GridCopy: Moving Data Fast on the Grid

Rajkumar Kettimuthu[1,2], William Allcock[1,2], Lee Liming[1,2]
John-Paul Navarro[1,2], Ian Foster[1,2,3]

[1]Mathematics and Computer Science Division
Argonne National Laboratory, Argonne, IL 60439
[2]Computation Institute
The University of Chicago, Chicago, IL 60637
[3]Department of Computer Science
The University of Chicago, Chicago, IL 60637

{kettimut, allcock, liming, navarro, foster}@mcs.anl.gov

Abstract

An important type of communication in grid and distributed computing environments is bulk data transfer. GridFTP has emerged as a de facto standard for secure, reliable, high-performance data transfer across resources on the Grid. GridCopy provides a simple GridFTP client interface to users and extensible configuration that can be changed dynamically by administrators to make efficient data movement in the Grid easier for users.

1. Introduction

Because of the various specializations of each site in Grid [1,2] environments and because some applications require use of more than one site, application users commonly have to move data between sites. For example, the output of a large simulation computed at one site may need to be archived at another site and visualized for end users at a third site. This data is often large, ranging from several hundred gigabytes to tens of terabytes. The data may be stored in a small number of large files or a large number of smaller files.

Data movement is not a productive activity, so the time spent on it should be minimized. Above all, the "hands-on" time spent by scientists or application users to accomplish data movement must be minimized. GridFTP [3–7] has been commonly used as data transfer protocol in the Grid. The GridFTP protocol extends the standard FTP protocol to provide a superset of the features offered by the various Grid storage systems currently in use. The protocol includes the following:

- Public-key-based Grid Security Infrastructure (GSI) [8] and Kerberos support (both accessible via GSS-API).
- Third-party control of data transfer.
- Parallel data transfer (one host to one host, using multiple TCP [9] streams).
- Striped data transfer (*m* hosts to *n* hosts, possibly using multiple TCP streams if also parallel).
- Manual setting of the TCP buffer size.
- Partial file transfer.
- Reliable and restartable data transfer.
- Data channel caching.
- Integrated instrumentation, for monitoring ongoing transfer performance.

The Globus implementation of GridFTP [7] provides a software suite optimized for the gamut of data access issues—from bulk file transfer to the details of getting the data out of complex storage systems within sites on the Grid, and almost every data requirement in between. To get the maximum performance from the GridFTP server, users need to do some client-side optimizations. Often, however, the users are unaware of these optimizations or find it difficult to do them.

GridCopy, or GCP, provides a simple user interface to this sophisticated functionality, and takes care of all tuning required to get optimal performance for data transfers. The primary contributions of GCP are threefold:

1. Provide a SCP-style interface for high-performance, reliable, secure data transfers
2. Transparently calculate the optimal TCP buffer size and optimal number of parallel TCP streams to maximize throughput
3. Support configurable URL translations to optimize throughput

This paper is organized as follows. Section 2 describes some limitations in TCP for transfers over long, fat

1-4244-0910-1/07/$20.00 ©2007 IEEE.

pipes. Section 3 discusses the motivation for developing GridCopy. Section 4 gives an overview of GridCopy, and Section 5 delves into design details. Section 6 contains experimental results. Section 7 briefly summarizes the advantages of this new interface.

2. Background

By default, GridFTP uses TCP as its transport-level communication protocol. In order to get maximal data transfer throughput, it is critical to use optimal TCP send and receive socket buffer sizes for the link being used. If the buffers are too small, the TCP congestion window never fully opens. If the receiver buffers are too large, TCP flow control breaks, and the sender can overrun the receiver, thereby causing the TCP window to shut. This situation is likely to happen if the sending host is faster than the receiving host. Overly large windows on the sending side are not a big problem as long as excess memory is available. The optimal buffer size is twice the bandwidth-delay product (BDP) of the link.

buffer size = 2 * bandwidth * delay

The drawback of TCP for the high-bandwidth and high-latency networks, inherent in its AIMD-based congestion control mechanism [10–12] is well known [13–17]. The problem is that the number of packets in flight can be large, and the time taken to recover from a congestion event is directly proportional to the BDP. Hence, TCP is not scale-invariant with respect to bandwidth. For example, if we have a 200 ms path with a capacity of 1 Gbit/s, it will take at least 28 minutes to recover from a single congestion event (based on a standard packet size of 1,500 bytes).

3. Motivation

To overcome existing TCP problems, we have designed GridFTP to include features such as establishing multiple TCP connections in parallel to accelerate startup in the TCP slow start phase and to accelerate the linear increase in the congestion avoidance phase, and negotiating the TCP socket buffer size between the GridFTP server and client according to the bandwidth-delay product of a network. With existing GridFTP clients such as globus-url-copy [18], uberftp [19], and RFT [20], the user has to specify the appropriate socket buffer size and the number of parallel connections to get optimal performance with GridFTP. This is not an easy task for many users.

Also, these clients require the user to know details about how local and remote file systems are organized and where local and remote GridFTP servers are installed. Typically, users are familiar with the "secure copy" command (scp) and prefer its simpler style of specifying source and destination files, but this command cannot attain the performance required by Grid users.

These factors motivated the development of GridCopy, which provides an SCP-style interface and takes care of all required tuning.

4. Overview of GridCopy

GCP accepts SCP-style source and destination specifications. Local paths can be relative or absolute; remote paths look the same but have the hostname and a colon as a prefix. If well-connected GridFTP servers can access the source file and/or the destination file, GCP translates the filenames into the corresponding names on the GridFTP servers. (This procedure is explained in Section 5.) In addition to translating the filenames/URLs into GridFTP URLs, GCP adds appropriate protocol parameters such as TCP buffer size and number of parallel streams, in order to attain the optimal network performance for the specific source and destination. GCP initiates the data transfer using a GridFTP client such as globus-url-copy or RFT.

If both the source and the destination are remote to the client, then, in contrast to SCP, GCP performs the transfer directly from the remote source to the remote destination (a "third party transfer") without any data passing through the client system. The GCP command also allows the source to be a directory, in which case the entire contents of the directory are transferred to the destination (which must also be a directory).

GCP offers a small number of command line options in addition to the source and destination specifications. The "-rft" option instructs GCP to use the Reliable File Transfer (RFT) service to manage the transfer, which allows the transfer to restart and continue to completion in the event of failure in the client, the source or destination GridFTP servers, or the RFT service itself. The "-big" option instructs GCP to use a striped GridFTP transfer, which uses multiple nodes at the source and destination to consume more network bandwidth for the transfer than a single system could use by itself.

In addition to these two options, GCP accepts and passes through any globus-url-copy or RFT options that the user specifies. Most GCP users may not know what these options are, but sophisticated users may customize their transfers further by using these additional features.

5. GCP Design

Tools such as ping [21] and synack [22] can be used to estimate end-to-end delay; and tools such as IGI [23],

pathChirp [24], STAB [25], abing [26], pathrate [27], Iperf [28], pipechar [29], pchar [30], and Spruce [31] can be used to estimate end-to-end bandwidth. Latency estimation tools need to be run on one of the two nodes between which the latency needs to be estimated. Bandwidth estimation tools have two components. One component needs to be run at one end, and the second component needs to be run at the other end. Thus, these tools cannot be used to estimate available bandwidth or bottleneck capacity and delay between arbitrary hosts on the Internet.

For data transfers between a client and server, the tools mentioned above can be used to estimate the bandwidth-delay product. However, in Grid environments, users often perform third-party data transfers, in which the client initiates transfers between two servers. Sometimes third-party transfers are needed to get optimal performance; furthermore, GridFTP's striping features can be used only in server-to-server transfers. The end-to-end delay and bandwidth estimation tools cited above are not useful for third-party transfers. King [32], developed at the University of Washington at Seattle, makes it possible to calculate the round-trip time (RTT) between arbitrary hosts on the Internet. GCP uses King to estimate the RTT between source and destination nodes in a transfer. Estimating the bandwidth between any two arbitrary hosts on the Internet (without installing any tools on those hosts) is difficult. Thus, GCP assumes a fixed one Gbit/s bandwidth for all source and destination pairs.

King estimates RTT between any two hosts in the Internet by estimating the RTT between their domain name servers. To do this, King depends on the fact that most domain name servers in the current Internet (~75%–80%) support *recursive* queries from any host. The accuracy of King depends crucially on another fact about the domain name servers in the Internet. In many cases the name servers are located close (in network latency terms) to their hosts. While the first fact is the result of a default choice by many name server administrators, the second fact arises more out of administrative convenience than anything else.

For example, if King estimates the RTT between the source and the destination to be 50 ms, GCP sets the TCP buffer size to 0.05 s * (1 Gbit / 8 bits) = 6.25 MB. GCP caches the source, destination, and buffer size in $HOME/.gcp/opts.conf. $HOME is the home directory of the user running GCP. By default, GCP uses four parallel streams for the first transfer between two sites by a user. This value is configurable. GCP calculates the TCP buffer size for each stream as follows:

BDP / max(1, streams / loss_factor),

where loss_factor is set to two, by default, to accommodate for the fact that the streams that are hit by congestion would go slower and the streams that are not hit by congestion would go faster. Let us assume a packet gets dropped on one stream. TCP backoff causes the stream that dropped the packet to halve its bandwidth. But as there are four streams, instead of losing 1/2 the total bandwidth, we lose only 1/8. The other streams will consume that now-free 1/8 if they have sufficient buffer space. The loss_factor in the above equation is used to provide the extra buffer space for the streams to consume any additional bandwidth made available as a result of some streams dropping a packet.

GCP caches the source, destination, number of streams used, TCP buffer size for each stream, and throughput obtained along with a "decrease streams" flag set to one in $HOME/.gcp/opts.conf. This flag is used to determine whether to decrease the number of streams for the next transfer between the same endpoints. For the subsequent transfer from the same user between the same two endpoints, if "decrease flag" is set, GCP reduces the number of parallel TCP streams to one less than the value used previously. If the throughput obtained is not less than 97% of the stored throughput value, GCP overwrites the number of streams used, updates TCP buffer size value and throughput obtained for that source and destination pair, and leaves the "decrease streams" flag on, if the number of streams used is greater than one. Otherwise (if the throughput obtained with three streams is less than 97% of the throughput obtained with four streams), GCP just turns the "decrease streams" flag off. If the "decrease streams" flag is not set, GCP just uses the number of streams and the TCP buffer size corresponding to the source, destination pair in $HOME/.gcp/opts.conf.

GCP uses a configuration file to translate the user's simple specification of source and destination into a potentially complicated data movement request. The configuration file provides a set of translation rules that translate the source and destination specifications into service instances and paths that are known to those services. During this translation, the hostname may be changed to use a "designated transfer service" for the host that was originally specified (or the local system). A port number may also be substituted. The system administrators at each resource site in a virtual organization, using their knowledge of the local system configuration, must fill this configuration file uniquely. A shared section provides translation rules to be applied when using remote sites. This solution has scalability issues. We plan to provide more scalable solutions using MDS [33] and/or PubSub [34] models. System administrators at individual sites can publish the translation information to one or more of these services and GCP can obtain the translation information from one such service. The translation information has to be replicated and/or distributed across multiple services and/or locations to eliminate bottlenecks and single point

of failures. The translation information needs to be cached locally, and the cache has to be kept as consistent as possible.

If there is no translation corresponding to the source and/or destination provided by the user, GCP will attempt the transfer with the source and destination URLs provided by the user. Also, if the user knows that the source and/or destination URL(s) provided by him are/is appropriate, he can turn off the translation of source and/or destination URL(s).

6. Experimental Results

We compared the performance of GCP with globus-url-copy (GUC) for both memory-to-memory transfers and disk-to-disk transfers over three different network links on the TeraGrid [35]:

1. A 4 ms RTT link between the University of Chicago/Argonne National Laboratory and the National Center for Supercomputing Applications (NCSA)
2. A 15 ms RTT link between NCSA and the Pittsburgh Supercomputing Center (PSC)
3. A 75 ms RTT link between PSC and the San Diego Supercomputer Center

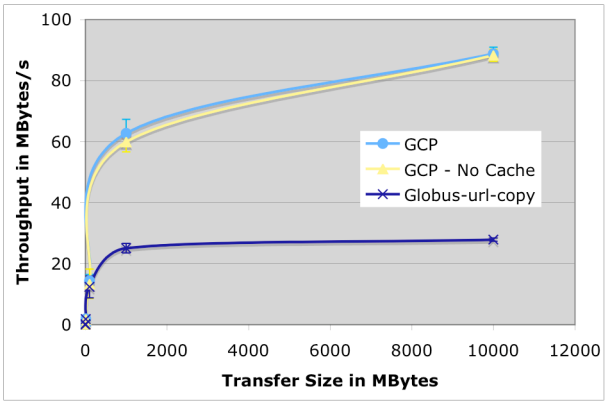

Figure 1: Comparison of performance of memory-to-memory transfers in GCP with GUC over a network with 4 ms RTT.

For all experiments we measured performance for a range of transfer sizes from 1 Kbytes to 10 Gbytes. For memory-to-memory experiments, we used /dev/zero as source and /dev/null as destination. Each point in the graphs represents an average throughput value of 50 transfers run a different times in a day. The "Globus-url-copy" legend in Figures 1–6 refers to transfers done with the default TCP buffer size (usually 64 KBytes). "GCP – No Cache" refers to transfers done with GCP using the bandwidth-delay product (BDP) as the TCP buffer size and the BDP calculated using the delay computed with King for each transfer. "GCP" refers to transfers done with GCP using BDP as the TCP buffer size and the BDP picked up from a local cache (as explained in Section 5). In Figure 6, we also show "GUC – Non Dedicated" results, corresponding to transfers between login (rather than dedicated) nodes on both ends.

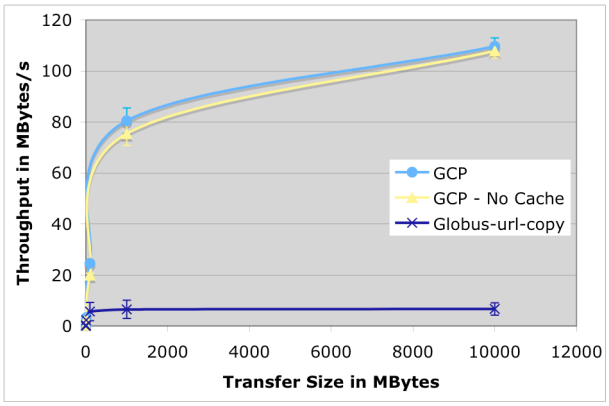

Figure 2: Comparison of performance of memory-to-memory transfers in GCP with GUC over a network with 15 ms RTT.

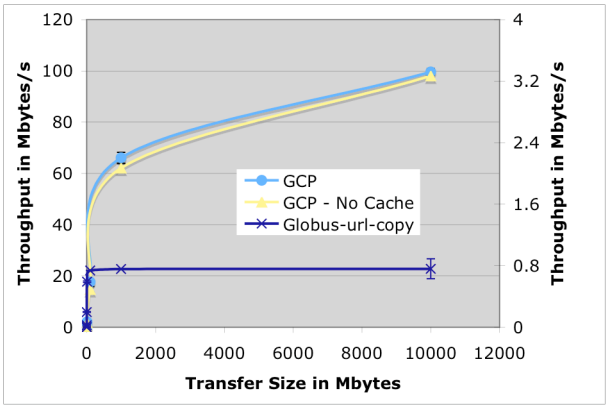

Figure 3: Comparison of performance of memory-to-memory transfers in GCP with GUC over a network with 75 ms RTT.

The "Globus-url-copy" values in Figure 3 and "Globus-url-copy" and "GUC – Non Dedicated" values in Figure 6 should be read on the secondary y-axis on the right side. All experiments used just one TCP stream. (GCP runs did not use its feature for tuning the number of TCP streams used.) Furthermore, all experiments except those labeled "GUC – Non Dedicated" legend in Figure 6 transferred data between dedicated servers (nodes that run GridFTP servers and are used only for transferring data) on both ends. From Figures 1–6, we see that GCP improves performance dramatically—by a factor ranging from a low of three to more than two orders of magnitude compared to GUC, with no additional inputs other than just the source and

destination URLs from the user. The longer the RTT, the higher is the improvement factor.

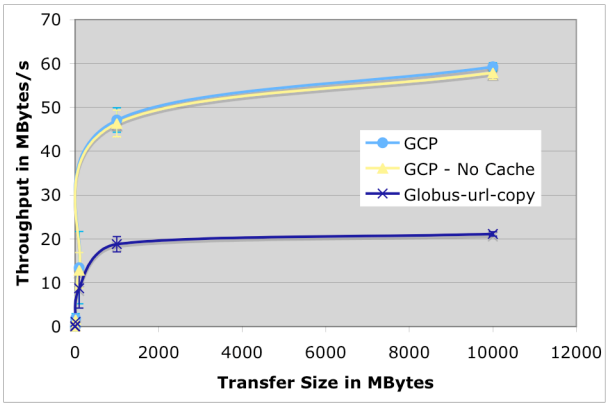

Figure 4: Comparison of performance of disk-to-disk transfers in GCP with GUC over a network with 4 ms RTT.

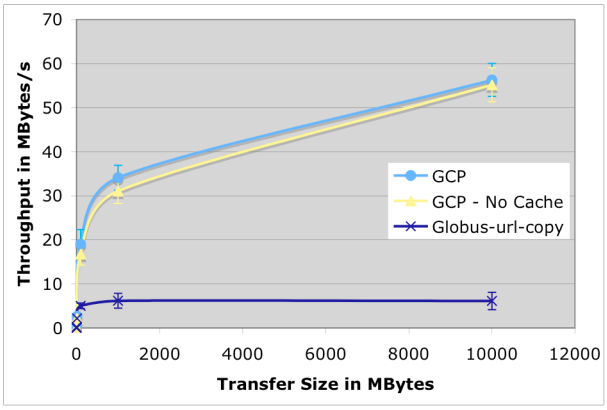

Figure 5: Comparison of performance of disk-to-disk transfers in GCP with GUC over a network with 15 ms RTT.

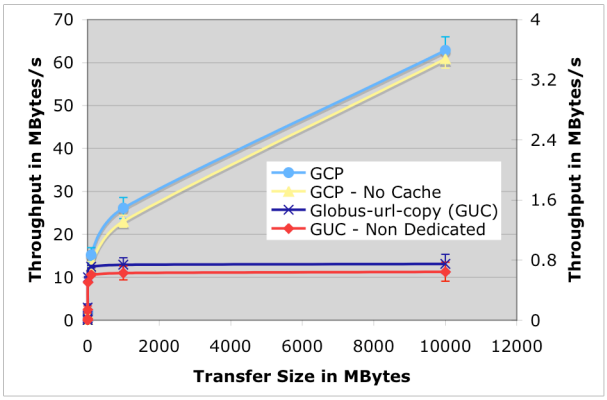

Figure 6: Comparison of performance of disk-to-disk transfers in GCP with GUC over a network with 75 ms RTT.

As observed from Figures 1-6, throughput achieved with 'GCP - No Cache' is slightly less than 'GCP' with caching enabled. The reason for this is that 'GCP - No Cache', for each transfer, runs 'King' to estimate the latency between source and destination, whereas 'GCP' runs 'King' only once for each source, destination pair (and caches the calculated value in the local file system) and picks up the latency value stored in the local file system for subsequent transfers between the same source and destination.

The performance difference for smaller files (≤1 MB) is not clearly visible in Figures 1–6. Thus, we show in Table 1 the performance of GCP and GUC for smaller files over a wide-area network with 15 ms RTT. For files of size greater than or equal to 100 KB, GCP performs better than GUC. For file sizes less than or equal to 10 KB, GUC is slightly better than GCP. The crossover happens somewhere between 10 KB and 100 KB. Similar performance trends are seen for small file transfers over other networks. Because of space constraints, we do not show performance data for small files over other networks.

Table 1: Comparison of performance of small files (files of size ≤1 MB) in GCP with GUC over a network with 15 ms RTT.

XFER SIZE	GCP (B/s)	GCP-NC (B/s)	GUC (B/s)
1KB	283	200	304
10KB	2833	2427	3391
100KB	29596	26929	24489
1M	295819	259634	228358

7. Summary

GridCopy (GCP) provides a simple SCP-style interface that enables users to transfer data efficiently over wide area networks without manual configuration. It allows system administrators to provide appropriate translations so that well-connected nodes can be used to perform fast data transfers. GCP can provide as much as two orders of magnitude improvement in data transfer performance relative to tools that do not perform such automated configuration.

Acknowledgments

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

References

[1] I. Foster, C. Kesselman, and S. Tuecke, "The anatomy of the Grid: Enabling scalable virtual organization," The International Journal of High Performance Computing Applications, vol. 15, no. 3, pp. 200–222, Fall 2001.

[2] I. Foster and C. Kesselman, The Grid: Blueprint for a new computing infrastructure. Morgan Kaufmann Publishers Inc., 1999.

[3] W. Allcock, J. Bester, J. Bresnahan, A. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel, and S. Tuecke, "Data management and transfer in high performance computational grid environments," Parallel Computing Journal, vol. 28, no. 5, pp. 749–771, 2002.

[4] ——, "Secure, efficient data transport and replica management for high-performance data-intensive computing," in 18th IEEE Symposium on Mass Storage Systems, San Diego, California, April 17–20, 2001, pp. 13–28.

[5] W. Allcock, "GridFTP: Protocol extensions to FTP for the grid," Global Grid ForumGFD-R-P.020, 2003.

[6] W. Allcock, A. Chervenak, I. Foster, L. Pearlman, V. Welch, and M. Wilde, "Globus Toolkit support for distributed data-intensive science," in International Conference on Computing in High Energy and Nuclear Physics, Beijing, China, September 2001.

[7] W. Allcock, J. Bresnahan, R. Kettimuthu, M. Link, C. Dumitrescu, I. Raicu, and I. Foster, "The Globus striped GridFTP framework and server," in SC'05, ACM Press, 2005.

[8] S. Tuecke, V. Welch, D. Engert, L. Pearlman, and M. Thompson, "RFC 3820: Internet X.509 Public Key Infrastructure (PKI) Proxy Certificate Profile" June 2004

[9] J. Postel, "RFC 793: Transmission Control Protocol," September 1981

[10] V. Jacobson, "Congestion avoidance and control," ACM SIGCOMM Computer Communication Review, vol. 18, no. 4, pp. 314–329, 1988.

[11] M. Allman, V. Paxson, and W. Stevens, "RFC 2581: TCP Congestion Control," 1999.

[12] S. Floyd and T. Henderson, "RFC 2582: The NewReno Modification to TCP's Fast Recovery Algorithm," 1999.

[13] D. Katabi, M. Handley, and C. Rohrs, "Internet congestion control for future high bandwidth-delay product environments," in ACM SIGCOMM, 2002.

[14] S. Floyd, "HighSpeed TCP for large congestion windows," RFC 3649, Experimental, December 2003.

[15] T. Kelly, "Scalable TCP: Improving performance in highspeed wide area networks," SIGCOMM Computer Communication Review, vol. 33, no. 2, pp. 83–91, 2003.

[16] C. Jin, D. X. Wei, and S. H. Low, "FAST TCP: Motivation, architecture, algorithms, performance," in IEEE Infocom, March 2004.

[17] D. J. Leith and R. Shorten, "H-TCP protocol for high-speed long distance networks," in Second International Workshop on Protocols for Fast Long-Distance Networks, Argonne, IL, Feb, 2004.

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

[19] http://dims.ncsa.uiuc.edu/set/uberftp/

[20] Ravi K. Madduri, Cynthia S. Hood, and William E. Allcock, "Reliable file transfer in Grid environments," in 27th Annual IEEE Conference on Local Computer Networks (LCN 2002), 2002, pp. 737–738

[21] http://www.ping127001.com/pingpage.htm

[22] http://www-iepm.slac.stanford.edu/tools/synack/

[23] http://www.cs.cmu.edu/~hnn/igi/

[24] V. Ribeiro, R. Riedi, R. Baraniuk, J. Navratil, and L. Cottrell, "pathchirp: Efficient available bandwidth estimation for network paths," in Passive and Active Measurement Workshop (2003).

[25] Vinay Ribeiro, Rudolf Riedi, and Richard Baraniuk, "Locating available bandwidth bottlenecks," IEEE Internet Computing, vol. 8, no. 5, pp. 34–41, September–October 2004.

[26] http://www-iepm.slac.stanford.edu/tools/abing/current/_README

[27] http://www-static.cc.gatech.edu/fac/Constantinos.Dovrolis/pathrate.html

[28] http://dast.nlanr.net/Projects/Iperf/

[29] http://www-iepm.slac.stanford.edu/bw/pipechar.html

[30] http://www.kitchenlab.org/www/bmah/Software/pchar/

[31] http://project-iris.net/spruce/

[32] K. P. Gummadi, S. Saroiu, S., and S. D. Gribble, "King: Estimating latency between arbitrary internet end hosts," SIGCOMM Computer Communication Review, vol. 32, no. 3, pp. 5–18, July 2002.

[33] http://www.globus.org/toolkit/mds/#mds_gt4

[34] http://www.pubsub.com/

[35] http://www.teragrid.org/