

PDIO: High-Performance Remote File I/O for Portals-Enabled Compute Nodes

N. T. B. Stone, D. Balog, B. Gill, B. Johanson, J. Marsteller, P. Nowoczynski, D. Porter, R. Reddy, J. R. Scott, D. Simmel, J. Sommerfield, K. Vargo, C. Vizino

Abstract—Portals Direct I/O ("PDIO") is a special-purpose middleware infrastructure for writing data from compute processor memory on Portals-enabled compute nodes to remote agents anywhere on the WAN in real-time. The prototype implementation provided a means for aggregation of outgoing data through multiple load-balanced routing daemons, end-to-end parallel data streams through externally connected "I/O nodes", and a bandwidth feedback mechanism for stability and robustness. It was used by one research group, demonstrated live at several conferences, and shown to deliver bandwidths of up to 800 Mbit/sec.

Although the prototype met the initial design requirements for the target application, it had some limitations due to the special-purpose nature of that design. Based on experiences with that implementation, the beta version now under development has a number of interface, functionality and performance enhancements. We present the motivations for this infrastructure and the revisions that should make it a general purpose solution for users on PSC's Cray XT3 and other compute platforms.

Index Terms—computer input-output, data communication, data handling

I. INTRODUCTION

HPC researchers frequently need real-time remote access to their simulation data. Such needs range from simultaneous coupled simulations (formerly known as "metacomputing" but also as "grid computing"), to interactive simulation steering, to the simple need for prompt post-processing or rendering at another site. Distributed applications or others with heterogenous spatial characteristics implicitly require data transport from one computational site/stage to the next. Meeting these challenges with real-time delivery of simulation output data requires innovation and effective load-balancing of high-end compute and network resources.

Manuscript received March 6, 2006. This work was supported in part by the National Science Foundation.

N. T. B. Stone, D. Balog, B. Gill, B. Johanson, J. Marsteller, P. Nowoczynski, R. Reddy, J. R. Scott, D. Simmel, J. Sommerfield, K. Vargo, and C. Vizino are with the Pittsburgh Supercomputing Center, 300 S. Craig St., Pittsburgh, PA 15213 USA (phone: 412-268-4960; fax: 412-268-5832; email: stone@psc.edu).

D. Porter is with the Laboratory for Computational Science and Engineering, University of Minnesota, 499 Walter Library, 117 Pleasant Street SE, Minneapolis, MN 55455 USA.

PSC has developed a new remote I/O library that routes simulation data from memory within the compute processors on the Cray XT3[‡] to remote receivers anywhere on the WAN. Since this service routes traffic from the internal Portals [1] network directly to I/O on other machines, without the intervention of other application libraries like MPI, it is called "Portals Direct I/O" (PDIO).

The first prototype was originally designed to facilitate real-time remote rendering and steering for a specific scientific application. This resulted in a successful first implementation demonstrated live at multiple conference venues. Despite early successes in both performance and function, PSC saw the need to revise the design based on experiences with the prototype.

The newly revised "beta" version of this middleware includes several major enhancements. First, the new user-level interface is transparent, encapsulated by an intercept shim that augments the behavior of the normal UNIX file operations. Second, PDIO utilizes more scalable and robust communication protocols both over Portals within the machine and over TCP across the WAN. These updated protocols also allow support for broader use cases. Third, PDIO and associated machine resources are dynamically allocated by a centralized resource manager, to allow the PDIO infrastructure to be a truly shared resource for all users on the system. This revised design is expected to provide a popular service since virtually all computational scientists wish to access their data on machines outside the original compute platform and PDIO will accelerate that opportunity.

II. DESIGN GOALS

The PSC has a long tradition of striving to meet user requirements and accommodate unusual requests wherever feasible. This extends from customized job reservations and scheduling policies [2] to innovative software development like automated checkpoint-recovery systems [3,4]. The work described here grew out of one such request.

[‡] The Cray XT3 is a commercial HPC product. For product information see <http://www.cray.com/products/xt3/>.

The original prototype was designed as a special-purpose solution for the “PPM” application [5] by the Woodward collaboration. The beta version was redesigned based on experience with this implementation to convert the resulting special-purpose solution to a general-purpose solution, suitable to other use cases. Several users and other sites have already expressed interest in the beta version of PDIO.

The high-level design goal of PDIO is to deliver data resident in compute node memory directly to external WAN-connected hosts without staging the data through any disks or other file systems. The low-level requirements have changed from the first prototype to the beta version, now in development. Following are the specific design goals of each version separately.

A. Prototype Design Goals

The Woodward collaboration has a reputation for their specialized real-time rendering tools [6], but to be fully appreciated these tools require significant data streams or volumes. These collaborators expressed a need to get data from memory on compute nodes to remote sites at speeds of order 100 MByte/sec for the purpose of remote rendering and interactive simulation controls.

1) Seamless Remote File Delivery

The first assent was the appearance of remote data in file format, as opposed to in-memory DMA-type transfers. This was mandated by the file system interface of existing rendering tools. Most users in practice have made similar accommodations for their own or 3rd party tools or post-processing based on output files.

We further established that the best means for accomplishing this was a middleware library bridging the gap between the internal Portals computational interconnect to the external WAN. For example, writing a harvester that dynamically migrated files from local disk to remote hosts was deemed a poor match for the interactive feedback and controls inherent in the steered application. The proposed solution must aggregate traffic at the sender side, transmit it, and reconstitute it into files at the receiver side in order to achieve the desired performance and functionality. Remote file naming and structures must appear as they otherwise would on the local compute resource, allowing preexisting rendering tools to make immediate use of data in the new locale.

2) High Performance

Although in practice researchers always want the maximum possible data delivery throughput, rates of order 100 MByte/sec were required in order to satisfy

the high resolution and interactive nature of the visualization step of this steered simulation.

3) Arbitrary Output and Destinations

Remote visualization and control were the primary goal. These naturally involve the prepared visualization data. However capturing output from non-interactive batch sessions at the researcher’s home site was an obvious secondary target. In this way researchers could capture comparable data streams and other low-level simulation parameters for playback at other times, local to their home site and without initiating large batch-mode file transfers post-simulation.

4) Acceptable Constraints

Reasonable constraints included the knowledge that individual writes could be bound to unique files, a characteristic specific to this application.

B. Revised (Beta Release) Design Goals

1) Transparent Invocation

The prototype was implemented behind an API that required explicit initialization and invocation, including distinct header files and function names. While this was acceptable to the Woodward collaboration, experience confirms that the prevailing mindset among scientific applications developers is that they will avoid all site-specific customizations (beyond basic build procedures) whenever possible, and are hesitant to make any changes to their source code beyond those justified by their science. A distinct PDIO API would therefore constitute a significant barrier to mainstream user adoption. Any production release must be implemented in a manner that its usage is transparent to the applications developers (although run-time switches are generally accepted, *e.g.* batch script customizations).

2) Support for General File Access Patterns

The prototype implementation imposed an implicit constraint, namely that each `write()` invocation was associated with an independent remote file. While this satisfied the stated requirements of the Woodward collaboration, as noted, it is unsuitable to most users. The full release must support more typical file access patterns: multiple writes per file, parallel or concurrent writers, seek operations, *etc.*

3) Performance & Robustness Enhancements

The prototype implementation contained certain internal timeout and threshold constraints. Although these were tuned for effective utilization by the PPM application, generalization required that we not merely tune these for each application but eliminate dependence upon them wherever possible. Furthermore, those middleware parameters that persist must be dynamically configurable either by administrators or individual users.

The PDIO daemon’s ring buffer management was also revised to enable multi-destination support and time-based partial buffer flushing.

4) *Back-Channel Communication to the Client*

While the prototype implementation had a feedback mechanism for performance reasons and state management, the beta version will feature propagation of all relevant system error messages (e.g. “device full”) from the remote receiver back to the running application. This will strengthen the abstraction of file system locality.

5) *Resource Management*

Terascale computing platforms generally have a finite and limited number of externally connected nodes (“I/O nodes”). I/O nodes and their physical memory, network connectivity, PDIO daemons and their Portals communication identifiers are all examples of system resources that are consumed by the PDIO system. These resources must therefore be allocated in order to avoid user collisions and must be dynamically managed in order to maintain the infrastructure in an automated way.

The beta version of PDIO must have its own resource manager that handles these tasks, subject to both administrator and user controls.

III. IMPLEMENTATION DETAILS

The PDIO implementation has changed somewhat to meet the revised design goals. Where there are differences between the two versions these will be highlighted. Otherwise the following description applies to both versions.

A. *PDIO Components*

There are three communication layers in the PDIO system, as shown in Figure 1: the client library, the I/O daemon and the remote receiver. These serve as the data sender, aggregator/router and receiver, respectively. There is also a resource manager to allocate and track the active components in the system.

1) *Client Library*

The client library is the only component of the PDIO system that runs on the compute nodes. It stores a minimal amount of stream state (e.g. file offsets) and passes the data specified in each `write()` invocation to a PDIO daemon over the Portals communication layer. Beginning with the beta version, its interface to the application is through a thin intercept shim under the standard C I/O functions (e.g. `open`, `write`, `fcntl`, `close`).

The usage model is that each compute node process writes to what appears to be a “local” disk file, while the files that the application is writing are created in the file system of the remote client. The only requirements are that the output filename be prefixed

with a recognizable string (e.g. “`pdio://host:port/`”) to designate data destined for a remote PDIO receiver, and that added elements (e.g. an include path and a link library) must appear on the compile/link command lines.

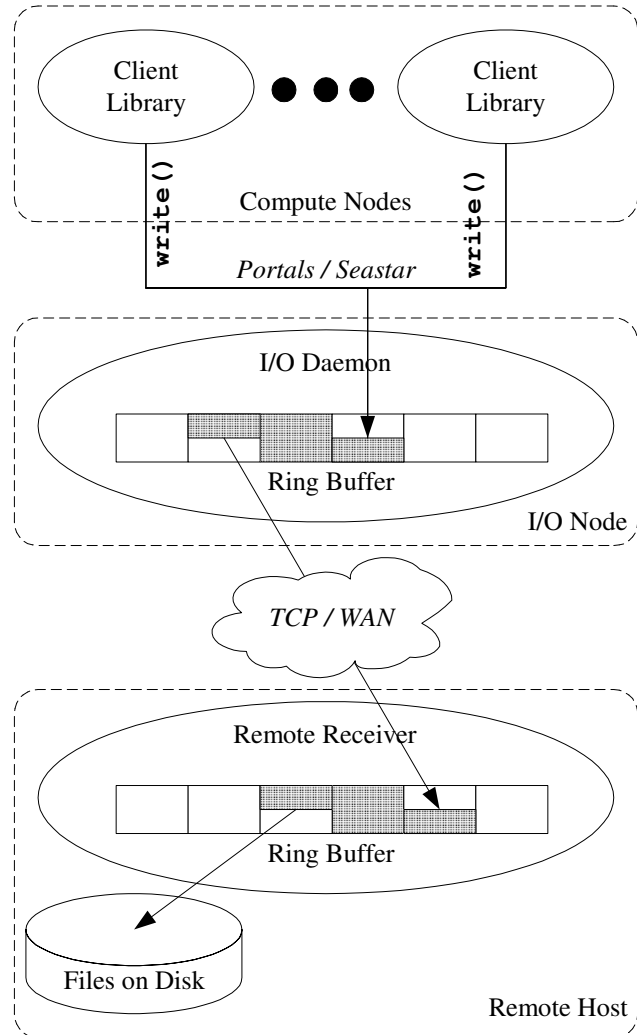


Figure 1: PDIO communication layers. Dotted lines encapsulate the various hosts; compute and I/O nodes are both within the Cray XT3. *Italics* text identifies transport protocols and media. Multiple compute processes can connect to each I/O daemon. There are multiple I/O daemons and remote receivers (not shown) that operate in parallel.

2) *I/O Daemon*

The PDIO daemon runs on an externally connected I/O node. It receives Portals data/messages from the clients on the compute nodes, aggregates them into optimally-sized buffers in memory and asynchronously routes them over the WAN via parallel TCP/IP data streams to remote receivers. Each process utilizes one or more multi-threaded ring buffers filled by Portals and emptied by TCP/IP, maximizing throughput within each daemon process. Multiple daemons can

be used across one or more I/O nodes for system-wide optimal load-balanced performance. Beginning with the beta version, these daemons are launched by a “mother hen” process, which is itself started at machine boot time. The “mother hen” spawns a configurable number of PDIO daemons and keeps them running by restarting any that are lost at any time.

3) Remote Receiver

The PDIO receiver is the final component in this chain. It is the only component that runs outside the compute resource, on any WAN-connected host. It “listens” on a well-known TCP port, waiting for an incoming connection from a PDIO daemon. Upon connection it spawns off a receiver thread/process that reconstitutes the incoming aggregated TCP buffers into individual file I/O operations. As of the beta release, all data written by the PDIO receiver are owned by the UID of the receiver process. This may change in future releases. The PDIO receiver process can persist over many streams and user jobs. There is no need to stop or restart it except as desired.

The PDIO receiver is provided in source form and requires compilation by the user. It is written in C and has not been difficult to compile on any platforms tested to date (e.g. IA32/IA64/x86-64 Linux, Tru64 UNIX).

4) Resource Manager

The PDIO resource manager (RMGR) dynamically tracks all available PDIO daemons, assigning them to running jobs as requested. Batch or interactive jobs generate requests by invoking the PDIO “agent” (analogous to the SSH agent[7]). The agent serves not only to allocate daemons but also to pass certain environment variable assignments to the client library through the running application, and can even be used to pass tuning controls to the daemons.

PDIO daemons are deallocated and terminated when the agents are terminated, at which point the “mother hen” that used to own each daemon spawns new daemons to replace those that were exhausted. All new daemons immediately register their presence, special features and Portals identifiers with the resource manager to await the next allocation.

B. PDIO Features

1) Portals/Seastar to TCP routing

The Cray XT3 communication fabric (see Figure 2) is based on the Cray Seastar chip. Each Opteron processor has a 6.4 GByte/sec memory channel and a 6.4 GByte/sec hypertransport channel connected to a Cray Seastar chip. The Seastars connect directly to one another (6 links per Seastar) by 7.6 GByte/sec communication links, so that the hardware provides an extremely high bandwidth, low-latency mesh

interconnect. Cray has implemented the Portals [1] DMA communication library over this hardware, providing a high-performance, portable interface to higher level software. By utilizing Portals the PDIO library takes direct advantage of this low-level communication protocol and exposes the compute processor memory to the I/O nodes by DMA.

The routing of traffic from the internal Portals network to the external TCP network is an obvious feature of PDIO – an implicit requirement in its design and success. The data transmission starts on the compute nodes via the client library and is propagated through the PDIO daemons, which route traffic to the PDIO receiver(s) on the external TCP network. Without this type of routing no internal data would be directly accessible outside the compute resource.

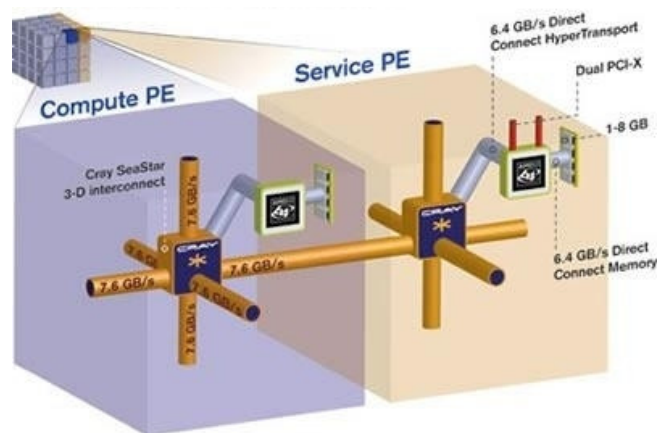


Figure 2: The Cray XT3 Scalable Architecture. This illustrates the role and configuration of the Seastar interconnect between compute (left) and service and I/O (right) nodes. See text for bandwidths.

2) Multi-Threaded Ring Buffer Management

The PDIO daemons and receivers both utilize multi-threaded ring buffer management for optimum performance. At each end of the WAN TCP connection buffers are filled by one thread (e.g. the Portals aggregator in the PDIO daemon and the TCP socket `recv()` in the PDIO receiver) and emptied by a separate thread (e.g. the TCP socket `send()` in the PDIO daemon and file system `write()` in the PDIO receiver). This parallelization allows for asynchronous, parallel I/O within each process, minimizing resource contention at each end. Beginning with the beta release, PDIO daemons can support multiple ring buffers corresponding to connections to multiple remote receivers.

3) Multiple I/O Daemons

PDIO clients have the ability to identify and select from a number of available PDIO daemons. Balancing connectivity from hundreds or thousands of clients to a smaller number (dozens) of daemons is essential to reduce resource contention. Furthermore, these daemons can run on any of the I/O nodes,

making optimal use of those I/O nodes with specialized hardware. One such specialization was to equip some I/O nodes with 10GigE network cards with 10GigE links to the TeraGrid. PSC has configured a single DDNS alias (`tg-gridftp.bigben.psc.teragrid.org`) to point to two such nodes, which were used for the SCIO5 demo as described below.

4) *A Configurable Number of TCP Output Streams*

Each PDIO daemon supports one or more external WAN connections. So by launching multiple PDIO daemons on each I/O node, users can increase the number of parallel output streams per node, optimizing the compute-Seastar-TCP resource utilization of each I/O node. Multiple socket streams per host (even as few as two) are known to achieve higher aggregate throughput on many networks than single streams.

5) *Buffer Aggregation in the I/O Daemon*

Scientific applications in general write output with a variety of buffer sizes. Individual writes of order kBytes are not unusual, but TCP buffers this size result in poor network performance. Using larger (of order MBytes) transmission buffers can significantly improve throughput on high-bandwidth, high-latency networks[8].

One of the more important roles of the PDIO daemon is to aggregate smaller writes from the PDIO client library into larger, more optimal network buffers before routing them to the external TCP network. During this aggregation data is copied from memory on a compute node, where the scientific application runs, to memory on an I/O node, which has external connectivity to the WAN.

Users can either specify their preferred size for these network buffers or accept the default values.

6) *Parallel Remote Receivers*

The PDIO receiver process binds to a well-known TCP port and waits in `accept()` for incoming connections. Upon connection it spawns off a new sub-process to handle the incoming request, returning to `accept()` new connections that may be coming. This allows multiple parallel streams to be handled by each remote receiver host.

7) *Standard Job Submission Procedures*

The job submission procedures required by a site's scheduling system do not require modification. The only run-time modification required by users is to launch a PDIO agent within the batch script. Users can pass configurable parameters to the daemons and client library via command line options for the agent or by a personal configuration file (`$HOME/.pdio.conf`).

8) *End-to-End Flow Control*

It's important for all three levels—client, daemon and receiver—have the ability to throttle back the data

streams they're receiving, especially considering the potentially large bandwidth difference between the data coming from the compute nodes over the internal Cray Seastar network and the data going out over the wide-area TCP network to the remote client. The PDIO daemon asynchronously manages multiple data streams from hundreds of clients in parallel while still providing a blocking `write()` behavior at the client end, if the transmission chain backs up. This allows PDIO to limit the rate at which messages are sent all the way back to the scientific application, slowing down the application if necessary to avoid data loss. While this may appear to be a weakness it is a strong design feature. It ensures reliability and stability whether running over slower (*e.g.* low-end commodity) or faster (*e.g.* NSF TeraGrid) networks, and similarly accommodates remote file system performance issues.

If the WAN transmission pipeline is not backed up then the `write()` invocation at the client side (on the compute node) will return as soon as the DMA operation from compute to I/O node has completed. This operation has characteristic speeds approaching memory resident file systems, as opposed to disk-based file systems. This should allow the application to return to computation more quickly, spending less time in file I/O functions, provided that the WAN connectivity and file systems at the remote host(s) are fast enough. As noted below the prototype has already demonstrated performance approaching Gigabit-Ethernet speeds, provided adequate remote resources.

C. *PDIO Security*

PDIO data does not touch any file systems within the compute resource, but only resides either in flight (*e.g.* between client and daemon, daemon and receiver), in memory (*e.g.* in the PDIO daemon ring-buffer) or on remote media, after it is stored by the PDIO receiver. Data security must be addressed at both the PDIO daemon and receiver levels.

Portals communication supports the notion of per-message Access Control Lists (ACLs). So there is no appreciable opportunity for one user to intercept another's PDIO data in flight within the compute resource. And accessing another user's data in the PDIO daemon's memory is prohibited by normal UNIX process/memory constraints.

As of the beta version, communications between the PDIO daemon and remote receiver are still performed unencrypted over TCP socket streams, to avoid performance penalties. And, while no user or password data is exchanged in this manner, this does present opportunity for malicious users to intercept data in flight during WAN communications or to attack the open port on the receiver. Once at the

remote site all files are currently owned by the UID under which the remote receiver was launched.

Future projects are already under consideration to address both shortcomings via alternative WAN communication protocols.

IV. USER EXPERIENCE

Woodward *et al.* were the first scientists to utilize this capability to remotely render compressible turbulence data and steer the simulation from a remote site while the simulation was proceeding. Their application is called the Piecewise Parabolic Method (“PPM” [5]). By modifying 35 lines of source code they added PDIO functionality to their application. They ran this remote visualization and control demonstration live at multiple conference venues [9-11] and at their home site in Minnesota. Each time they simulated the turbulent fluid dynamics in shear driven mixing layers and sent the output via PDIO to the remote site for real-time rendering.

During early test trials at their home site (from `bigben.psc.edu` to `*.lcse.umn.edu`, which traverses the Abilene/Internet2 [12] network) PDIO delivered roughly 320 Mbit/sec, pressing the practical limits of their laboratory’s internet connection. This established the performance and feasibility of this approach. By contrast, at the later SCI05 demonstration PDIO delivered rates nearing 800 Mbit/sec over a period of minutes and 200 Mbit/sec over the course of the full 90-minute live demonstration. (The rate was intentionally reduced to avoid filling the disk at the remote host over the course of the full demo.) With no PDIO-related failures during the live demonstrations PDIO has shown excellent reliability and stability for both short (minutes) and long (hours) periods of use.

V. CONCLUSIONS

PSC has created a transparent middleware layer that delivers high-performance, load-balanced, streaming output to any remote host on the Wide Area Network. We have virtually extended the file system from compute nodes to the entire internet. This provides Cray XT3 users with an unprecedented means for transferring data from memory on compute nodes to their home site, conference venue or any other site in real-time. We anticipate that researchers will use this for real-time heterogenous applications like remote rendering, coupled simulations, data-driven simulations, monitoring or even simply to have faster access to their data for post-processing.

The need for this type of solution in Massively Parallel Processing (MPP) computing environments is imminent. MPPs, contrasted with large clusters,

typically run micro-kernel operating systems on the compute nodes to enable application scalability. But these nodes typically lack external connectivity and may not even support an IP stack. The press toward petascale computing demands that our software innovation provide scalable I/O solutions that account for these hardware and OS constraints, like separation of I/O initiators from external connectivity on dedicated “I/O nodes”. PDIO makes effective use of this hardware configuration. This work could be applied directly to such platforms by either building Portals for the internal OS/network in question or replacing the internal Portals aggregation with other native transports – resulting in similar advantages for users on arbitrary HPC platforms.

REFERENCES

- [1] R. Brightwell, W. Lawry, A. B. Maccabe, R. Riesen, “Portals 3.0: Protocol Building Blocks for Low Overhead Communication,” in *Proceedings of the 2002 Workshop on Communication Architecture for Clusters*, April, 2002.
- [2] C. Vizino, J. Kochmar, and J. R. Scott, “Custom Features of a Large Cluster Batch Scheduler”, in *Proceedings of the 2005 Int. Conf. on Parallel and Distributed Processing Techniques and Applications*, 2005.
- [3] N. T. B. Stone, J. Kochmar, P. Nowoczynski, J. R. Scott, D. Simmel, J. Sommerfield, and C. Vizino, “Terascale I/O Solutions,” *Lecture Notes in Computer Science*, vol. 2660, pp. 13-22, 2003.
- [4] N. T. B. Stone, J. Kochmar, R. Reddy, J. R. Scott, J. Sommerfield, and C. Vizino, “A Checkpoint and Recovery System for the Pittsburgh Supercomputing Center Terascale Computing System,” *PSC technical report CMU-PSC-TR-2001-0002*, unpublished. Available: http://www.psc.edu/publications/tech_reports/
- [5] P. R. Woodward, “A Complete Description of the PPM Compressible Gas Dynamics Scheme,” in *Implicit Large Eddy Simulation: Computing Turbulent Flow Dynamics*, edited by F. Grinstein, Cambridge University Press, 2006 (in press).
- [6] P. R. Woodward, D. H. Porter, A. Iyer, “Initial experiences with grid-based volume visualization of fluid flow simulations on PC clusters,” in *Proc. Visualization and Data Analysis 2005 (VDA2005)*, San Jose, CA, Jan., 2005. Available: <http://www.lcse.umn.edu/VDA2005>
- [7] OpenSSH project, “ssh-agent man page”, unpublished. Available: <http://www.openbsd.org/cgi-bin/man.cgi?query=ssh-agent&sektion=1>
- [8] M. Mathis, R. Reddy, “Enabling High Performance Data Transfers,” 2006. Available: <http://www.psc.edu/networking/projects/tcptune/>
- [9] Global Lambda Integrated Facility, iGrid2005 Conference, 2005. Available: <http://www.igrid2005.org/>
- [10] IEEE, Visualization 2005 Conference, 2005. Available: <http://vis.computer.org/vis2005/>
- [11] IEEE / ACM, SCI05 Conference, 2005. Available: <http://sc05.supercomputing.org/>
- [12] The Abilene Backbone Network, 2006. Available: <http://abilene.internet2.edu/>