GridFTP Multilinking

John Bresnahan,[1,2,3] Michael Link,[1,2] Rajkumar Kettimuthu,[1,2] and Ian Foster[1,2,3]
[1]Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439
[2]Computation Institute, University of Chicago, Chicago, IL 60637
[3]Department of Computer Science, University of Chicago, Chicago, IL 60637
{bresnaha, mlink, kettimut, foster}@mcs.anl.gov

***Abstract*—GridFTP is the de facto standard in projects requiring secure, robust, high-speed bulk data transport. For example, the high energy physics community is basing its entire tiered data movement infrastructure for the Large Hadron Collider computing Grid on GridFTP; the Laser Interferometer Gravitational Wave Observatory routinely uses GridFTP to move 1 TB a day during production runs; and GridFTP is the recommended data transfer mechanism to maximize data transfer rates on the TeraGrid. GridFTP is well known for efficiently transferring data from one destination to another. In this paper, we present a new feature for GridFTP – multilinking – and show how it can be used to transfer a single file to many destinations efficiently.**

I. INTRODUCTION

Large-scale collaborative science applications necessitate efficient and secure transport of data across geographically dispersed locations. The GridFTP [1] extensions to the File Transfer Protocol [2] define a general-purpose mechanism for secure, reliable, high-performance data movement. The Globus implementation of GridFTP [3] provides a modular and extensible data transfer system architecture suitable for wide-area and high-performance environments. Key features of GridFTP include the following:

1. Performance: Typical GridFTP provides order of magnitude performance improvements compared to standard FTP. GridFTP achieves good performance by using non-TCP protocols such as UDT [4] and parallel streams to minimize bottlenecks inherent in TCP/IP.
2. Cluster-to-cluster data movement: GridFTP can do coordinated data transfer by using multiple computer nodes at the source and destination. This approach can increase performance by another order of magnitude.
3. Reliability: GridFTP provides support for reliable and restartable data transfers.
4. Multiple security options: The Globus GridFTP framework supports various security options, including Grid Security Infrastructure (GSI) [5], anonymous access, username- and password-based security such as regular FTP servers, SSH-based [6] security, and Kerberos [7].
5. Modularity: The XIO-based [8] Globus GridFTP framework makes it easy to plug in other transport protocols. The Data Storage Interface (DSI) [9] allows for easier integration with various storage systems.
6. Third-party control: GridFTP also allows secure third-party clients to initiate transfers between remote sites.
7. Partial file transfer: Scientists often find it expedient to download only portions of a large file, instead of the entire file. GridFTP supports this capability by specifying the byte position in the file to begin the transfer.
8. Negotiation of TCP buffer/window sizes: GridFTP employs FTP command and data channel extensions to support both automatic and manual negotiation of TCP to get optimal performance.

In this paper, we take advantage of the modularity provided by GridFTP to achieve the following:

- Efficient transfers of a single dataset to many endpoints.
- An architecturally sound method for forming GridFTP overlay networks.
- An architecture for routing datasets from external networks to internal non-routable IP addresses

In addition, we describe:

- An experimental study to illustrate the benefits of this new functionality.
- The protocol enhancements and the usage options.

The rest of the paper is organized as follows. In Section 2, we present related work. In Section 3, we describe the architecture in detail. In Section 4, we present the experimental study. In Section 5, we highlight the enhancements to the GridFTP protocol and show how the new functionality can be utilized by using the commonly used GridFTP client "globus-url-copy." In Section 6, we present a brief summary.

II. RELATED WORK

Work most closely related to ours is that presented by

Rizk et al. [10]. Our network overlay architecture contribution is largely incremental. In our work we use the Globus XIO interface to add the needed functionality; in their work Rizk et al. used the DSI (Data Storage Interface) [9]. From a software engineering perspective XIO is a more modular, safer, and reusable way to achieve packet forwarding. From a performance and functional viewpoint, however, we expect the results to be similar to those presented in [10]. The most significant difference between our work and theirs is that we present two additional important advances: dataset broadcasting and internal routing to non-routable IP addresses. These two advances add significant power to GridFTP data transfers.

An extensive amount of work has been done in network multicast and data broadcast [11-16]. With regard to the broadcasting of datasets, we do not claim to achieve a new functionality; rather, we present a new way to achieve deployment of such functionality. We offer all of the known advantages of GridFTP, including speed, security, and availability, and we couple those advantages with the dataset broadcasts to many endpoints. Unlike other work in this area, we do not require new protocols or new software stacks. We have leveraged the extensibility of an existing and commonly deployed service to provide an important need to scientific communities.

III. Architecture

The purpose of this work is to efficiently route a single dataset to many locations. Our goal is to use all available network cycles effectively, moving the total amount of data (number of destinations * file size) at network speeds. Ideally we would like to transfer to all destinations in the time it takes to transfer to a single destination. Network latency, disk latency, and bandwidth make this goal difficult to fully achieve.

Distributing the data to many endpoints is not difficult and has been a feature of globus-url-copy (a popular GridFTP client) [17] for some time. It would be simple, but slow, to have the client loop over the destination set, sending to each in series. Of course the transfer time would scale linearly with the number of destinations and thus be undesirable.

In our architecture, destination servers are configured so that they can forward packets along to other endpoints. Thus the GridFTP servers act both as servers receiving data and as clients sending data. The servers are set up as a tree such that the first destination writes the data to disk (if it is configured to do so) and then forwards it on to N more hosts (by default N is 2). This process is repeated until all destinations have received the dataset.

One can think of the architecture as a directed graph. Each destination is a vertex with N edges connecting it to N other vertices. A spanning tree is formed connecting all vertices. The degree of any one vertex is a client side configuration option. Figure 1 illustrates this for the default value of N, which is 2. The data blocks are sent first from the client to a root destination. The root destination then forwards it to two more servers and so on, forming a spanning tree. The forwarding of this data is done at the packet level. It does not wait for a complete file to arrive at any given node. Thus we can overlap much of the latency involved in each transfer.

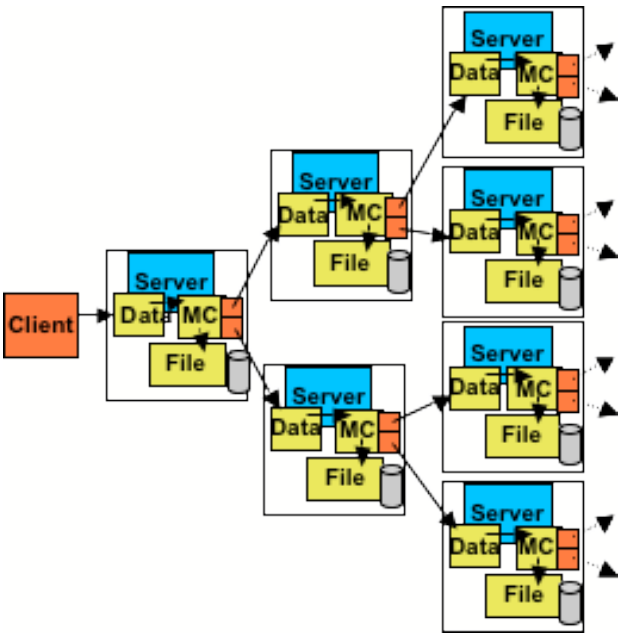

Figure 1: Architecture of GridFTP multilink.

A. Globus XIO

GridFTP uses the Globus Extensible Input Output (XIO) [8] interface for network and disk I/O operations. The Globus XIO framework presents a single, standard open/close read/write interface to many different protocol implementations. The protocol implementations, called drivers, are responsible for manipulating and transporting the user's data. Drivers are grouped into a stack. When an I/O operation is requested, the Globus XIO framework passes the operation request down the driver stack. An XIO driver can be thought of as a modular protocol interpreter that can be plugged into an I/O stack without concern about

the application using it. This modular abstraction is what allowed us to achieve our success here without disturbing the application's tested code base and without forcing endpoints to run new and unfamiliar code.

Figure 1 shows five boxes with different names. The "Client" box represents client logic; the "Server" box represents the GridFTP server logic. The "Data," "MC," and "File" boxes represent Globus XIO drivers. The "Data" driver handles the network interactions for the GridFTP server, and the "File" driver handles the file system interactions. "MC" is the new XIO driver created to provide multilink capability. The multilink functionality is achieved by allowing the GridFTP client to add the new XIO driver, the multilink (MC) driver, to the GridFTP server's disk I/O stack. Because of the modular driver abstraction that Globus XIO provides, as the GridFTP server writes data blocks to its file system, the data blocks are first passed through the multilink driver. As the multilink driver passes the data block on to be written to disk, it also forwards the block on to other GridFTP servers in the tree simultaneously. This approach is minimally invasive to the tested and robust GridFTP server.

B. *Network Overlay*

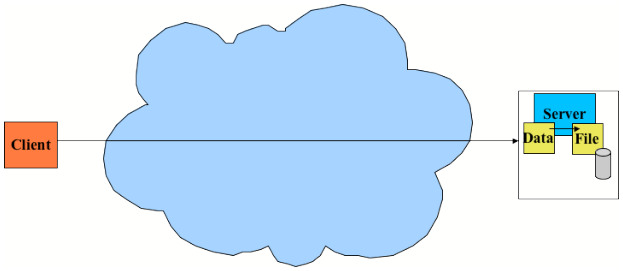

Figure 2: Standard GridFTP transfer.

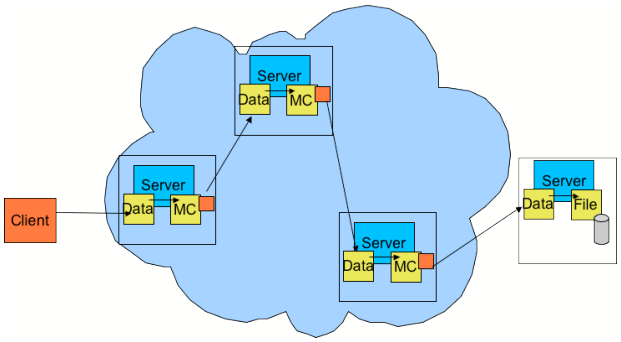

Figure 3: Transfer using a GridFTP overlay network.

In addition to allowing for dataset broadcasts, the multilink driver and this architecture allow us to create a network overlay where many GridFTP servers act as routers, forwarding packets along to each other until they get to the final destination where there are written to disk. The advantage of this type of system has been actively researched by Pheobus [18]. The multilink driver can be configured to only forward data along to the next server, and to not write it to disk. Further, it can be told to only forward to a single endpoint. With this configuration, shown in Figures 2 and 3, we achieve the network overlay described above.

Figure 2 shows the standard case where data is sent from a client to a server through the Internet. The Internet does the routing and the client does not have any control over it. Figure 3 shows how the multilink driver can be used to route data through the network via GridFTP servers. This feature allows the user to have greater control over the network path the data takes and thus achieve the advantages provided by network overlay research.

C. *Internal Routing*

Internal, nonroutable IP addresses are a common way to configure the worker nodes of a cluster. NAT [19] is used to provide the worker nodes with nonroutable IP addresses. Thus they cannot be contacted directly from the Internet. Access to these worker nodes is achieved by first contacting an externally visible head node that has a routable external IP address as well as a nonroutable internal IP address.

Because the internal worker nodes cannot be contacted directly, this common setup makes sending data directly to data node difficult. Common workarounds for this problem involve awkward port forwarding or manually transferring the data to the head node and then again manually transferring it to the worker nodes. This process is not only cumbersome but also inefficient.

The GridFTP multilink functionality allows a client to form a network overlay route such that the first hop contacts the head node, which then contacts the internal worker node. This process results in a direct transfer to the worker node without using disk space on the head node and without additional user or system administrator intervention.

IV. Experimental Results

To show the effectiveness of this architecture, we ran

experiments on the TeraGrid [20]. We used 30 nodes at TeraGrid's University of Chicago/Argonne National Laboratory site. The hosts are dual 1.3 GHz Intel Itanium processors with 4 GB of memory. We designated 29 hosts as destinations, and we ran multilink-enabled GridFTP servers on them. One node was designated as the client node, and from it all transfers were started. All transfers were performed with globus-url-copy (a command-line GridFTP client), and the TCP buffer size was set to 128 KB. We provided as input to globus-url-copy 29 pairs of source and destination URLs to measure the performance in the absence of multilink functionality. In this case the source file was sent to each endpoint serially. We then transferred the source file to all destinations using the multilink broadcast architecture.

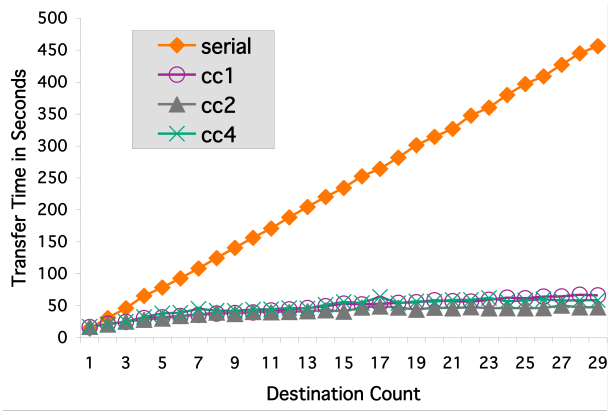

Figure 4: Comparison of the total transfer time for one-to-many GridFTP transfers with and without multilink capability.

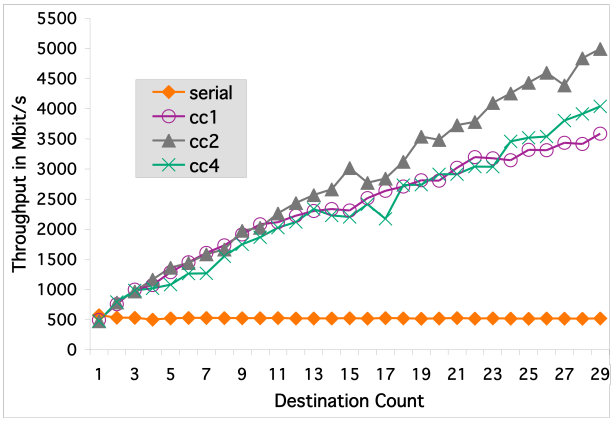

Figure 5: Comparison of the throughput for one-to-many GridFTP transfers with and without multilink capability.

Figure 4 shows the completion time of a broadcast session against the number of destinations. The "serial" line corresponds to the transfers performed in a serial fashion from the client. All other lines represent the multilink sessions performed with this architecture. Each line represents a different vertex degree in the spanning tree (i.e., each server forwarding to a different number of destinations). Here "cc1," or concurrency1, corresponds to each server forwarding to one other server, "cc2" corresponds to each server forwarding to two other servers, and "cc4" corresponds to each server forwarding to four other servers. As expected, the total transfer time for the serial transfer scales linearly with increasing number of destinations, whereas the total transfer time for multlink sessions very slowly increases with more destinations.

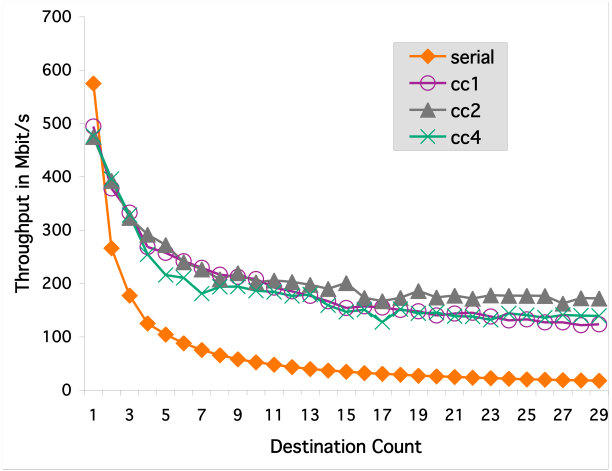

Figure 6: Effective throughput for various cases.

Figure 5 shows the throughput achieved for a multilink session by using serial transfers and different vertex degrees in the multilink architecture. The throughput is measured by dividing the size of the file being sent (1 GB) by the time it takes for the file to reach all the destinations. We note that the throughput achieved with multilink architecture is significantly better. Interesting, the throughput achieved with "cc2" is better than "cc1" and "cc4." We believe that the resources at the intermediate servers (for example, the network interface card) are not effectively utilized with "cc1" and the resources are used to a maximum extent with "cc2." Thus, "cc4" does not bring any further improvements. In fact it hurts the performance because of increased contention for resources. An alternative explanation is that the latency is additive in terms of the tree. Assume, for example, that the end-to-end latency for "cc1" is n. The end-to-end latency for "cc2" should be log(n), plus

the additional processing latency added at each intermediate node to forward the data to an additional node. For "cc2" the latency added by additional processing is less than the latency saved by shortening the tree. When we go to "cc4," the additional processing adds more latency than what is saved by shortening the tree.

Figure 6 shows the effective throughput achieved for various cases. Effective throughput is defined as (number of destinations * file size) / time. As the number of destinations increases, the closer the effective throughput is to the throughput achieved for a single destination transfer, the better it is. The overhead introduced by the intermediate nodes in the multilink architecture can be measured by how far off the effective throughput is from the throughput of a single destination transfer.

V. PROTOCOL DETAILS

The additions to the GridFTP protocol are minor. Every server in the tree (except for leaf nodes) becomes a client to another server, but that client speaks the standard GridFTP protocol. The only change needed is a command to add the multilink driver to the file system stack:

SITE SETDISKSTACK 1*{<driver name>[:<driver options>]}

The second parameter to the site command is a comma-separated list of driver names optionally followed by a ":" and a set of driver-specific URL encoded options. From left to right, the driver names form the I/O stack from bottom to top. Adding the multilink driver to this list will enable the multilink functionality. For security reasons the GridFTP server does not allow clients to load arbitrary XIO drivers into the server. The GridFTP server admin must white list the drivers individually.

A. Usage

The multilink functionality can be used with globus-url-copy by using the command-line option "-mc <filename>." The file must contain a line-separated list of destination URLs., for example:

gsiftp://host1:5000/home/user/tst1
gsiftp://host2:5000/home/user/tst2
gsiftp://host3:5000/home/user/tst3

The source URL is specified on the command line as always. An example globus-url-copy command is

% globus-url-copy -mc multilink.file gsiftp://localhost/home/user/src_file

ACKNOWLEDGMENT

This work was supported in part by the Office of Advanced Scientific Computing Research, Office of Science, U.S. Dept. of Energy, under Contract DE-AC02-06CH11357, and in part by National Science Foundation's CDIGS.

REFERENCES

[1] W. Allcock, "GridFTP: Protocol Extensions to FTP for the Grid," Global Grid Forum GFD-R-P.020, 2003.

[2] J. Postel and J. Reynolds, "File Transfer Protocol," IETF, RFC 959, 1985.

[3] W. Allcock, J. Bresnahan, R. Kettimuthu, M. Link, C. Dumitrescu, I. Raicu, and I. Foster, "The Globus Striped GridFTP Framework and Server, SC'05," ACM Press, 2005.

[4] Y. Gu and R. L. Grossman, "UDT: UDP-based Data Transfer for High-Speed Wide Area Networks," Comput. Networks 51, no. 7 (May 2007), 1777–1799.

[5] www.globus.org/security/overview.html

[6] T. Ylonen and C. Lonvick, eds., "The Secure Shell (SSH) Authentication Protocol," IETF, RFC 4252, 2006

[7] http://web.mit.edu/Kerberos/

[8] W. Allcock, J. Bresnahan, R. Kettimuthu, and J. Link, "The Globus eXtensible Input/Output System (XIO): A Protocol Independent IO System for the Grid," in Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium - Workshop 4, Vol. 5, IEEE Computer Society, Washington, DC, 2005. 179.1. DOI= http://dx.doi.org/10.1109/IPDPS.2005.429

[9] R. Kettimuthu, M. Link, J. Bresnahan, and W. Allcock, "Globus Data Storage Interface (DSI) – Enabling Easy Access to Grid Datasets," First DIALOGUE Workshop: Applications-Driven Issues in Data Grids, Aug. 2005.

[10] P. Rizk, C. Kiddle, and R. Simmonds, "A GridFTP Overlay Network Service," in Proceedings of the 7th IEEE/ACM International Conference on Grid Computing, 2006.

[11] M. Piatek, T. Isdal, T. E. Anderson, A. Krishnamurthy, and A. Venkataramani, "Do Incentives Build Robustness in Bittorrent?" in NSDI. USENIX, 2007.

[12] http://www.iam.unibe.ch/~rvs/research/summer_school_2005/TR%20RVS%2005.doc

[13] R. Sherwood, R. Braud, and B. Bhattacharjee, Slurpie: a cooperative bulk data transfer protocol. In Proceedings of the 23th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM). IEEE Computer Society, Los Alamitos, CA, 2004, 941-951.

[14] M. Mysore and G. Varghese, "FTP-M: An FTP-Like Multicast File Transfer Application," Technical Report. UMI Order Number: CS2001-0684, University of California at San Diego, 2001.

[15] http://www.tcnj.edu/~bush/uftp.html

[16] K. Lee, S. Ha, AND V. Bharghavan, "RMA: A Reliable Multicast Architecture for the Internet," In Proceedings of the IEEE Infocom '99, Vol. 3, March 1999, pp. 1274-1281.

[17] http://www.globus.org/toolkit/docs/4.0/data/gridftp/rn01re01.html

[18] http://e2epi.internet2.edu/phoebus.html
[19] K. Egevang and R. Francis, "The IP Network Address Translator," IETF, RFC 1631, 1994.
[20] TeraGrid. http://www.teragrid.org.

The submitted manuscript has been created by UChicago Argonne, LLC, operator of Argonne National Laboratory ("Argonne"). Argonne, a U.S. Department of Energy Office of Science Laboratory, is operated under Contract no. DE-AC-06CH11357. The U.S. Government retains for itself, and others acting on its behalf, a paid-up nonexclusive, irrevocable worldwide license in said article to reproduce, prepare derivative works, distribute copies to the public, and perform publicly and display publicly, by or on behalf of the government.