Applied Techniques for High Bandwidth Data Transfers across Wide Area Networks

Jason Lee, Dan Gunter, Brian Tierney
Computing Sciences Directorate
Lawrence Berkeley National Laboratory
University of California, Berkeley, CA 94720
{jrlee,dkgunter,bltierney}@lbl.gov

Bill Allcock, Joe Bester, John Bresnahan, Steve Tuecke
Mathematics and Computer Science Division
Argonne National Laboratory
9700 South Cass Ave., IL, 60439
{allcock,bester,breshaha,tuecke}@mcs.anl.gov

Abstract

Large distributed systems such as Computational/Data Grids require large amounts of data to be co-located with the computing facilities for processing. Ensuring that the data is there in time for the computation in today's Internet is a massive problem. From our work developing a scalable distributed network cache, we have gained experience with techniques necessary to achieve high data throughput over high bandwidth Wide Area Networks (WAN). In this paper, we discuss several hardware and software design techniques and issues, and then describe their application to an implementation of an enhanced FTP protocol called GridFTP. We also describe results from two applications using these techniques, which were obtained at the Supercomputing 2000 conference.

1.0 Introduction

Large distributed systems such as Computational/Data Grids [10] require large amounts of data to be co-located with the computing facilities for processing. Ensuring that the data is there in time for the computation to start in today's Internet is a massive problem. At LBNL we developed a high-performance striped data cache called the Distributed Parallel Storage System (DPSS)[29]. The idea behind the striped server is several disks on several hosts operate in parallel to supply a very high-speed data stream to one or more clients. The DPSS was specifically optimized for access to large data objects by remote clients over a wide area network.

In the course of designing and using the DPSS we have gained experience with some techniques to achieve high data throughput over the WAN. In addition, some general lessons have been learned on the proper configuration of hardware. The following is a brief summary of these techniques, several of which are described in more detail in later sections of this paper:

- use parallel everything (i.e.: servers, disks, SCSI controllers, network cards)
- use tuned TCP buffers, optimally set for each client
- use parallel streams, but only if the receive host is powerful enough
- use asynchronous and overlapped network and disk I/O
- no user-space data copying allowed (manipulate all data buffers via pointers)

- provide a simple client API that is as similar to the POSIX I/O interface as possible.

LBNL and ANL worked together to abstract these techniques from the DPSS and apply them to the development of a high-throughput striped server which interfaces to the outside world using an enhanced version of the File Transfer Protocol (FTP)[21] called GridFTP [11]. In this paper, we will use both the DPSS and GridFTP to illustrate implementation issues related to the techniques, and the preliminary results from a very high bandwidth WAN test of both storage servers at SuperComputing 2000.

1.1 Motivation

In scientific computing environments, small clusters of PC's running Linux are becoming more common, and these will likely become an important part of the future data intensive computing landscape. Large PC clusters running data intensive applications will likely use a Storage Area Network (SAN) with multiple high performance Redundant Array of Inexpensive Disk (RAID)[20] systems as data servers. Although considerably cheaper than the previous large RAID-based solutions, this type of hardware configuration is still relatively expensive because it requires the purchase of Fibre Channel switches and interface cards[7]. For large clusters the costs will be amortized over many nodes, and perhaps their required performance will be unattainable by other means, but sites with small clusters will probably not be able to afford this.

Although small clusters most likely cannot afford an expensive SAN/RAID system, they do need better performance than that provided by the typical solution today, which is a medium strength (e.g. 4 CPU) NFS server connected to a single RAID system. This is certain to be a bottleneck for some data intensive applications. In order to attain the necessary performance at a low cost, we argue that a set of striped servers composed of commodity hardware and software, and running over the existing high-speed LAN, should be used. The scalability and price/performance ratio of striped commodity servers make them an excellent solution for this environment

For example, a one terabyte data set might be staged from a tape system such as HPSS to a striped cache system. If the data is striped across 8 data cache hosts, then a 32 node cluster would receive up to 8 times more I/O bandwidth than it would using a single data server.

In addition to high-throughput from data cache to cluster, high-throughput from data cache to data cache across a WAN is also very important. There are several scientific research communities that need the ability to copy and/or replicate data quickly between disk caches at multiple sites [5][14]. A striped data server that is optimized for WAN data transfers is ideal for this environment.

2.0 Techniques for High Performance

Several techniques for achieving high-performance from storage servers in a WAN environment are presented below. It is important to note that these techniques are not additive, but rather complementary. Applying a single technique may have little or no effect, because the absence of any one of the techniques can create a bottleneck.

2.1 Tuned TCP Buffers

The standard transport layer in use today is the Transport Control Protocol (TCP) [27]. TCP uses what it calls the “congestion window”, or CWND, to determine how many packets can be sent before waiting for an acknowledgement. The larger the congestion window size, the higher the throughput. This follows directly from Little’s Law[17] that (average throughput)*(delay) = window size. The TCP “slow start” and “congestion avoidance” algorithms determine the size of the congestion window [16]. The maximum congestion window is proportional to the amount of buffer space that the kernel allocates for each socket. For each socket, there is a default size for this buffer, which can be changed by the program (using a system library call) just before opening the socket. There is also a kernel-enforced maximum buffer size. The buffer size can be adjusted for both the send and receive

ends of the socket.
To get maximal throughput it is critical to use optimal TCP send and receive socket buffer sizes for the link you are using. If the buffers are too small, the TCP congestion window will never fully open up. If the buffers are too large, the sender can overrun the receiver, and the TCP window will shut down. The optimal buffer size of a link is (bandwidth) * (round trip time (RTT)), where RTT equals twice the one-way delay on the link. For an explanation of why this is true, see [22] and [29].
For example, if your RTT is 50 ms, and the end-to-end network consists of all 100BT ethernet or higher, the TCP buffers should be 0.05 sec * 10 MB/sec = 500 KBytes. Two TCP settings need to be considered: The **default** TCP send and receive buffer size, and the **maximum** TCP send and receive buffer size. Note that most of today's UNIX operating systems ship with a maximum TCP buffer size of only 256 KB (and the default maximum for Linux is only 64 KB!). The maximum buffer size need only be set once. However, since setting the default TCP buffer size greater than 128 KB will adversely affect LAN performance, the UNIX *setsockopt* call should be used in your sender and receiver to set the optimal buffer size for the link you are using. For more information on setting the default, maximum, and optimal TCP buffers, consult the LBNL "TCP Tuning Guide"[26].

2.2 Parallel Streams

The design of TCP has been influenced much less by the high-performance community than by the demands of the Internet, and in particular by the need to enforce fair sharing of precious network resources. For this reason, TCP's behavior is unnecessarily conservative for data-intensive applications on high bandwidth networks.
TCP probes the available bandwidth of the connection by continuously increasing the window size until a packet is lost, at which point it cuts the window in half and starts "ramping up" the connection again. The higher the bandwidth-delay product, the longer this ramp up will take, and less of the available bandwidth will be used during its duration. When the window of the bottleneck link is large enough to keep the pipe full during the ramp up, performance does not degrade. However this requires large buffers on all intervening routers. Furthermore, when there is random loss on the connection, it has been shown [16] that the link utilization is proportional to $q(ut)^2$, where q is the probability of loss and ut is the bandwidth-delay product.
In order to improve this situation where the network becomes the bottleneck, parallel streams can be used. This technique is implemented by dividing the data to be transferred into N portions and transferring each portion with a separate TCP connection. The effect of N parallel streams is to reduce the bandwidth-delay product experienced by a single stream by a factor of N because they all share the single-stream bandwidth (u). Random packet losses for reasonable values of q (<0.001) will usually occur in one stream at a time, therefore their effect on the aggregate throughput will be reduced by a factor of N. When competing with connections over a congested link, each of the parallel streams will be less likely to be selected for having their packets dropped, and therefore the aggregate amount of potential bandwidth which must go through premature congestion avoidance or slow start is reduced. It should be noted, however, that if the bandwidth is limited by small router buffers in the path, all the streams are likely to experience packet loss in synchrony (when the buffer fills, arriving packets from each stream are all dropped) and thus gain little advantage over a single stream.
Experience has shown that parallel streams can dramatically improve application throughput, (see [23] and [29]), and can also be a useful technique for cases where you don't have *root* access to a host in order to increase its maximum TCP buffer size. However, parallel streams can drastically reduce throughput if the sending host is much faster than the receiving host. For example, we have seen a 50% loss in aggregate throughput of 2 streams versus 1 stream on a Linux 2.2.14 receive host with a NetGear 1000BT card using a multi-threaded receiver.

2.3 Striped Disks and Servers

In order to aggregate the potential throughput of numerous hosts, each with one or more disk controllers and several disks per controller, the data being transferred should be subdivided into "stripes" and spread evenly across the servers and disks. Different software or hardware systems may be responsible for striping at different levels of the storage hierarchy. For example, a RAID system may stripe across the file system on a host, while a separate server stripes across all the hosts. The important point is to make sure that the striping occurs at all levels. Thus the disks can saturate the disk controllers, the disk controllers can saturate the network interface card (NIC), and the NICs can saturate the router.

Placement algorithms affect the parallelism of the system. A good stripe placement for sequential operations, i.e. for data access patterns with a high temporal locality, is round-robin. For random-access data sets with a high spatial locality (i.e. several widely spaced areas of the dataset are accessed in parallel), partitioning the file into one contiguous sequence of stripes per disk may improve performance.

The size of the stripe must balance the need to evenly distribute the data across the storage resources (smaller is better) with the need to perform efficient low-level I/O to both disk and network (bigger is better). Although the exact size of the stripe may not be critical, small numbers might lead to a horrible bottleneck. For example, see the results in Table 1. For random access, large disk reads can dramatically improve disk throughput.

It is important to have enough parallel disks on each server to saturate the network under non-sequential access patterns. For example, if you are using a 64KB stripe size and the same SCSI disk that was used for the results in Table 1, and your server network card has a maximum throughput of 40 MB/s, then you will need 8-9 parallel disks to saturate the NIC.

3.0 Hardware Configuration Issues

The simple employment of these techniques in an application does not guarantee the absence of performance bottlenecks. Interaction patterns across the hardware can play an important role in creating performance problems in an application. In the following sections we will try to examine and summarize some of the technical issues that were encountered during the development of the techniques.

Access Method	1 KB blocks	8 KB blocks	64 KB blocks	128 KB blocks
sequential	18.2 MB/s	18.5 MB/s	22.7 MB/s	22.7 MB/s
random	.08 MB/s	2.2 MB/s	4.6 MB/s	8.0 MB/s
speedup	231	13.3	4.9	2.7

3.1 Disks and Controllers

In today's high-speed computing environment it is important to ensure that both your disks and controllers are not simply fast enough, but well matched (properly configured and tuned). Improperly configured hardware can cause unnecessary thrashing of system resources. In a bottom up approach to tuning the I/O subsystem one should first start by testing the disks, then the controllers and then move up through the system till you reach host adapters.

The disks should be of equal size and speed. The state of the system is limited by its slowest component, therefore a slower disk will constrain the performance of a striped file system. Smaller disks will skew the striping performance because the larger disk will be accessed more often, instead of striping the data equally across all the disks.

Once a decision has been made on the which hardware to use, each component of the system should be tested. First, test the speed of a single disk, in isolation. Next, add a second disk and check the speed of the two disks being accessed simultaneously. With the addition of each disk, the aggregate speed should increase by the speed of a single disk. Once the aggregate throughput stops increasing, the limit of the disk controller has probably been reached; if the throughput is not enough to saturate the NIC, another disk controller will need to be added, and the process of adding disks should continue.

For example, with a NIC on Gigabit Ethernet (~300Mb/s), a SCSI controller at 160Mb/s, and SCSI disks as shown in Table above, to saturate the NIC for random access reads there will have to be 2 controllers and at least 2 disks per controller.

3.2 Networking and Processors

As the network, disk and memory speeds all increase, the speed, power and number of CPU's in the system become an increasingly important factor. Many of the newer high-speed network cards now require a significant amount of CPU power to drive them. The number of interrupts that are delivered to the OS when running at gigabit speeds can overwhelm slower CPU's. There are several ways to lower the load on the CPU:

- Interrupt coalescing; the network card packages several TCP packets together before interrupting the OS
- TCP checksumming on the network card, instead of in software on the host computer
- Larger Maximum Transmission Unit (MTU)

All three of these techniques attempt to accomplish the same goal: reduce the per-packet processing overhead. From [6], "Smaller frames usually mean more CPU interrupts and more processing overhead for a given data transfer size. Often the per-packet processing overhead sets the limit of TCP performance...".

It should be noted that not all gigabit network interfaces support all these options, or will interoperate with other cards or switches. Most notably when using a larger MTU (sometimes referred to as a Jumbo Frame) packets may not be able to cross some networks or interoperate with some switches.

We tested several different vendors' cards, and found a large degree of variance in how they performed, depending on variables such as the PCI bus width (64 vs. 32bit), how much memory was on the card, what driver/OS were used to drive the card. For instance, by changing from Linux kernel version 2.2 to version 2.4, our local area *iperf* throughput values rose from approximately 320 Mb/s to just over 500 Mb/s using the same hardware. In conclusion, one should always test out the specific cards in the environment that will be used in production.

3.3 Implementation

We have made two very different implementations of the various techniques that we have described in this paper. The first one is the DPSS, which uses a custom API and was created especially for use in a WAN environment. Secondly, groups at ANL and LBNL worked together to design a system that uses techniques learned while developing the DPSS and applies these techniques to build a more general purpose high-performance FTP server. This 'enhanced' version of FTP. called GridFTP, supports striped servers, parallel streams, and tuned TCP window buffers, and was developed in conjunction with ANL's data transfer libraries [8].

Both the DPSS and the GridFTP server operated on an identical hardware configuration. Typical striped server implementations consist of several low-cost Unix workstations as data block servers, each with several disk controllers, and several disks on each controller. A four-server system with a capacity of one Terabyte (costing about $10-$12K in late-2000) can produce throughputs of over 70

MB/s by providing parallel access to 20-30 disks.

3.4 DPSS

The main features of the DPSS are described in the first section and include but are not limited to: highly parallel, tunable TCP buffers, and asynchronous I/O. During DPSS usage, requests for blocks of data are sent from the client to the "DPSS master" process, which determines which "DPSS block servers" the blocks are located on, and forwards the requests to the appropriate servers. The server then sends the block directly back to the client.

The application interface to the DPSS is through either a low level "block" API, or a higher level POSIX-like API. The data layout on the disks is completely up to the application, and the usual strategy for sequential reading applications is to write the data round-robin, striping blocks of data across the servers. The DPSS client library is multi-threaded, where the number of client threads is equal to the number of DPSS servers. Therefore, the speed of the client is scaled with the speed of the server, assuming the client host is powerful enough.

3.5 GridFTP

GridFTP consists of extensions to the FTP protocol to provide features necessary for a Grid environment. Use of a common protocol provides interoperability: GridFTP can communicate with any existing FTP server, or any new implementation that follows the extended FTP protocol.

Most current FTP implementations support only a subset of the features defined in the FTP protocol and its accepted extensions. Some of the seldom-implemented features are useful to Grid applications, but the standards also lack several features Grid applications require. We selected a subset of the existing FTP standards and further extended them, adding the following features: Security (both Grid Security Infrastructure (GSI) and Kerberos support), Parameter set/negotiate, which allows interoperability with existing FTP implementations, Third party transfers (server to server), parallel transfers (multiple TCP streams per transfer), striped transfers (multiple host to multiple host), partial file transfers, and flexible reliability / recovery functionality via a plug-in interface. The actual protocol extensions are beyond the scope of this paper, but a proposed draft submitted to the Grid Forum may be reviewed at http://www.gridforum.org.

GridFTP can be used for bulk data transfer as in this application, or can be used as a data access mechanism with semantics very similar to Unix open/close/seek/read/write from an application perspective. For this implementation, the physical mechanism employed is essentially a map of the file into a pool of 64KB blocks. The distribution of these blocks is user selectable and may be either partitioned (the file is divided into *n* pieces and one piece is stored on each node, where *n* is the number of nodes) or round robin.

To initiate a transfer, third party in this case, a control connection is established between the application and the master server at each site. These master servers form control connections to the back end servers. Once these connections are established, a fairly standard FTP protocol exchange takes place between the application and the master servers. The master forwards the commands to the back end servers and condense the individual responses into a single response that is then sent to the application. The back end servers check a local database to determine which blocks, if any, they have, they then establish data connections with the appropriate source/destination server, with the specified level of parallelism, and execute the transfer.

4.0 Results

Gaining access to the next generation of high-speed networks in order to explore the techniques outlined above is difficult. We were able to participate in a contest called the "Bandwidth Challenge" at SuperComputing 2000, which used a time-shared OC-48 (2.4 Gb/s) network path over NTON [19] and SuperNet [24] from LBNL in Berkeley, CA to the conference show floor in Dallas, TX, as shown in [3]. Both the DPSS -- as a server for an application called Visapult [3] -- and GridFTP

participated in the contest, and each had exclusive access to this network path for a one hour run. We were able to monitor the router in Berkeley during both runs. In this section, we will briefly describe each application, and analyze their results. Due the transient nature of the network, we did not have time to run more controlled experiments, so the degree to which these results characterize general performance characteristics of either the DPSS or GridFTP is uncertain, and will be the subject of future work.

The servers for both the DPSS and GridFTP were identical hardware, which consisted of: 4 dual-processor Linux boxes and 4 single-processor Solaris boxes, all with SCSI disks and a single controller, running over Gigabit Ethernet.

4.1 Visapult / DPSS Results

The LBNL entry in the challenge was a visualization application, Visapult, that transforms scientific simulation data in parallel on a compute node, then transmits it in parallel over the network for rendering. The dataset, 80GB in size, was stored on the DPSS at LBNL and the compute cluster, an 8-processor SGI Origin with 4 Gigabit Ethernet interfaces, was in Dallas. This process of parallel reading of a large dataset from a distant location while transforming it on a powerful compute node

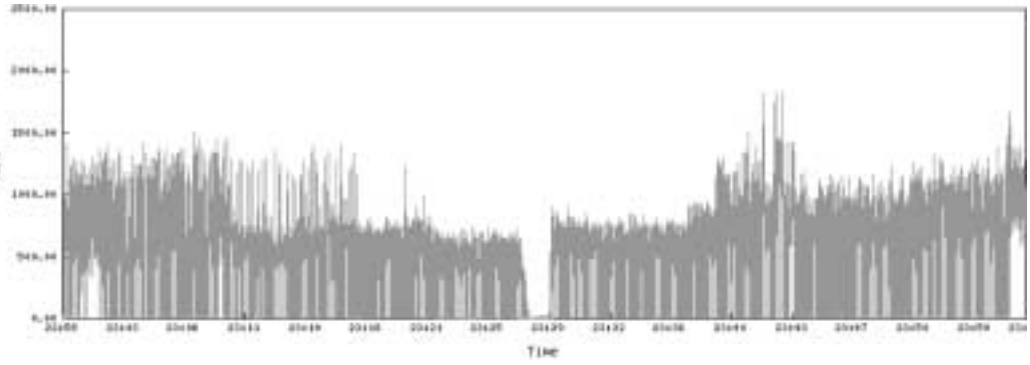

Figure 1: Throughtput at SC200 router during Visapult Bandwidth Challenge run

is common in high energy physics (HEP) applications. During the course of the contest, background DPSS *get* operations were run to use up the spare DPSS bandwidth (roughly 500 Mb/s) due to a bottleneck at Visapult's rendering engine. The DPSS obtained a peak of 1.48 Gb/s and sustained throughput of 582 Mb/s, as measured at the ingress router to the show floor.

A graph of 5-second polls of the router packet counts is shown in Figure 1. The graph shows the router throughput over time. Because the test had 64 streams (8 nodes x 8 processors on the SGI), the CWND was only about 200KB, instead of the 13MB a single stream would have required. This allowed us to better utilize the network and adapt to it.

4.2 GridFTP Results

The ANL entry in the network challenge was a climate modeling application. This application is representative of a wide range of data intensive applications. These applications require large amounts of data and, to reduce network overhead, often employ replication to make the data access more efficient and less costly. ANL, through the Globus project, provides the infrastructure required for such applications. During our hour of dedicated network access we were transferring data to create a new replica of climate data at LBNL. We were able to transfer 230.8 GB of data, for an agregrate data transfer rate of 512.9 Mb/s, with peaks of greater than 1 Gb/s over 5 second intervals.

On the show floor we had an eight node single CPU Linux cluster each equipped with 4 SCSI disks and a Gigabit Ethernet adapter. These were connected to a Cisco switch with dual bonded GigE out to the show floor routers, then via OC-48 to LBNL. The data files were 2 GB in size and were arranged using the partitioned layout on both source and destination. Each node used a parallelism of four (4 TCP streams for its share of the data), and there were as many as four files being transferred simultaneously. This resulted in a maximum of 128 total data streams (8 nodes x 4 streams x 4 files).

We performed 5-second polls to the LBNL router. A graph of the router throughput, smoothed with a 10-point averaging window to make trends in the data clearer, is shown below in Figure 2. The data transfer for this application was very uneven and "bursty" for two reasons. One is the nature of the transfer. A 2 GB file can be transferred in about 20 seconds and then the data connections must be

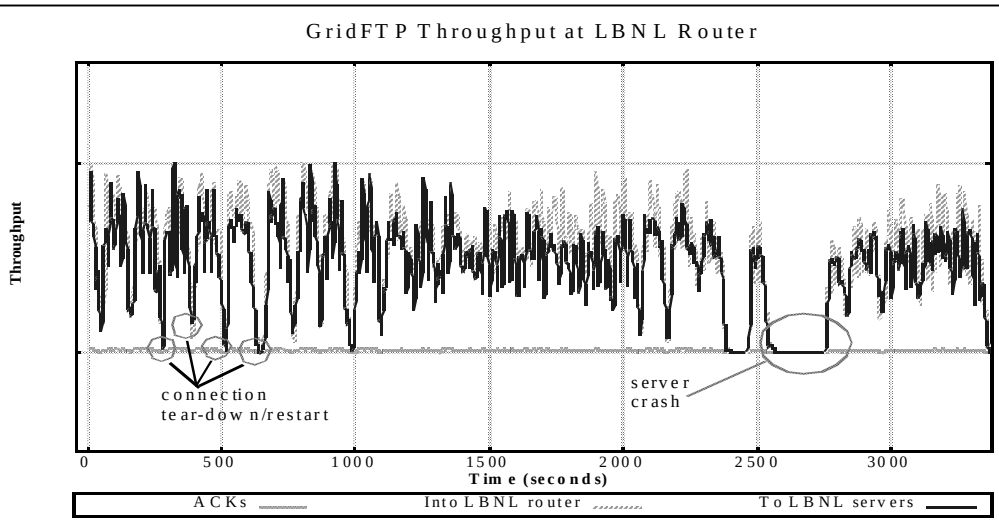

Figure 2

torn down and a new transfer started. This causes spikes in the data transmission rate, especially when all four files ended at approximately the same time. We also had a several minute period where one of the receiving servers had crashed and we had to reboot, and restart the transfer.

5.0 Conclusions

The techniques described in this paper and implemented in both GridFTP and the DPSS will be needed to realize the potential of next generation high bandwidth networks. However, use of these techniques still requires extra effort and knowledge usually not available to the application programmer. We feel that the example implementations here show not only how to use these techniques, but also how these techniques can be accessed in a fashion that is not much different then that of a local standard file access, while at the same time taking full advantage of a high speed wide area network.

The basic functionality of GridFTP is currently in place. The code is in late alpha testing and should be going to beta soon. When released it will be available under the Globus public license at http://www.globus.org. As a result of our experiences at SC 2000 we have already made 2 small, but important improvements to our current implementation. We have added 64 bit file support for larger than 2 GB files, and we have added data channel caching. The data channel caching will be particularly useful since it will avoid the overhead of setup and tear down of the sockets, which can be significant, particularly when authentication is enabled on the data channels. We are also going to

explore the possibility of implementing our striped server on top of a parallel virtual file system.
The DPSS project is now fully functional and can be downloaded from http://www-didc.lbl.gov/DPSS. While we are still making minor adjustments to the DPSS, we are mostly interested in looking at how high bandwidth data transfer needs can be integrated with higher-level services. We are investigating integration efforts in the areas of file replication, staging, caching, and location transparency. In addition, we are considering the use of dynamic data from performance monitoring as a feedback mechanism to a live transfer. We feel that monitoring data can contribute on several levels, such as what link to use, what storage resource to use in a replicated data set, and whether to move the data to the computation or vice-versa.

6.0 Acknowledgments

This work was supported by the Director, Office of Science. Office of Advanced Scientific Computing Research. Mathematical, Information, and Computational Sciences Division under U.S. Department of Energy Contract No. DE-AC03-76SF00098. This is report no. LBNL-47183.

7.0 References

[1] B. Bershad et. al., “The Scalable I/O Initiative”, white paper, available through the Concurrent Supercomputing Consortium, CalTech, Pasadena, CA Feb. 1993, http://www.cacr.caltech.edu/SIO/

[2] Bethel, et. al., “Bandwidth Statistics Reported by SciNet Router”, http://www-didc.lbl.gov/presentations/SC00.LBNL.netchallenge.pdf, Slide 5, November 2000

[3] Bethel, W., Tierney, B., Lee, J., Gunter, D., Lau, S., “Using High-speed WANs and Network Data Caches to Enable Remote and Distributed Visualization”, Proceedings of the IEEE SuperComputing 2000 Conference, Nov. 2000, LBNL-45365

[4] Carns, P., Ligon III, Ross, R., Thakur, R., “PVFS: A Parallel File System For Linux Clusters”, Proceedings of the 4th Annual Linux Showcase and Conference, Atlanta, GA, October 2000, pp. 317-327

[5] The DataGrid Project: http://www.cern.ch/grid/

[6] Dykstra, P., “Gigabit Ethernet Jumbo Frames”, http://www.columbia.edu/acis/networks/advanced/jumbo/jumbo.html

[7] “Fibre Channel · Overview of the Technology”, Fibre Channel Industry Association (FCIA), http://www.fibrechannel.com/technology/overview.html

[8] Globus: http://www.globus.org

[9] Goland, Y. et. al, “HTTP Extensions for Distributed Authoring -- WEBDAV”, IETF RFC 2518, Feb. 1999

[10] The Grid: Blueprint for a New Computing Infrastructure”, edited by Ian Foster and Carl Kesselman. Morgan Kaufmann, Pub. August 1998. ISBN 1-55860-475-8.

[11] GridFTP: Universal Data Transfer for the Grid, White Paper, http://www.globus.org/datagrid/deliverables/C2WPdraft3.pdf

[12] Hartman, J., Murdock, I., Spalink, T., “The Swarm Scalable Storage System”, Proceedings of the 19th IEEE International Conference on Distributed Computing Systems, June 1999.

[13] Hartman, J., Ousterhout, J, “The Zebra Striped Network File System”, ACM Transactions on Computer Systems 13, 3, August 1995, 279-310.

[14] Hoschek, W., J. Jaen-Martinez, A.Samar, H. Stockinger, K. Stockinger, “Data Management in International Data Grid Project”, to IEEE, ACM International Workshop on Grid Computing (Grid'2000), Bangalore, India, 17-20 Dec. 2000.

[15] Hwang, K, J. Jin, P. Novaux, “RAID-x: A New Distributed Disk Array for I/O Centric Cluster Computing”, In Proc. 9th IEEE Symp. on High Performance Distributed Computing, Aug 2000

[16] Jacobson, V., “Congestion Avoidance and Control,” Proceedings of ACM SIGCOMM ‘88, August 1988.

[17] Kleinrock, L., Queueing Systems, Vols. I&II, J. Wiley and Sons, 1975.

[18] Lakshman, T., Madhow, U., “The performance of TCP/IP networks with high bandwidth-delay products and random loss”, IEEE Transactions on Networking, vol. 5 no 3, pp. 336-350, July 1997

[19] National Transparent Optical Network (NTON) http://www.ntonc.org/

[20] Patterson, David A., G. Gibson, R. Katz, “A Case for Redundant Arrays fo Inexpensive Disks (RAID). In International Conference on Management of Data (SIGMOD), pages 109-116, June 1988

[21] Postel, J. and Reynolds, J., “File Transfer Protocol (FTP)”, IETF RFC 959, October 1985

[22] Semke, J. Mahdavi, M. Mathis, “Automatic TCP Buffer Tuning,” Computer Communication Review, ACM SIGCOMM, volume 28, number 4, Oct. 1998.

[23] Sivakumar, H, S. Bailey, R. L. Grossman, “PSockets: The Case for Application-level Network Striping for Data Intensive Applications using High Speed Wide Area Networks”, Proceedings of IEEE Supercomputing 2000, Nov., 2000. http://www.ncdm.uic.edu/html/psockets.html)

[24] SuperNet Network Testbed Projects: http://www.ngi-supernet.org/

[25] Shen, X, A. Choudhary, “A Distributed Multi-Storage Resource Architecture and I/O Performance Prediction for Scientific Computing”, In Proc 9th IEEE Symp. on High Performance Distributed Computing, Aug 2000

[26] “TCP Tuning Guide”, http://www-didc.lbl.gov/tcp-wan.html

[27] Transmission Control Protocol (TCP), IETF RFC 793, September 1981

[28] Tierney, B., Johnston, W., Crowley, B., Hoo, G., Brooks, C., Gunter, D., “The NetLogger Methodology for High Performance Distributed Systems Performance Analysis”, Proceedings of the IEEE High Performance Distributed Computing conference (HPDC-7), July 1998, LBNL-42611

[29] Tierney, B. J. Lee, B. Crowley, M. Holding, J. Hylton, F. Drake, “A Network-Aware Distributed Storage Cache for Data Intensive Environments”, Proceeding of IEEE High Performance Distributed Computing conference (HPDC-8), August 1999, LBNL-42896. http://www-didc.lbl.gov/DPSS/

[30] Watson, R., Coyne, R.,“The Parallel I/O Architecture of the High-Performance Storage System (HPSS)”, IEEE MS Symposium, 1995