A CACHE-BASED DATA INTENSIVE DISTRIBUTED COMPUTING ARCHITECTURE FOR “GRID” APPLICATIONS

Brian Tierney, William Johnston, Jason Lee
Lawrence Berkeley National Laboratory, Berkeley, CA 94720

Abstract
Modern scientific computing involves organizing, moving, visualizing, and analyzing massive amounts of data from around the world, as well as employing large-scale computation. The distributed systems that solve large-scale problems will always involve aggregating and scheduling many resources. Data must be located and staged, cache and network capacity must be available at the same time as computing capacity, etc. Every aspect of such a system is dynamic: locating and scheduling resources, adapting running application systems to availability and congestion in the middleware and infrastructure, responding to human interaction, etc. The technologies, the middleware services, and the architectures that are used to build useful high-speed, wide area distributed systems, constitute the field of data intensive computing, and are sometimes referred to as the “Data Grid”. This paper explores the use of a network data cache in a Data Grid environment.

1. INTRODUCTION

High-speed data streams resulting from the operation of on-line instruments and imaging systems are a staple of modern scientific, health care, and intelligence environments. The advent of high-speed networks is providing the potential for new approaches to the collection, organization, storage, analysis, visualization, and distribution of the large-data-objects that result from such data streams. The result will be to make both the data and its analysis much more readily available.

For example, high energy physics experiments generate high rates and massive volumes of data that must be processed and archived in real time. This data must also be accessible to large scientific collaborations — typically hundreds of investigators at dozens of institutions around the world.

In this paper we will describe how “Computational Grid” environments can be used to help with these types of applications, and give a specific example of a high energy physics applications in this environment. We describe how a high-speed application-level network data cache is a particularly important component in a data intensive grid architecture, and describe our implementation of such a cache.

2. DATA INTENSIVE GRIDS

The integration of the various technological approaches being used to address the problem of integrated use of dispersed resources is frequently called a “grid,” or a computational grid — a name arising by analogy with the grid that supplies ubiquitous access to electric power. See, e.g., [9]. Basic grid services are those that locate, allocate, coordinate, utilize, and provide for human interaction with the various resources that actually perform useful functions.

Grids are built from collections of primarily independent services. The essential aspect of grid services is that they are uniformly available throughout the distributed environment of the grid. Services may be grouped into integrated sets of services, sometimes called “middleware.” Current grid tools include Globus [8], Legion [15], SRB [2], and workbench systems like Habanero [10] and WebFlow [1]. Recently the term “Data Grid” has come into use to describe middleware and services for data intensive Grid applications [3], and several data grid research projects have be started [5][[17].

From the application's point of view, the Grid is a collection of middleware services that provide applications with a uniform view of distributed resource components and the mechanisms for assembling them into systems. From the middleware systems points of view, the Grid is a standardized set of basic services providing scheduling, resource discovery, global data directories, security, communication services, etc. However, from the Grid implementor's point of view, these services result from and must interact with a heterogeneous set of capabilities, and frequently involve "drilling" down through the various layers of the computing and communications infrastructure.

2.1 Architecture for Data Intensive Environments

Our model is to use a high-speed data storage cache as a common element for all of the sources and sinks of data involved in high-performance data systems. We use the term "cache" to mean storage that is faster than typical local disk, and temporary in nature. This cache-based approach provides standard interfaces to a large, application-oriented, distributed, on-line, transient storage system. In a wide-area Grid environment, the caches must be specifically designed to achieve maximum throughput over high-speed networks.

Each data source deposits its data in the cache, and each data consumer takes data from the cache, often writing the processed data back to the cache. A tertiary storage system manager migrates data to and from the cache at various stages of processing. (See Figure1.) We have used this model for data handling systems for high energy physics data and for medical imaging data. These applications are described in some detail in [14] and [13].

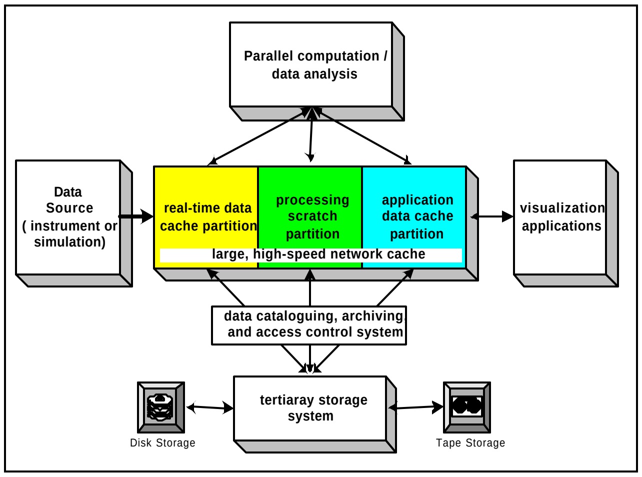
Parallel computation / data analysis
Data Source (instrument or simulation)
real-time data cache partition
processing scratch partition
application data cache partition
large, high-speed network cache
visualization applications
data cataloguing, archiving and access control system
tertiaray storage system
Disk Storage
Tape Storage

Figure 1 The Data Handling Model

The high-speed application-level cache serves several roles in this environment. It provides a standard high data rate interface for high-speed access by data sources, processing resources, mass storage systems (MSS), and user interface / data visualization elements. It provides the functionality of a single very large, random access, block-oriented I/O device (i.e., a "virtual disk"). This cache also serves to isolate the application from tertiary storage systems and instrument data sources, helping eliminate contention for those resources.

This cache can be used as a large buffer, able to absorb data from a high rate data source and then to forward it to a slower tertiary storage system. The cache also provides an "impedance matching" function between a small number of high throughput streams to a larger number of lower speed streams, e.g. between fine-grained accesses by many applications and the coarse-grained nature of a few parallel tape drives in the tertiary storage system.

Depending on the size of the cache relative to the objects of interest, the tertiary storage system management may only involve moving partial objects to the cache. In other words, the cache may contain a moving window for an extremely large off-line object/data set. Generally, the cache storage configuration is large (e.g., 100s of gigabytes) compared to the available disks of a typical computing environment (e.g., 10s of gigabytes), and very large compared to any single disk (e.g. hundreds of ~10 gigabytes).

In this type of environment, a client typically will copy large portions of a data set from the remote archive to local disk before visualizing or processing the data. However, if the network is fast enough and if the cache is tuned for remote access and uses parallel disks, the cache can provide data access to remote clients that is even faster than local disk. Additionally many applications actually only need a small portion of the total data set. By leaving the data on a remote cache, a much smaller amount of data may actually be moved over the network.

3. THE DISTRIBUTED-PARALLEL STORAGE SYSTEM

Our implementation of this high-speed, distributed cache is called the Distributed-Parallel Storage System (DPSS) [20]. LBNL designed and implemented the DPSS as part of the DARPA MAGIC project [6], and as part of the U.S. Department of Energy's high-speed distributed computing program. This technology has been successful in providing an economical, high-performance, widely distributed, and highly scalable architecture for caching large amounts of data that can potentially be used by many different users.

Typical DPSS implementations consist of several low-cost workstations as DPSS block servers, each with several disk controllers, and several disks on each controller. A four-server DPSS with a capacity of one Terabyte (costing about $10-$12K in mid-2000) can thus produce throughputs of over 70 MBytes/sec by providing parallel access to 20-30 disks.The overall architecture of the DPSS is illustrated in Figure2.

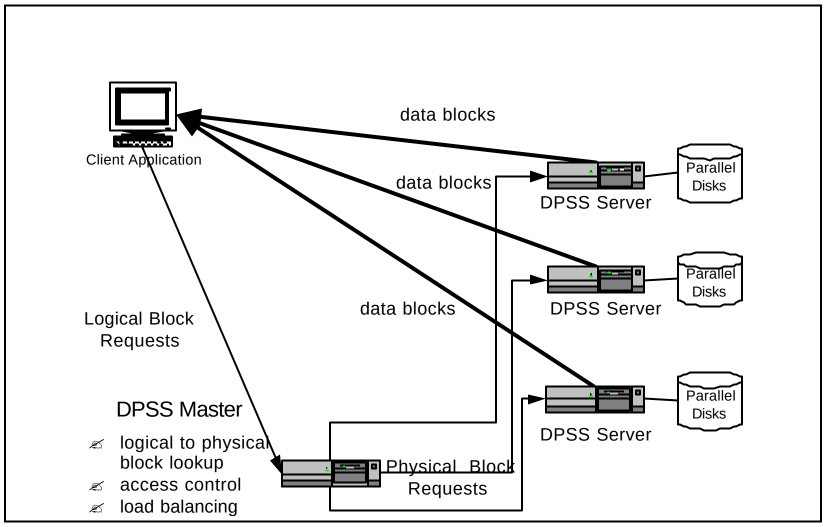

Figure 2 Overall DPSS Architecture

Other papers describing the DPSS in more detail include [20], which describes how the DPSS was used to provide high-speed access to remote data for a terrain visualization application, [21], which describes the basic architecture and implementation, and [22], which describes how the instrumentation abilities in the DPSS were used to help track down a wide area network problem.

The application interface to the DPSS cache supports a variety of I/O semantics, including Unix-like I/O semantics, through an easy to use client API library (e.g. dpssOpen(), dpssRead(), dpssWrite(), dpssLSeek(), dpssClose()). 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 client library also includes a flexible data replication ability, allowing for multiple levels of fault tolerance. 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 scales with the speed of the server, assuming the client host is powerful enough.

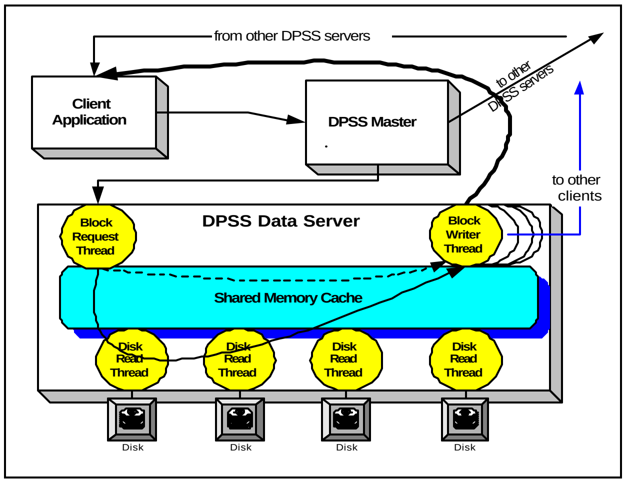

Figure 3 DPSS Server Architecture

The internal architecture of the DPSS is illustrated in Figure3. 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. Servers may be anywhere in the network: there is no assumption that they are all at the same location, or even the same city.

DPSS performance, as measured by total throughput, is optimized for a relatively smaller number (a few thousand) of relatively large files (greater than 50 MB). Performance is the same for any file sizes greater than 50 MB. We have also shown that performance scales well with the number of clients, up to at least 64 clients. For example, if the DPSS system is configured to provide 50 MB/sec to 1 client, it can provide 1 MB/sec to each of 50 simultaneous clients. The DPSS master host starts to run out of resources with more than 64 clients.

Because of the threaded nature of the DPSS server, a server scales linearly with the number of disks, up to the network limit of the host (possibly limited by the network card or the CPU). The total DPSS system throughput scales linearly with the number of servers, up to at least 10 servers.

The DPSS provides several important and unique capabilities for data intensive distributed computing environments. It provides application-specific interfaces to an extremely large space of logical blocks; it offers the ability to build large, high-performance storage systems from inexpensive commodity components; and it offers the ability to increase performance by increasing the number of parallel disk servers.

DPSS data blocks are available to clients immediately as they are placed into the cache. It is not necessary to wait until the entire file has been transferred before requesting data. This is particularly useful to clients requesting data from a tape archive. As the file moves from tape to the DPSS cache, the blocks in the cache are immediately available to the client. If a block is not available, the application can either block, waiting for the data to arrive, or continue to request other blocks of data which may be ready to read.

3.1 TCP Issues

The DPSS uses the TCP protocol for data transfers. For TCP to perform well over high-speeds networks, it is critical that there be enough buffer space for the congestion control algorithms to work correctly [11]. Proper buffer size is a function of the network bandwidth-delay product, but because bandwidth-delay products in the Internet can span 4-5 orders of magnitude, it is impossible to configure the default TCP parameters on a host to be optimal for all connections [18].

Figure4 shows the importance of the setting of the TCP buffer size correctly. This figure illustrates that buffers can be hand-tuned for either LAN access or WAN access, but not both at once. It is also apparent that while setting the buffer size big enough is particularly important for the WAN case, it is also important not to set it too big for the LAN environment. If the buffers are too large, throughput may decrease because the larger receive buffer allows the congestion window to grow sufficiently large that multiple packets are lost (in a major buffer overflow) during a single round trip time (RTT), which then leads to a time-out instead of a smooth fast retransmit/recovery.

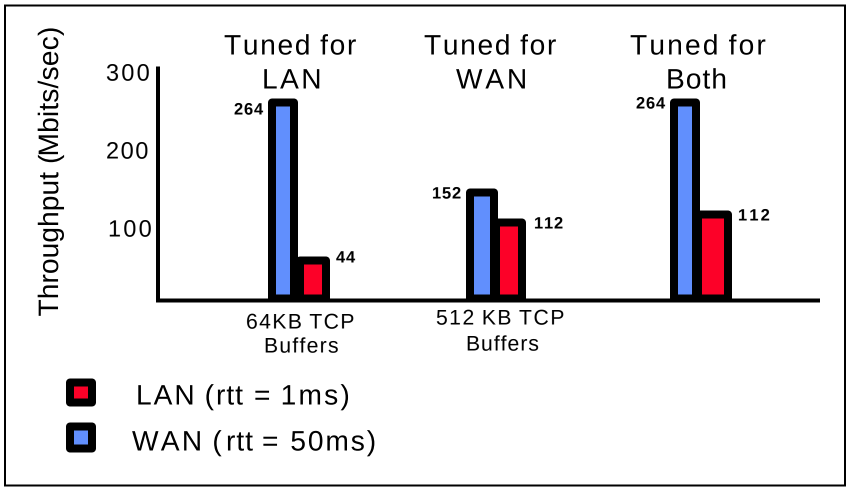

Figure 4 Importance of correct TCP tuning

To solve this problem, the DPSS client library attempts to automatically determine the bandwidth-delay product for each connection to a DPSS server and sets the TCP buffer size to the optimal value dynamically for each client. This provides optimal tuning both LAN and WAN clients simultaneously. Optionally the user can specify the buffer size to the DPSS client library via a shell environment variable.

We are currently developing a network monitoring service that will monitor the network path between specified sites, and store the bandwidth delay products for these network paths in a directory service. Then "network-aware" application can query this service for the optimal TCP buffer size to use on the fly.

4. A HIGH ENERGY PHYSICS APPLICATION

We have conducted a set of high-speed, network based, data intensive computing experiments between Lawrence Berkeley National Laboratory (LBNL) in Berkeley, Calif., and the Stanford Linear Accelerator (SLAC) in Palo Alto, Calif. The results of this experiment were that a sustained 57 megabytes/sec of data were delivered from datasets in the distributed cache to the remote application memory, ready for analysis algorithms to commence operation. This experiment represents an example of our data intensive computing model in operation.

The prototype application was the STAR analysis system that analyzes data from high energy physics experiments. (See [7].) A four-server DPSS located at LBNL was used as a prototype front end for a high-speed mass storage system. A 4-CPU Sun E-4000 located at SLAC was a prototype for a physics data analysis computing cluster, as shown in Figure5. The National Transparent Optical Network testbed (NTON - see [16]) connects LBNL and SLAC and provided a five-switch, 100-km, OC-12 ATM path. All experiments were application-to-application, using TCP transport.

Multiple instances of the STAR analysis code read data from the DPSS at LBNL and moved that data into the memory of the STAF application where it was available to the analysis algorithms. This experiment resulted in a sustained data transfer rate of 57 MBytes/sec from DPSS cache to application memory. This is the equivalent of about 4.5 TeraBytes / day. The goal of the experiment was to demonstrate that high-speed mass storage systems could use distributed application-level caches to make data available to the systems running the analysis codes. The experiment was successful, and the next steps will involve completing the mechanisms for optimizing the MSS staging patterns and completing the DPSS interface to the bit file movers that interface to the MSS tape drives.

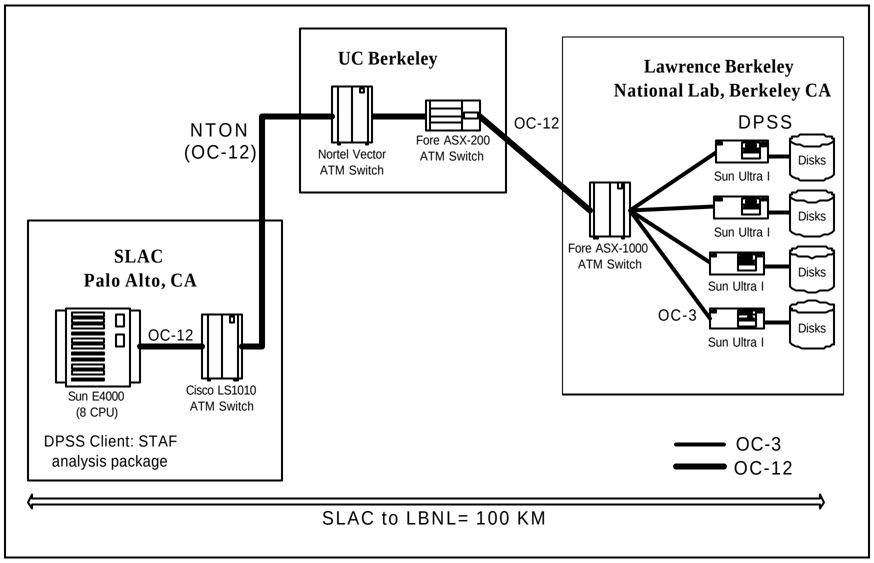

Figure 5 HEP Experiment Configuration

4.1 Current Results

We recently achieved 71 MBytes/sec (568 Mbits/sec) of I/O across the wide area using the configuration shown in Figure6. This was using a single application runing on an 8 node Compac Alpha Linux cluster, located at Sandia National Lab in Livermore, CA, about 100 km from the DPSS system at Lawrence Berkeley National Lab in Berkeley, CA. Note that the bottleneck link in the network is an OC-12 "packet over Sonet" (i.e.: not ATM) pipe from Berkeley to Oakland. The maximum data rate on OC-12 ATM is 534 Mbits/sec due to ATM header overhead, which OC-12 packet over Sonet does not have. This is how we were able to acheive a throughput that is faster that the maximum OC-12 ATM data rate.

It should be noted that both this application and the HEP application above read data from the DPSS in chunk of at least 8 MBytes per dpssRead() call. Read block sizes on the order of at least 4 MBytes in size are required achieve such high I/O rates, in order to fill the entire I/O pipeline.

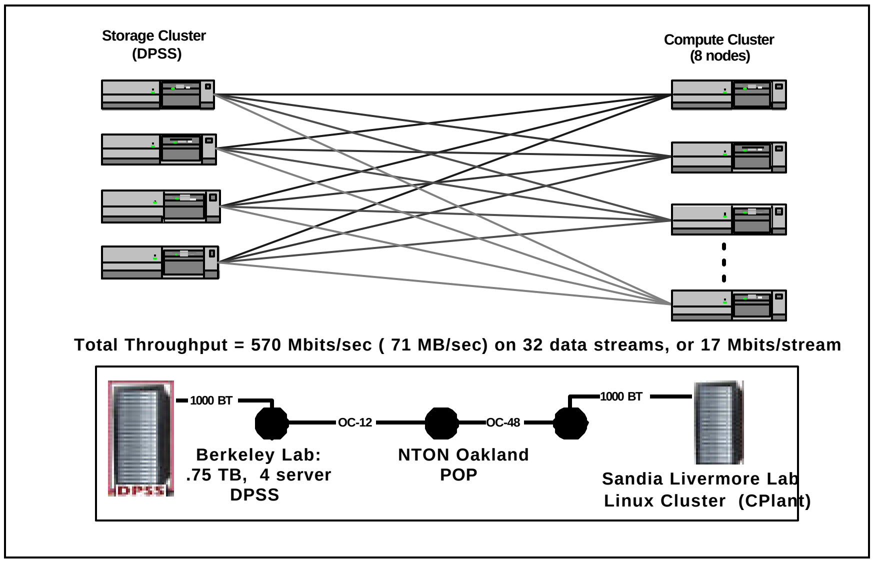
Storage Cluster (DPSS)
Compute Cluster (8 nodes)
Total Throughput = 570 Mbits/sec (71 MB/sec) on 32 data streams, or 17 Mbits/stream
DPSS
1000 BT
OC-12
OC-48
1000 BT
Berkeley Lab: .75 TB, 4 server DPSS
NTON Oakland POP
Sandia Livermore Lab Linux Cluster (CPlant)

Figure 6 Compute Cluster Application

5. CONCLUSIONS

We believe this architecture, and its integration with systems like Globus, will enable the next generation of configurable, distributed, high-performance, data-intensive systems; computational steering; and integrated instrument and computational simulation. We also believe a high performance network application-level cache system such as the DPSS will be an important component to these "computational grid" and "data grid" environments.

Acknowledgments

The work described in this paper is supported by the U. S. Dept. of Energy, Office of Energy Research, Office of Computational and Technology Research, Mathematical, Information, and Computational Sciences and ERLTT Divisions under contract DE-AC03-76SF00098 with the University of California, and by DARPA, Information Technology Office.

6. REFERENCES

[1] Erol Akarsu, Geoffrey C. Fox, Wojtek Furmanski, Tomasz Haupt, "WebFlow - High-Level Programming Environment and Visual Authoring Toolkit for High Performance Distributed Computing," Proceedings of IEEE Supercomputing '98, Nov. 1998.

[2] Chaitanya Baru, Reagan Moore, Arcot Rajasekar, Michael Wan, "The SDSC Storage Resource Broker," Proc. CASCON'98 Conference, Nov.30-Dec.3, 1998, Toronto, Canada. (http://www.npaci.edu/DICE/SRB)

[3] A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, S. Tuecke, "The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Data Sets", Internet II Network Storage Symposium, Oct. 1999, http://dsi.internet2.edu/netstore99/

[4] K. Czajkowski, I. Foster, C., Kesselman, N. Karonis, S. Martin, W. Smith, S. Tuecke. "A Resource Management Architecture for Metacomputing Systems," Proc. IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing,1998.

[5] EU DATAGRID Project, http://home.cern.ch/~les/grid/

[6] B. Fuller and I. Richer "The MAGIC Project: From Vision to Reality," IEEE Network, May, 1996, Vol. 10, no. 3. http://www.magic.net/

[7] W. Greiman, W. E. Johnston, C. McParland, D. Olson, B. Tierney, C. Tull, "High-Speed Distributed Data Handling for HENP," Computing in High Energy Physics, April, 1997. Berlin, Germany. http://www-itg.lbl.gov/STAR/

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

[9] Grid: *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. http://www.mkp.com/books_catalog/1-55860-475-8.asp

[10] Habanero: http://www.ncsa.uiuc.edu/SDG/Software/ Habanero/

[11] V. Jacobson, "Congestion Avoidance and Control," Proceedings of ACM SIGCOMM '88, August 1988.

[12] W. Johnston, G. Jin, C. Larsen, J. Lee, G. Hoo, M. Thompson, B. Tierney, J. Terdiman, "Real-Time Generation and Cataloguing of Large Data-Objects in Widely Distributed Environments," International Journal of Digital Libraries - Special Issue on "Digital Libraries in Medicine". November, 1997. (Available at http://www-itg.lbl.gov/WALDO/)

[13] William E. Johnston. "Real-Time Widely Distributed Instrumentation Systems," In *The Grid: Blueprint for a New Computing Infrastructure*. Edited by Ian Foster and Carl Kesselman. Morgan Kaufmann, Pubs. August 1998.

[14] William E. Johnston, W. Greiman, G. Hoo, J. Lee, B. Tierney, C. Tull, and D. Olson. "High-Speed Distributed Data Handling for On-Line Instrumentation Systems," Proceedings of ACM/IEEE SC97: High Performance Networking and Computing. Nov., 1997. http://www-itg.lbl.gov/ ~johnston/papers.html

[15] Legion: See http://www.cs.virginia.edu/~legion/

[16] NTON, "National Transparent Optical Network Consortium." See http://www.ntonc.org/.

[17] Partical Phyics Data Grid (PPDG) Project, http://www.cacr.caltech.edu/ppdg/

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

[19] M. Thompson, W. Johnston, J. Guojun, J. Lee, B. Tierney, and J. F. Terdiman, “Distributed health care imaging information systems,” PACS Design and Evaluation: Engineering and Clinical Issues, SPIE Medical Imaging 1997. (Available at http://www-itg.lbl.gov/Kaiser.IMG)

[20] Brian Tierney, William E. Johnston, Hanan Herzog, Gary Hoo, Guojun Jin, Jason Lee, Ling Tony Chen, Doron Rotem. “Distributed Parallel Data Storage Systems: A Scalable Approach to High Speed Image Servers,” ACM Multimedia ‘94 (San Francisco, October 1994). http://www-itg.lbl.gov/DPSS/papers/

[21] Brian Tierney, W. Johnston, H. Herzog, G. Hoo, G Jin, and J. Lee, “System Issues in Implementing High Speed Distributed Parallel Storage Systems,” Proceedings of the USENIX Symposium on High Speed Networking, Aug. 1994, LBL-35775. http://www-itg.lbl.gov/DPSS/ papers.html.

[22] Brian Tierney, W. Johnston, G. Hoo, J. Lee, “Performance Analysis in High-Speed Wide-Area ATM Networks: Top-to-Bottom End-to-End Monitoring,” IEEE Network, May, 1996, Vol. 10, no. 3. LBL Report 38246, 1996. http://www-itg.lbl.gov/DPSS/papers.html