiple TCP streams in parallel between a single source and destination can improve aggregate bandwidth relative to that achieved by a single stream [28, 46]. GridFTP supports such parallelism via FTP command extensions and data channel extensions. A GridFTP implementation can use long virtual round trip times to achieve fairness when using parallelism or striping [29]. Note that striping and parallelism may be used in tandem, i.e., you may have multiple TCP streams open between each of the multiple servers participating in a striped transfer.

Partial file transfer. Some applications can benefit from transferring portions of files rather than complete files: for example, analyses that require access to subsets of massive object-oriented database files. FTP allows transfer of the remainder of a file starting at a specified offset. GridFTP supports requests for arbitrary file regions.

Automatic negotiation of TCP buffer/window sizes. Using optimal settings for TCP buffer/window sizes can dramatically improve data transfer performance. However, manually setting TCP buffer/window sizes is an error-prone process (particularly for non-experts) and is often simply not done. GridFTP extends the FTP command set and data channel protocol to support both manual setting and automatic negotiation of TCP buffer sizes for large files and for large sets of small

files. Our system currently supports only manual setting of the TCP buffer size.

Support for reliable and restartable data transfer. Reliable transfer is important for many applications that manage data. Fault recovery methods are needed to handle failures such as transient network and server outages. The FTP standard includes basic features for restarting failed transfers that are not widely implemented. GridFTP exploits these features and extends them to cover its new data channel protocol.

5 Globus Striped GridFTP Design

The Globus striped GridFTP system aims for (a) modularity, to facilitate the substitution of alternative mechanisms and use in different environments and configurations, and (b) efficiency, in particular the avoidance of data copies. As in systems such as the x-Kernel [32], we achieve these goals via an architecture that allows a protocol processing pipeline to be constructed by composing independent modules responsible for different functions.

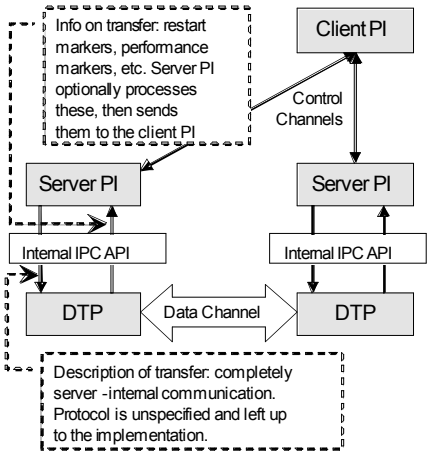

Figure 1: Globus GridFTP architecture

The implementation (Figure 1) comprises three logically distinct components: *client* and *server protocol interpreters* (PIs), which handle the control channel protocol (these two functions are distinct because the protocol exchange is asymmetric), and the *data transfer process* (DTP), which handles the accessing of the actual data and its movement via the data channel protocol. These components can be combined in various ways to create servers with different capabilities. For example, combining the server PI and DTP components in one process creates a conventional FTP server, while a striped server might use one server PI on the head node of a cluster and a DTP on all other nodes.

The DTP itself is further decomposed into a three-module pipeline (Figure 2). The *data access module* provides an interface to data source(s) and/or sink(s). The *data processing module* performs server-side data processing, if requested by an extended store/retrieve (ESTO/ERET) command. Finally, the *data channel protocol module* reads from, and/or writes to, the data channel. This basic structure allows for a wide variety of systems, from simple file server logic (data access module reads/writes files, data processing module does nothing, data channel protocol module writers/reads the data channel) to more complex and specialized behaviors (e.g., data module generates data dynamically in response to user requests).

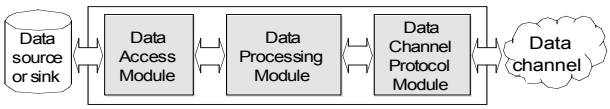

Figure 2: Globus GridFTP data transfer pipeline

5.1 The Protocol Interpreter

The server PI handles the control channel exchange. In order for a client to contact a GridFTP server, either the server PI must be running as a daemon and listening on a well known port (2811 for GridFTP), or some other service (such as inetd) must be listening on the port and be configured to invoke the server PI. The client PI then carries out its protocol exchange with the server PI.

During the preparatory phase of the protocol exchange, the server PI is concerned simply with developing a description of the transfer that is to take place. No communication is necessary with the DTP at this point; indeed, the DTP need not even be running. When a command is received that requires DTP activity, the server PI passes it the description of the transfer (first starting it, if needed), after which the DTP can carry out the transfer on its own. Once the transfer request is passed, the server PI simply acts as a relay for transfer status information. For example, the server DTP may send performance markers, restart markers, etc., to the server PI, which optionally processes them, and then sends them to the client PI.

PI-to-DTP communications are internal to the server, and thus this protocol used can evolve with no impact on the client. We used Message Passing Interface (MPI) [26] in an early prototype, which worked well but requires that MPI be installed. We currently use a binary protocol over TCP.

The data channel communication structure is governed by data layout. In general, if the number of nodes at both ends is equal, each node communicates

with just one other node. Otherwise, each sender makes a connection to each receiver, and sends data to each receiver based on data offsets.

5.2 DTP Data Access Module

This module is responsible for reading from, and/or writing to, a data source or sink. Its public interface includes transfer operations (list, send, receive) and command operations (e.g., make/remove directory, rename, checksum). Different implementations of this interface can be provided. We provide one for POSIX-accessible file systems and are working on one for the High Performance Storage System (HPSS).

5.3 DTP Data Processing Module

This module allows for (optional) server-side data processing, such as compression, scaling, or on-the-fly concatenation of multiple files. Normal (no server side processing) transfers are initiated with the STOR <filename>, for a put, or RETR <filename>, for a get. Data processing modules are invoked for puts and gets via the ESTO and ERET commands, respectively, which both take as arguments three strings: a module name, opaque module parameter, and filename. The module name is used to locate a loaded module in the module registry. The module is passed the parameter string and filename, and performs any necessary processing on the data as it transits the server.

We currently implement data processing module functionality within the data access module. We plan to separate this functionality out as a separate module and to allow for chaining of multiple modules.

5.4 DTP Data Channel Protocol Module

This module handles data channel processing, i.e., the operations required to fetch data from, or send data to, the data channel. A single server may support multiple data channel protocols, in which case the MODE command is used to select the protocol to be used for a particular transfer.

5.5 Security Considerations

The Globus GridFTP design provides for secure authentication of control channel requests (obligatory) and for data channel integrity and confidentiality (optional). GSS-API Grid Security Infrastructure (GSI) [24] and Kerberos [41] authentication bindings are supported. Standard Kerberos does not support data channel authentication, but there exist "user to user" extensions to Kerberos that do.

Security operations are performed via the GSS-API, for which Grid Security Infrastructure (GSI) [24] and Kerberos [41] authentication bindings are supported. We discuss GSI here.

A session is established when the client initiates a TCP connection to the port on which the server is listening. The first thing that must happen is an authentication per RFC 2228. By default, the client presents a delegated proxy certificate [55], and the server must present a "host certificate" issued by a CA trusted by the client and with a DN ending with a common name that is a direct match of that returned by a reverse DNS lookup of the server's IP address. It is possible to specify a subject name other than the default, and this is in fact necessary if you run the server as a user, in which case the server presents that user's subject name to the client. If authentication is not successful, the connection is dropped.

If authentication is successful, an authorization callout is invoked to (a) verify authorization and (b) determine the local user id as which the request should be executed. This callout is linked dynamically; Globus GridFTP provides an implementation that supports both a Globus "gridmapfile" and Community Authorization Service [42] credentials, which may encode in SAML assertions the specific files that a user is authorized to read and/or write. Sites can also provide alternative implementations. Server does a setuid to the local user id as determined by the authorization callout.

If authorization succeeds, the control channel has been established and the rest of the control channel protocol exchange can proceed. The control channel is encrypted and integrity protected by default.

To establish the data channel (the connection over which the actual data of interest will flow), a listening port must be established and the other end informed of this port. The GridFTP protocol requires that the receiver be the listener and that the sender issue the TCP connect. Thus, the client sends a PASV command to the server that is to receive the data. The receiver begins listening on a TCP port and responds to the command indicating the IP address and Port of the listener. (If this is a striped transfer, the client sends a striped PASV, or SPAS, command and an array of IP/ports is returned.) The client then sends to the other server a PORT (or SPOR, for striped port) command, which takes the IP/ports as a parameter. This command directs the server to initiate the TCP connect, and establish the data channel.

Third party transfer presents a security issue, as the receiving server starts listening on a port, but it has no way of knowing the IP address of the server that will connect to it. To mitigate this issue, we default to

requiring GSI authentication on the data channel as well. In this case, the server performs a delegation and both ends of the authentication must present the user's subject name (no host certificate is involved). All the parties involved in the transfer must accept the same CA.

Cryptographic confidentiality and integrity protection are both supported on the data channel, but are not enabled by default due to its cost (an order of magnitude is not unusual on high speed links).

When the PI and DTP are run in separate processes, they communicate over an Interprocess Communication link. Establishment of this link is exactly as per the control channel, with the PI acting as the client (using the delegated credential) and the DTP presenting its host certificate.

We have considered running the PI as a non-privileged user by default. This would prevent an external connection from ever being connected to a root process. In that case, the host certificate should be owned by the user. The only objection to this approach is that some other services might require that the host certificate be owned by root. We are exploring other options that would allow the PI to be run as non-privileged user.

File system security is handled via normal operating system mechanisms. Once the process is running as an unprivileged user, it is subject to access control and quotas imposed by the operating system.

6 Experimental Studies

We perform experiments in three settings: a local area network (LAN) with a 0.2 milliseconds (msec) round trip time (RTT) and a bottleneck link of 612 Mbit/s, a metropolitan area network (MAN) with 2.2 msec RTT and a bottleneck link of 1 Gbit/s, and a wide area network (WAN) with a 60 msec RTT and a bottleneck link of 30 Gbit/s. The MAN is the 1 Gbit/s Distributed Optical Testbed (DOT) [2]. The WAN is the TeraGrid [12] link between NCSA in Illinois and SDSC in California, on which each individual host has a 1 Gbit/s bottleneck link. Hosts are either 1133 MHz or more dual Pentium processors with at least 512 Mbyte memory and 1 Gbyte swap space, or (on TeraGrid) dual 1.3 Ghz Intel Itanium processors.

In all tests, we set the TCP buffer size to (bandwidth-delay-product/number-of-streams).

6.1 Comparison with Other FTP Servers

We first compare our server with two popular FTP servers, WU-FTPD [6] and NCFTP [4], under identical conditions: no striping, parallelism, or authentication, and in stream mode. We used a block size of 64 Kbytes for disk IO. We present in Figures 3 and 4 performance when transferring a file of size 1, 10, 100, and 1000 Mbytes, in our LAN and WAN. All data points are the means of 10 runs, with error bars also shown. We see that our server achieves superior performance in all cases, and does somewhat better relative to the other systems for larger files. It could be because of the efficient asynchronous event handling mechanism used in our implementation.

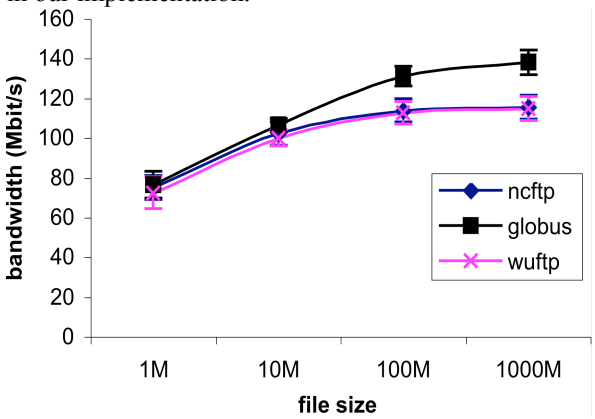

Figure 3: Single-stream throughput on LAN

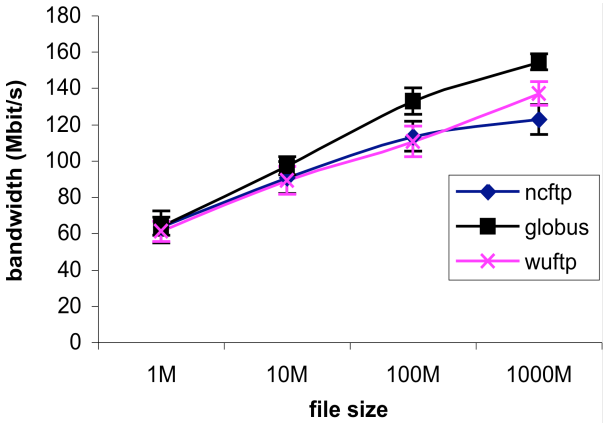

Figure 4: Single-stream throughput on WAN

6.2 Harnessing Parallelism

We next look at the impact of multiple streams on total achieved performance. Figures 5-7 show performance achieved in LAN, MAN, and WAN settings as a function of the number of streams. We show data for four different cases: Iperf, memory-to-memory Globus (/dev/zero to /dev/null), and disk-to-disk Globus, each running on a single node, as a function of the number of streams used; and the

Bonnie file system benchmark [1] that first writes and then reads a 1 Gbyte file on one of the two computers used in our experiment. For the Iperf and Globus memory-to-memory, we ran the application for 60 seconds. For the Globus disk-to-disk test, we transfer a 1 Gbyte file. For Bonnie, we measured read performance at the sender and write performance at the receiver, and report the lower of the two values.

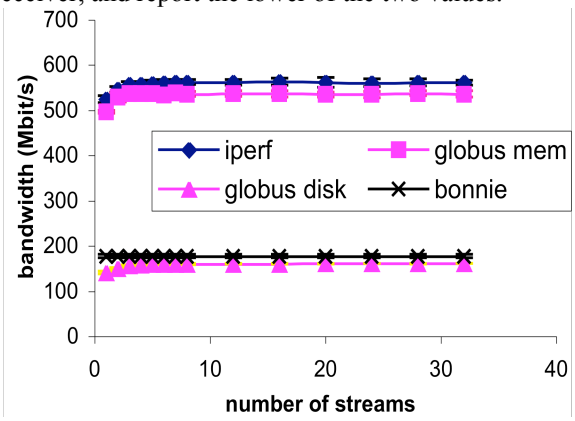

Figure 5: Parallel throughput on LAN

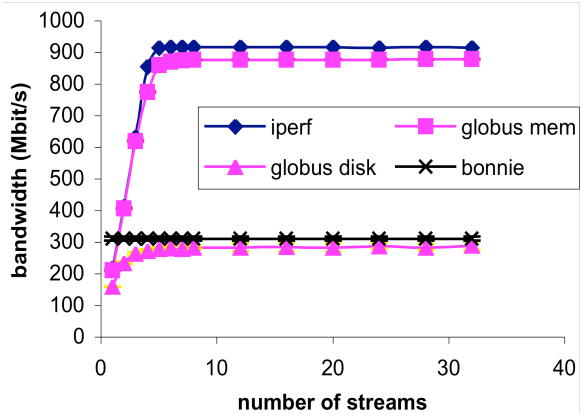

Figure 6: Parallel throughput on MAN

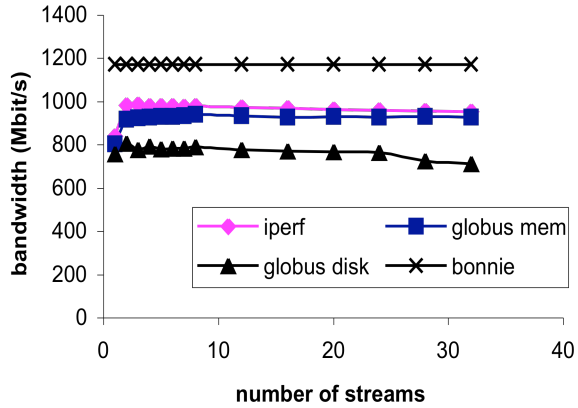

Figure 7: Parallel throughput on WAN

In the LAN case, Globus memory-to-memory performance matches that of Iperf, reaching 92% of bottleneck bandwidth; Globus disk-to-disk performance tracks that of Bonnie. Up to five streams seem to make a difference in all cases, after which little additional benefit is gained. In the WAN disk-to-disk case, we see somewhat more degradation of performance with increased streams. We attribute this result to more 'seek' operations at the receiver when using more streams, due to blocks received out of order (Figure 8).

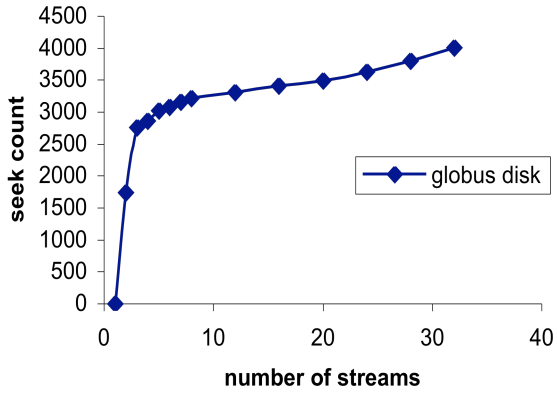

Figure 8: Seek operations at receiver vs. number of streams when transferring a 3 Gbyte file

6.3 Striping

We tested striped data transfers in both memory-to-memory and disk-to-disk modes. For disk-to-disk transfers, we used files of size 3*num_nodes Gbytes, except when using 64 nodes on each end, when we used (3/2)*num_nodes=96 Gbytes. Figure 9 shows memory-to-memory striped transfer performance. Note that with 32 nodes on each side, we achieved 26 Gbit/s over the 30 Gbit/s connection.

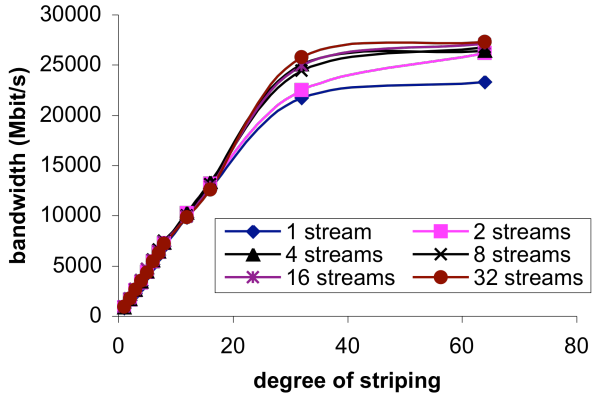

Figure 9: Globus mem-to-mem WAN performance

We have noted in numerous experiments that at lower speeds, increased streams did not equate to increased performance, only as we approached the bottleneck link speed, did the number of streams begin to have an effect. Though we were unable to obtain packet loss data, we suspect that this is because we have few or no packet losses until we begin to "compete with ourselves" and overflow the router buffers. In general, parallel streams are more effective with higher RTTs and with higher packet loss, though if packet loss were to become extreme to the point that all streams were losing packets, we might lose the benefits of multiple streams.

Figure 10 shows disk-to-disk striped transfer performance. We observe a significant reduction in performance compared to memory-to-memory transfers. To determine why, we ran Bonnie on multiple machines to measure the effect of multiple simultaneous operations on file system performance. Both NCSA and SDSC run GPFS [3]. NCSA has two GPFS scratch file systems, GPFS NSD and high performance GPFS SAN; in all experiments presented here, we used fast I/O machines connected to the SAN. Figure 11 shows the impact of multiple simultaneous operations on disk throughput. Our earlier experiments transferred data from SDSC to NCSA, and thus it is SDSC read performance and NCSA write performance that are relevant. It seems that SDSC read performance is currently the major obstacle to higher performance disk-to-disk transfers.

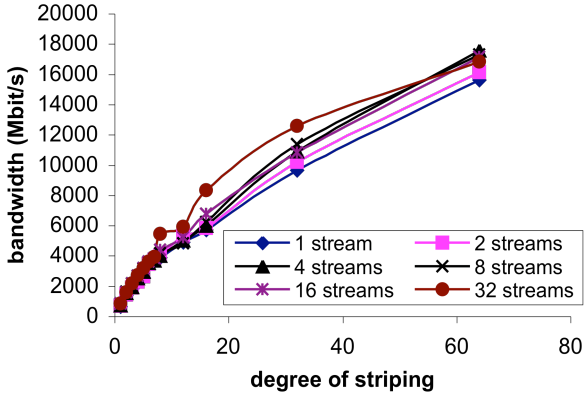

Figure 10: Globus disk-to-disk WAN performance

6.4 Scalability

Our final experiments evaluate Globus GridFTP performance as a function of the number of clients. We use the DiPerf test framework [17] to deploy clients on multiple servers and to collect performance data. The server, located in Los Angeles, was a 2-processor 1125 MHz x86 machine running Linux 2.6.8.1 with Web100 patches, 1.5 GB memory and 2 GB swap space, 1 Gbit/s Ethernet network connection and 1500 B network MTU. The clients were created on hosts distributed over PlanetLab [9] and at the University of Chicago (UofC). PlanetLab machines are generally connected by 10 Mbit/s Ethernet, and the UofC machines by 100 Mbit/s Ethernet

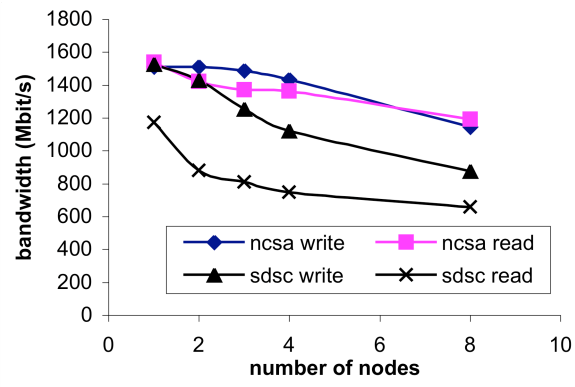

Figure 11: Parallel disk performance

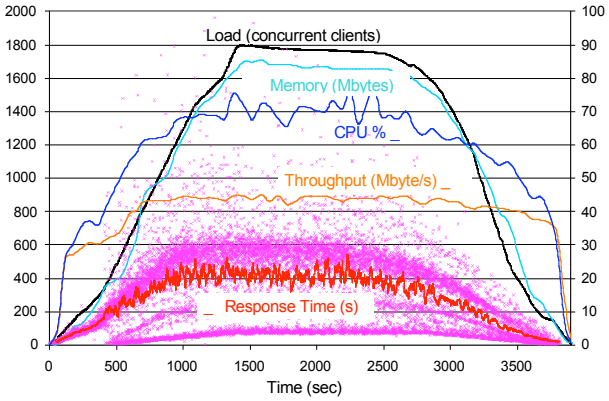

Figure 12: Scalability results with 1800 clients

Figure 12 shows results obtained with 1800 clients mapped in a round robin fashion on 100 PlanetLab hosts and 30 UofC hosts. A new client is created once a second. Each client runs for 2400 seconds and during this time repeatedly requests the transfer of a 10 Mbyte file from the server's disk to the client's /dev/null. A total of 150.7 Gbytes are transferred in 15,428 transfers. The left axis in Figure 12 indicates load (number of concurrent clients), response time (secs), and memory allocated (Mbytes), while the right axis denotes both throughput (Mbyte/s) and server CPU y%. The dots in the figure represent individual client

response times, while each of the lines represents a 60-second running average. Many other interesting characteristics are apparent in this figure, but are beyond the scope of this paper.

These results are encouraging. The server sustained 1800 concurrent requests with just 70% CPU and 0.94 Mbyte memory per request. Furthermore, CPU usage, throughput, and response time remain reasonable even when allocated memory exceeds physical memory, meaning that paging is occurring. Total throughput reaches 25 Mbyte/s with less than 100 clients and exceeds 40 Mbyte/s with around 600 clients.

7 Discussion

We have described a new open source implementation of the GridFTP protocol. In designing this system, we set out to create a robust, performant, and modular data transfer framework for use in a variety of data-intensive tools and applications. The resulting Globus GridFTP system integrates a variety of techniques, including a modular protocol processing pipeline and parallel I/O, to meet its design goals in a way that no other system has done before. Our system also provides support for IPv6.

We have tested our system thoroughly, as have early adopters. Performance is excellent in all situations studied, comparing favorably with that of other FTP servers for single-stream transfers and doing far better when striping is used. Performance with other network protocols, data transforms, and storage systems remains to be studied.

Our system's modular structure has allowed its use in many different contexts. We give four examples. The "TeraGrid Copy" (tgcp) program automatically selects appropriate parallelism and window size parameters to maximize performance on the TeraGrid network. The GT4 GRAM execution management service [16] uses our mechanisms for data staging and streaming. The NeST storage appliance [11] and the Earth System Grid's OPeNDAP-G system [20] use our libraries for data transport.

We have many ideas for further research and development. As indicated earlier, successful completion of an end-to-end transfer may involve intermediate staging of data products [39], negotiation with firewalls, use of alternative network protocols, and/or reservation of network or storage resources. Some such functions may appropriately be placed within, or require support from, our libraries. We also hope to exploit emerging Web services specifications to define more powerful and standards-based control interfaces, and to implement proposed GridFTP protocol improvements [40].

Acknowledgments

This work was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, U.S. Department of Energy, under Contract W-31-109-Eng-38, by Los Alamos National Laboratory, by the National Science Foundation's Middleware Initiative, and by IBM. We gratefully acknowledge access to facilities provided by NCSA, SDSC, DOT, and ISI.

References

1. Bonnie file system benchmark, 2005. www.textuality.com/bonnie.
2. Distributed Optical Testbed, 2004. www.dotresearch.org.
3. General Parallel File System (GPFS), 2004. www-1.ibm.com/servers/eserver/clusters/software/gpfs.html.
4. NcFTPd Server, 2005. www.ncftp.com.
5. Tsunami Network Protocol Implementation, 2004. www.indiana.edu/~anml/anmlresearch.html.
6. Washington University FTP Daemon (WU-FTPD), 2005. www.wu-ftpd.org.
7. Allcock, W. GridFTP: Protocol Extensions to FTP for the Grid. Global Grid ForumGFD-R-P.020, 2003.
8. Allman, M., Paxson, V. and Stevens, W. TCP Congestion Control. IETF, RFC-2581, 1999.
9. Bavier, A., Bowman, M., Chun, B., Culler, D., Karlin, S., Muir, S., Peterson, L., Roscoe, T., Spalink, T. and Wawrzoniak, M., Operating System Support for Planetary-Scale Services. *1st Symposium on Network Systems Design and Implementation*, 2004.
10. Beck, M., Moore, T. and Plank, J., An End-to-End Approach to Globally Scalable Network Storage. *ACM SIGCOMM*, 2002.
11. Bent, J., Venkataramani, V., LeRoy, N., Roy, A., Stanley, J., Arpaci-Dusseau, A.C., Arpaci-Dusseau, R.H. and Livny, M., Flexibility, Manageability, and Performance in a Grid Storage Appliance. *11th IEEE International Symposium on High Performance Distributed Computing*, 2002, IEEE Computer Society Press.
12. Catlett, C. The TeraGrid: A Primer, 2002. www.teragrid.org.
13. Chien, A., Faber, T., Falk, A., Bannister, J., Grossman, R. and Leigh, J. Transport Protocols for High Performance: Whither TCP? *Communications of the ACM*, *46* (11). 42-49. 2003.
14. Clark, D., Lambert, M. and Zhang, L. NETBLT: A Bulk Data Transfer Protocol. IETF, RFC 998, 1987.
15. Cohen, B. Incentives Build Robustness in BitTorrent. 2003.
16. Czajkowski, K., Foster, I., Karonis, N., Kesselman, C., Martin, S., Smith, W. and Tuecke, S. A Resource Management Architecture for Metacomputing Systems.

4th Workshop on Job Scheduling Strategies for Parallel Processing, Springer-Verlag, 1998, 62-82.
17. Dumitrescu, C., Raicu, I., Ripeanu, M. and Foster, I., DiPerF: Automated DIstributed PERformance testing Framework. *5th International Workshop in Grid Computing*, 2004.
18. Dunigan, T., Mathis, M. and Tierney, B., A TCP Tuning Daemon. *IEEE Supercomputing 2002*, Baltimore, Maryland, 2002.
19. Floyd, S. HighSpeed TCP for Large Congestion Windows. IETF, RFC 3649, 2003.
20. Foster, I., Alpert, E., Chervenak, A., Drach, B., Kesselman, C., Nefedova, V., Middleton, D., Shoshani, A., Sim, A. and Williams, D., The Earth System Grid II: Turning Climate Datasets Into Community Resources. *Annual Meeting of the American Meteorological Society*, 2002.
21. Foster, I., D. R. Kohr, J., Krishnaiyer, R. and Choudhary, A. A Library-Based Approach to Task Parallelism in a Data-Parallel Language. *Journal of Parallel and Distributed Computing*, *45* (2). 148-158. 1998.
22. Foster, I., Fidler, M., Roy, A., Sander, V. and Winkler, L. End-to-End Quality of Service for High-end Applications. *Computer Communications*, *27* (14). 1375-1388. 2004.
23. Foster, I. and Kesselman, C. Globus: A Metacomputing Infrastructure Toolkit. *International Journal of Supercomputer Applications*, *11* (2). 115-128. 1997.
24. Foster, I., Kesselman, C., Tsudik, G. and Tuecke, S., A Security Architecture for Computational Grids. *5th ACM Conference on Computer and Communications Security*, 1998, 83-91.
25. Foster, I., Kohr, D., Krishnaiyer, R. and Mogill, J. Remote I/O: Fast Access to Distant Storage. *IOPADS'97*, 1997, 14-25.
26. Gropp, W., Lusk, E. and Skjellum, A. *Using MPI: Portable Parallel Programming with the Message Passing Interface*. MIT Press, 1994.
27. Gu, Y. and Grossman, R.L., UDT: An Application Level Transport Protocol for Grid Computing. *Second International Workshop on Protocols for Fast Long-Distance Networks*, 2003.
28. Hacker, T., Athey, B. and Noble, B., The end-to-end performance effects of parallel tcp sockets on a lossy wide-area network. *16th IEEECS /ACM International Parallel and Distributed Processing Symposium*, 2002.
29. Hacker, T.J., Noble, B.D. and Athey, B.D., Improving Throughput and Maintaining Fairness using Parallel TCP. *IEEE InfoCom*, 2004.
30. He, E., Leigh, J., Yu, O. and DeFanti, T.A., Reliable Blast UDP: Predictable High Performance Bulk Data Transfer. *IEEE Cluster Computing*, 2002.
31. Howard, J.H., Kazar, M.L., Menees, S.G., Nichols, D.A., Satyanarayanan, M., Sidebotham, R.N. and West, M.J. Scale and Performance in a Distributed File System. *ACM Transactions on Computer Systems*, *6* (1). 51-81. 1988.
32. Hutchinson, N.C. and Peterson, L.L. The x-Kernel: An architecture for implementing network protocols. *IEEE Transactions on Software Engineering*, *17* (1). 64-76. 1991.
33. Jin, C., Wei, D.X. and Low, S.H., FAST TCP: motivation, architecture, algorithms, performance. *IEEE Infocom*, 2004.
34. Johnston, W., Greiman, W., Hoo, G., Lee, J., Tierney, B., Tull, C. and Olson, D., High-Speed Distributed Data Handling for On-Line Instrumentation Systems. *ACM/IEEE SC97: High Performance Networking and Computing*, 1997.
35. Karonis, N., Supinski, B.d., Foster, I., Gropp, W., Lusk, E. and Bresnahan, J., Exploiting Hierarchy in Parallel Computer Networks to Optimize Collective Operation Performance. *14th International Parallel and Distributed Processing Symposium*, 2000, 377-384.
36. Katabi, D., Handley, M. and Rohrs, C., Congestion Control for High Bandwidth-Delay Product Networks. *Sigcomm*, 2002.
37. Kelly, T., Scalable TCP: Improving Performance in High-Speed Wide Area Networks. *First International Workshop on Protocols for Fast Long Distance Networks*, 2003.
38. Kielmann, T., Hofman, R., Bal, H., Plaat, A. and Bhoedjang, R., MagPIe: MPI's Collective Communication Operations for Clustered Wide Area Systems. *7th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 1999, 131-140.
39. Kosar, T. and Livny, M. A Framework for Reliable and Efficient Data Placement in Distributed Computing Systems. *Journal of Parallel and Distributed Computing*. 2005.
40. Mandrichenko, I. GridFTP Protocol Improvements. Global Grid Forum, GWD-E-21, 2003.
41. Neuman, B.C. and Ts'o, T. Kerberos: An Authentication Service for Computer Networks. *IEEE Communications Magazine*, *32* (9). 33-88. 1994.
42. Pearlman, L., Welch, V., Foster, I., Kesselman, C. and Tuecke, S., A Community Authorization Service for Group Collaboration. *IEEE 3rd International Workshop on Policies for Distributed Systems and Networks*, 2002.
43. Popek, G.J., Guy, R.G., Thomas W. Page, J. and Heidemann, J.S., Replication in Ficus Distributed File Systems. *Workshop on Management of Replicated Data*, 1990, IEEE, 20-25.
44. Postel, J. Transmission Control Protocol. Internet Engineering Task Force, RFC 793, 1981.
45. Postel, J. and Reynolds, J. File Transfer Protocol. Internet Engineering Task Force, RFC 959, 1985.
46. Qiu, L., Zhang, Y. and Keshav, S., On Individual and Aggregate TCP Performance. *7th International Conference on Network Protocols*, 1999.
47. Ramaswamy, S. and Banerjee, P. Automatic Generation of Efficient Array Redistribution Routines for Distributed Memory Multicomputers. *Frontiers '95: The 5th Symposium on the Frontiers of Massively Parallel Computation*, McLean, Va., 1995, 342-349.
48. Rosario, J.d., Bordawekar, R. and Choudhary, A. Improved Parallel I/O via a Two-Phase Runtime Access Strategy. *Proceedings of the Workshop on I/O in Parallel Computer Systems at IPPS '93*, April 1993, 56-70.

49. Seamons, K.E., Chen, Y., Jones, P., Jozwiak, J. and Winslett, M. Server-Directed Collective I/O in Panda. *Supercomputing '95*, 1995.
50. Sherwood, R., Braud, R. and Bhattacharjee, B., Slurpie: A Cooperative Bulk Data Transfer Protocol. *InfoComm*, 2004.
51. Sivakumar, H., Grossman, R.L., Mazzucco, M., Pan, Y. and Zhang, Q. Simple Available Bandwidth Utilization Library for High-Speed Wide Area Networks. *Journal of Supercomputing*. 2004.
52. Strayer, W.T., Lewis, M.J. and Cline Jr., R.E. XTP as Transport Protocol for Distributed Parallel Processing. *USENIX Symposium on High-speed Networking*. 1994.
53. Swany, M., Improving Throughput for Grid Applications with Network Logistics. *SC'04*, 2004.
54. Thain, D., Basney, J., Son, S.-C. and Livny, M., The Kangaroo Approach to Data Movement on the Grid. *10th IEEE International Symposium on High Performance Distributed Computing*, 2001, IEEE Computer Society Press, 7-9.
55. Tuecke, S., Welch, V., Engert, D., Pearlman, L. and Thompson, M. Internet X.509 Public Key Infrastructure Proxy Certificate Profile. Internet Engineering Task Force, RFC 3820, 2004.
56. WebDAV Web-based Distributed Authoring and Versioning, 2004. http://webdav.org.
57. Weigle, E. and Chien, A.A., The Composite Endpoint Protocol (CEP): Scalable Endpoints for Terabit Flows. *CCGrid*, 2005.