Effective Use of Dedicated Wide-Area Networks for High-Performance Distributed Computing

Nicholas T. Karonis,[*♦] Michael E. Papka,[•♦] Justin Binns,[♦] John Bresnahan,[♦] Joseph M. Link[♦]

[*]*Department of Computer Science*
Northern Illinois University
karonis@niu.edu

[•]*Department of Computer Science*
University of Chicago
papka@cs.uchicago.edu

[♦]*Mathematics and Computer Science Division*
Argonne National Laboratory
{binns,bresnaha,link}@mcs.anl.gov

Abstract

Recent advances in Grid technology have made it possible to build so-called computational Grids, or simply Grids, which couple unique or rare resources that are geographically separated and span multiple administrative domains. Such Grids are invariably composed of heterogeneous networks in which, at the least, a high-performance switch accommodates intracluster messages and a separate, sometimes dedicated, high-bandwidth network serving intersite messages across the wide area. While such wide-area networks provide unprecedented bandwidth capacity and reliability, the effective utilization of these networks remains an open challenge. Most applications by default use the TCP/IP protocol for its ease of use and reliability, but the high bandwidth and high latency sometimes found on these networks induce enormous bandwidth delay products that result in extremely large TCP congestion window sizes. This situation makes TCP a poor choice for data-intensive applications striving to achieve maximum bandwidth utilization on high-performance networks. To address this bandwidth utilization challenge for Grids connected over dedicated networks, we present a solution based on the UDP protocol with added reliability and the Message Passing Interface (MPI) standard. MPI provides an interface that allows application programmers to ignore network heterogeneity. To study the efficacy of our approach, we implemented our implementation of the Reliable-Blast UDP protocol in MPICH-G2, our Grid-enabled MPI. We demonstrated this implementation in an MPI data-intensive Grid visualization application on the NSF TeraGrid and its dedicated high-bandwidth fiber optic network. We observed an improvement in aggregate bandwidth utilization from 58 Mbps with MPICH-G2 using TCP alone to 9 Gbps with our technique.

1. Introduction

Recent advances in Grid technology, most notably through the efforts of the Globus Alliance [1], have made it possible to build so-called computational Grids, or simply Grids [2]. The Globus Toolkit® has provided the Grid infrastructure necessary to construct a number of Grids. The National Partnership for Advanced Computational Infrastructure (NPACI) and the National Computational Science Alliance's (NCSA) National Technology Grid, the European DataGrid project [3], the National Aeronautics and Space Administration Information Power Grid [4], and the National Science Foundation's TeraGrid [5] are just a few examples.

While the Globus Toolkit makes it *possible* for applications to run on Grids, their ability to *run efficiently* in those environments remains an open challenge. Techniques for dealing with the dynamic behavior of Grids have been applied successfully in some situations [6-12], but their use remains difficult, particularly for complex applications.

One especially challenging aspect of Grids is their networks, which are characterized predominantly by their heterogeneity. Grid networks typically include a high-performance switch for intramachine or intracluster messages and a separate, sometimes dedicated, high-bandwidth network serving intersite messages across the wide area. The NSF TeraGrid is one example. Each TeraGrid cluster is equipped with a Myrinet switch, and the TeraGrid sites are connected over a dedicated fiber optic network with up to 60-millisecond latency and an aggregate bandwidth capacity of 40 Gbps (although data may be transmitted between any two TeraGrid sites at a maximum data rate of only 30 Gbps). Having a *dedicated* wide-area network introduces numerous possibilities for optimal performance as well as numerous challenges. Similarly, the network *heterogeneity* provides an opportunity for peak point-to-point message-passing performance; but taking advantage of that opportunity requires special message-passing routines appropriate for each network component (e.g., one for communicating over a Myrinet switch and another for the wide area), as well as mechanisms to dynamically select the appropriate routines based on the point-to-point source and destination. The development of such specialized message-passing libraries is a deterrent for all but the most determined application programmers.

Despite these obstacles, the challenge of *efficiently utilizing* these heterogeneous Grid networks with dedicated high-bandwidth intersite connectivity remains. To meet this challenge, we propose a solution based on the *User Datagram Protocol* (UDP) with added reliability and the Message Passing Interface (MPI) standard [13]. Our choice of MPI enables application programmers to ignore network heterogeneity by having the MPI library automatically select a high-performance switch for intracluster messages or *Transmission Control Protocol* (TCP) sockets for intersite messages. Optionally, through the use of MPI idioms, application programmers can easily configure point-to-point intersite channels, thus enabling the reliable UDP transport. To study the efficacy of our approach, we implemented a version of the Reliable-Blast UDP (RBUDP) protocol [14] in MPICH-G2 [15, 16], our Grid-enabled MPI.

In the following sections we present an overview of the underlying Grid infrastructure, a detailed description of the application, and our experience on the NSF TeraGrid at the Supercomputing 2003 conference including our performance results. We conclude with a brief summary.

2. Grid Infrastructure

A computational Grid environment differs in many respects from standard parallel computing environments:

- A parallel computer is usually fairly homogenous. In contrast, a Grid may incorporate nodes with different processor types, memory sizes, and so forth.
- A parallel computer typically has a dedicated, optimized, high-bisection bandwidth communications network with a generally fixed topology. In contrast, a Grid may have a highly heterogeneous and unbalanced communication network, comprising a mix of different intramachine networks and a variety of Internet connections whose bandwidth and latency characteristics may vary greatly in time and space.
- A parallel computer typically has a stable configuration. In contrast, resource availability in a Grid can vary over time and space.
- A parallel computer typically runs a single operating system and provides basic utilities such as a file system and resource manager. In contrast, a Grid environment, being a collection of multiple vendor machines, integrates potentially radically different operating systems and utilities.

In Figure 1 we depict the layers necessary to run a Grid application. At the highest level we have a *Grid application*. The abstraction presented is either MPI or lower-level Grid services provided, for example, by the Globus Toolkit.

Below this we have a *Grid-enabled communication library:* in our case, MPICH-G2. The abstraction presented is the MPI programming model; the programmer queries structure and state and controls the underlying implementations by getting and setting, respectively, attribute values associated with MPI communicators. All details of how the MPI programming model is implemented across different resources are encapsulated, including startup, monitoring, and control. The Grid-enabled MPI implementation uses functions provided by a co-allocation library, in our case, the Dynamically Updated Resource Online Co-Allocator (DUROC) [17]. This library abstracts details relating to how processes are created, monitored, and managed in a coordinated fashion across different computers. The programmer can specify a set of machines on which processes are to be started; the DUROC library manages the acquisition of those resources and the creation of the processes.

Finally, a set of *Grid services* abstract the myriad complexities of heterogeneous environments. These services provide uniform protocols and APIs for discovering, authenticating with, reserving, starting computation on, and in general managing computational, network, storage, and other resources.

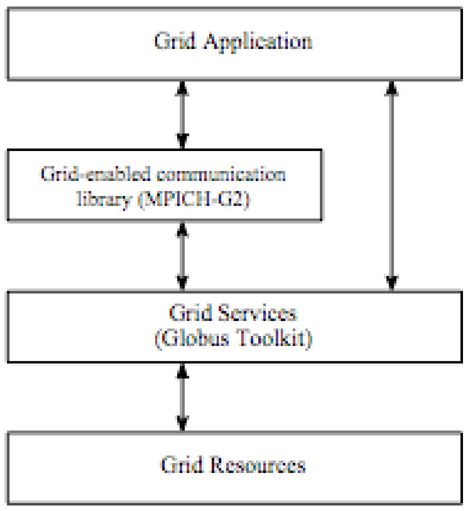
Grid Application
Grid-enabled communication library (MPICH-G2)
Grid Services (Globus Toolkit)
Grid Resources

Figure 1 A layered architecture for running applications on a Grid.

2.1 High-Bandwidth Wide-Area Networks

When communicating over a wide-area network most applications choose TCP for its ease of use and reliability. Unfortunately, the high bandwidth and high latency often found on Grid networks induce enormous bandwidth delay products that result in TCP congestion window sizes that are problematically large. The TCP protocol starts by exponentially increasing the sender-side congestion window until it encounters its first *congestion event*, which it uses as a trigger to first cut the size of the current congestion window in half. It then enters *congestion avoidance mode,* where it increases the window only linearly (by only 1,500 bytes for a standard TCP connection) per round-trip message. Because this process takes too long for the window size to increase to the optimal size, TCP is a poor choice for data-intensive applications striving to achieve maximum bandwidth utilization. Floyd [18] addresses this fundamental limitation of TCP by proposing a modification to TCP's congestion control mechanism for use with TCP connections with large congestion windows.

One technique that delivers modest improvements in bandwidth utilization over such networks is to open multiple pairs of TCP sockets between sender and receiver, partition the message sending the pieces in parallel down the TCP channels, and assemble the message on the receiving side [19, 20]. While this technique does not directly address the problem embedded in the TCP protocol, it achieves nearly the same effect as specifying a large congestion window by effectively increasing the number of unacknowledged bytes "in flight" at any given time.

A second technique is to use UDP. This protocol is easy to use, but it is not reliable, and its absence of a congestion avoidance mechanism makes it an inappropriate choice for nondedicated networks, particularly across a wide area. Adding reliability to a UDP-based protocol is straightforward [21, 22]; and on a dedicated network, UDP's lack of congestion avoidance is much less an issue. Furthermore, most high-bandwidth wide-area networks today are optical networks and very reliable (i.e., requiring few UDP retransmissions). Therefore, the use of UDP over high-bandwidth dedicated networks becomes an interesting and worthwhile option to explore.

2.2 The Globus Toolkit

The Globus Toolkit is a collection of software components designed to support the development of applications for high-performance distributed computing environments [23]. Core components typically define a protocol for interacting with a remote resource, plus an application program interface (API) used to invoke that protocol. Higher-level libraries, services, tools, and applications use core services to implement more complex global functionality. The various Globus Toolkit components are reviewed in [24] and described in detail in online documentations and in technical papers.

2.3 MPICH-G2: A Grid-Enabled MPI

MPICH-G2 is a complete implementation of the MPI-1 standard that uses Globus Toolkit services to support efficient and transparent execution in heterogeneous Grid environments, while also allowing for application management of heterogeneity. In this section, we briefly describe the techniques used to effect communication in heterogeneous systems where our primary focus is how MPICH-G2 uses Globus IO and the RBUPD method implemented in it.

Once the application has started, MPICH-G2 selects the most efficient communication method possible between any two processes, using either vendor-supplied MPI (*v*MPI) if available, or a Grid-based communication technique such as Globus IO. With Globus IO, one may also use one of three options: TCP sockets, parallel TCP sockets, via the Globus Toolkit GridFTP (for nondedicated networks), or UDP with added reliability (Reliable-Blast UDP, or RBUDP).

The last option is useful for applications that transfer large blocks of data in single messages from one process to another over a high-bandwidth dedicated network (e.g., visualization data over an optical network). MPICH-G2 delivers this option to its applications through the use of existing MPI idioms. Figure 2 depicts an excerpt from an MPICH-G2 application in which a target bandwidth of 1 Gbps is established over a reliable UDP channel between **MPI_COMM_WORLD** ranks 0 and 1. In this example both processes start by setting values that (a) designate each other as partners, (b) request a UDP payload size of 1,452 bytes, and (c) request a transmission rate of 1 Gbps. After setting these values, the processes place their request of MPICH-G2 by assigning their specified values to the communicator attribute **MPICHX_UDPSOCKET_PARAMETERS**. This technique of using communicator caches to influence MPICH-G2's behavior has been used successfully in a similar situation in which putting data into an MPICH-G2 communicator triggered a quality-of-service–enabled line [25] and again triggered the use of parallel TCP streams. Once the processes "put" their values into their communicators, MPICH-G2 automatically partitions all messages between them and sends them at the specified data rate using the reliable UDP method.

```
#include <mpi.h>

int main(int argc, char **argv)
{
     int numprocs, my_id;
     struct udp_params up; /* MPICH-G2 structure in mpi.h */

     MPI_Init(&argc, &argv);
     MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
     MPI_Comm_rank(MPI_COMM_WORLD, &my_id);

     If (my_id == 0 || my_id == 1) {
          /* must set these three fields */
          up.partner_rank         = (my_id ? 0 : 1);
          up.my_UDP_payload_size  = 1452;
          up.sendrate_Mbps        = 1000.0;

          MPI_Attr_put(MPI_COMM_WORLD, MPICHX_UDPSOCKET_PARAMETERS, &up);
     } /* endif */

     /* from this point all message exchanged between
      * MPI_COMM_WORLD ranks 0 and 1 will be automatically
      * partitioned and transported over reliable UDP
      */
    .
    .
    MPI_Finalize();

} /* end main() */
```
Figure 2 An excerpt from an MPICH-G2 application that uses **MPI_Attr_put** to enable reliable UDP transport between **MPI_COMM_WORLD** ranks 0 and 1. Here the application specifies a UDP payload size of 1,452 bytes and a transmission rate of 1 Gbps.

3. Application Overview

The application developed consists of two components: a client that allows the user to interact with the data to determine optimal parameters for the visualization and a high-resolution distributed rendering system that produces a full-resolution visualization matching the client parameters. The two components together form a visualization tool for the exploration of very large datasets. The application was tested using data produced by the FLASH code developed at the University of Chicago's ASCI/Alliance Center for Astrophysical Thermonuclear Flashes[26]. FLASH is a modular, adaptive, parallel simulation code for handling general compressible flow problems in astrophysical environments. The results of the simulations are stored in well-defined HDF5 files that represent a multiresolution block-structured grid. Block sizes vary in spatial dimension, depending on level of refinement, but always contain the same number of cells.

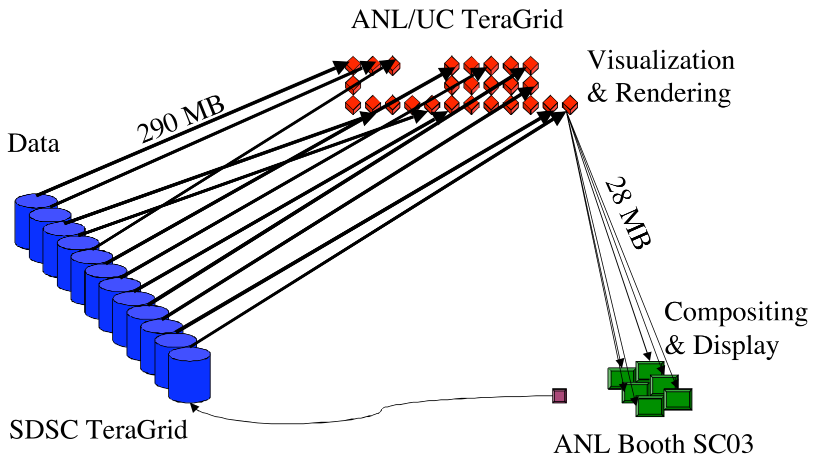

Figure 3 An overview of our application running with 12 loaders on the SDSC-TG, 24 renderers on the ANL-TG, and 6 compositors on a cluster located on the Supercomputing 2003 conference showroom floor in Phoenix, Arizona.

The constant cell count within a dataset can be used to construct a low-resolution subset that shares common characteristics with the full-resolution data. Using only the corner data of the blocks, one can construct a dataset that produces substantially similar results with much less detail, providing a source for the interactive client. The full-resolution dataset is then used for the final visualization, once the user has established the appropriate parameters.

The client is designed to provide an interactive, user-friendly visualization experience using subsampled datasets of manageable size. The client is built using Python and the Visualization Toolkit (VTK) [27]. The client provides a myriad array of visualization components, including isosurfacing, cutting planes, and vector visualization (via glyphs). The user can configure the client to produce the desired view and visualization results. A command is then issued to the high-resolution distributed rendering pipeline. XML-RPC is used to communicate the parameters and to start the rendering process [28]. Once the process has begun, the client is independent, and the user may begin establishing the next desired view.

The high-resolution remote-rendering component exists as a distributed pipeline through which a full dataset rendering is produced. The final output can range from standard desktop-sized images to very high resolution images for a tiled display. The output can be displayed to a display device, saved to a file, or both. When the visualization is displayed directly, it is displayed as it is rendered, with each piece of rendered data producing a *deep image* (that is, an image containing data from the both framebuffer and zbuffer) that is used for compositing. As the chunks are progressively composited in the final display, the full dataset is visualized.

A key feature of the remote rendering component is its distributed nature. Several pieces, including a data loading piece, a rendering piece, and a compositing piece, form the remote visualization pipeline, along with a master to coordinate operations. The visualization pipeline receives its instructions via a network interface. Through this interface, a client specifies all the information necessary to produce the desired visualization. The system is designed such that immediately following the production of one image, a second can begin without having to restart the backend processes. This is possible even if the dataset being used is different.

4. Experimental Results

To study the efficacy of our approach, we demonstrated the FLASH application at the Supercomputing 2003 conference in Phoenix, Arizona, in November 2003. We ran the application as a single **MPI_COMM_WORLD** across two of the NSF TeraGrid sites and a small compute cluster on the conference showroom floor. The distributed nature of the remote rendering component allowed us to optimize the resources available at the different TeraGrid sites and produced a faster pipeline that emphasized each site's strengths. Specifically, as depicted in Figure 3 we placed 12 loaders on the TeraGrid data-store cluster located at the San Diego Supercomputer Center (SDSC-TG); 24 renderers on the TeraGrid visualization cluster located at Argonne National Laboratory (ANL-TG); and 6 compositors running on a small Linux cluster on the SuperComputing 2003 exhibit floor, where the final image was displayed. All three clusters were equipped with version 2.4.3 of the Globus Toolkit and version 1.2.5.1a of MPICH-G2 specially modified to include an implementation of the RBUPD method.

4.1 The NSF TeraGrid

The SDSC-TG was equipped with 128 Madison dual-processor nodes with a CPU speed of 1.3 GHz and 4 GB RAM per processor and was running SUSE SLES8 operating system with the 2.4.19-SMP kernel. The ANL-TG was equipped with 96 Intel Itanium 2 dual-processor nodes with a CPU speed of 1 GHz and 2 GB RAM per processor and was running Itanium: SLES8 operating system with the 2.4.19-SMP kernel. Each node on the SDSC-TG and ANL-TG were equipped with Gigabit Ethernet, and the operating systems on the TeraGrid clusters had been upgraded to use 9000-byte jumbo Ethernet frames.

The TeraGrid's network connectivity is shown in Figure 4. The left side of Figure 4 depicts an overview of how the TeraGrid sites are interconnected. Each site connects to one of two local hubs over 3x10 Gbps fiber channels. The two hubs are connected over 4x10 Gbps fiber channels. This network topology can deliver a bandwidth of up to 30 Gbps in/out of any single TeraGrid site and an aggregate bandwidth of up to 40 Gbps. The right side of Figure 4 depicts a more detailed view of the network path from an individual TeraGrid cluster to its nearest hub. Here we see the cluster's many Gigabit Ethernet lines entering a switch, which multiplexes those signals across 3x10 Gbps channels to a TeraGrid border router, which in turn connects to the local hub via 3x10 Gbps channels.

4.2 Bandwidth Results

We measured the bandwidth of each of the 12 loaders located at SDSC-TG as they cumulatively sent a total of 96 messages, each 290 MB, to the 24 renderers. During our demonstrations we had exclusive access to the network connecting TeraGrid clusters but not between ANL-TG and the cluster located at the conference. We first ran our application without enabling the reliable UDP transport and ran with MPICH-G2 using only TCP sockets between the

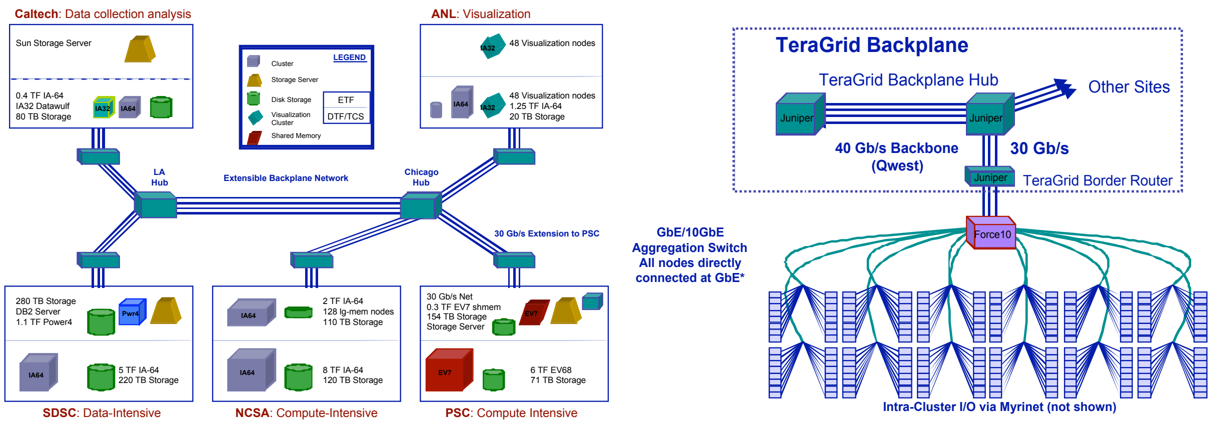

Figure 4 An overview of the NSF TeraGrid's network connectivity. The left side illustrates how the TeraGrid sites are interconnected and the right side provides a more detailed view of how a cluster at a single TeraGrid site connects to its nearest hub.

three clusters. We observed an aggregate bandwidth of only 58 Mbps across the 288 point-to-point loader-to-renderer channels.

We then ran our demonstration again, this time enabling the reliable UDP transport. We modified our MPI application to enable the use of our reliable UDP between each of the 12 SDSC-TG loaders and all 24 of the ANL-TG renderers (i.e., the only dedicated portion of the network). We configured the 288 channels specifying a UDP payload size of 8,952 bytes (selected such that the 8,952 UDP payload bytes, the 8-byte UDP header, and the 20-byte IPv4 header would all fit in a single jumbo Ethernet frame). Initially we requested transmission rate of 1 Gbps, but this resulted in too many retransmissions in our reliable UDP protocol, which in turn resulted in poor effective point-to-point bandwidths. We slowly reduced the requested transmission rate until we observed only a few retransmissions and finally settled at 800 Mbps. We again measured the bandwidth of each of the 12 loaders as they cumulatively sent a total of 96 messages to the 24 renderers. As shown in Figure 5 MPICH-G2's use of UDP with added reliability resulted in an aggregate bandwidth of roughly 9 Gbps, a significant improvement over the 58 Mbps achieved when it used only TCP.

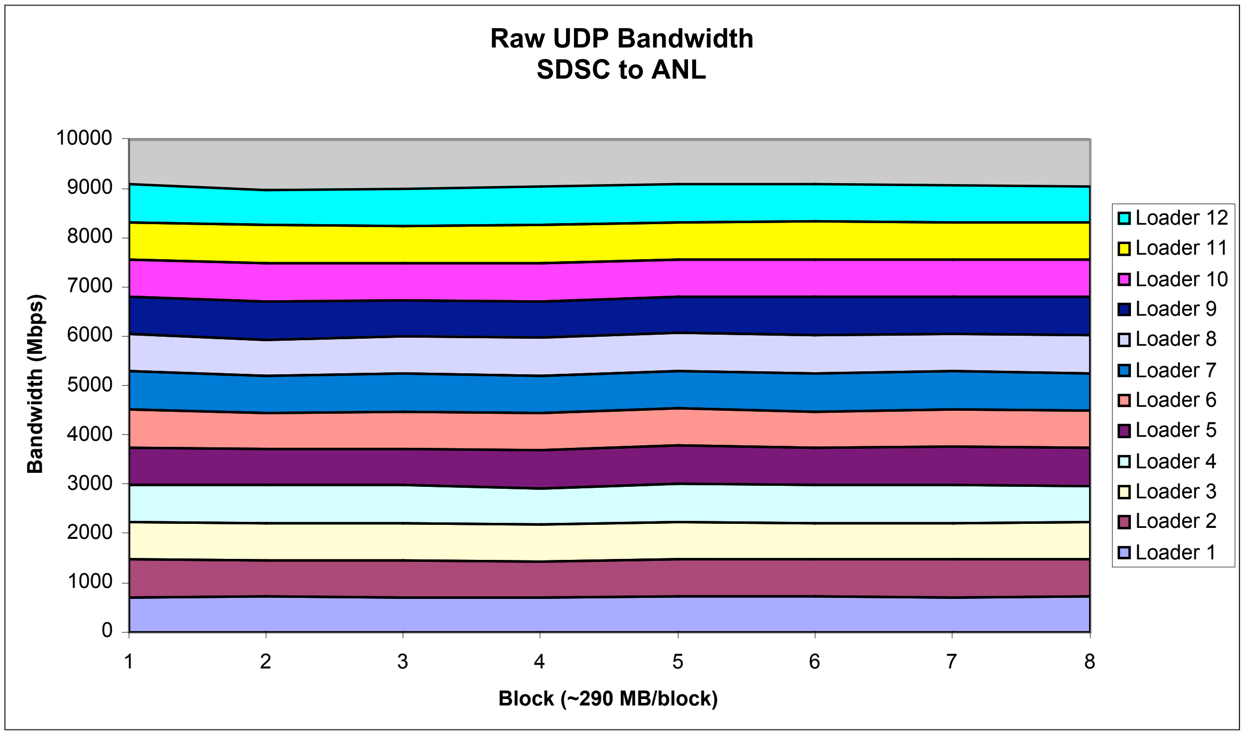

Figure 5 The 9 Gbps aggregate bandwidth when enabling MPICH-G2's reliable UDP transport on the dedicated network channels connecting the 12 SDSC-TG loaders and the 24 ANL-TG renderers.

5. Conclusion

In recent years a small number of software research and development projects have produced middleware that facilitates the construction of computational Grids. These Grids couple rare or unique resources that are geographically separated and often span multiple administrative domains. While these Grids represent enabling technology for many important applications, particularly high-performance and distributed computing applications, the efficient utilization of Grid resources remains an open challenge. Of particular interest is the efficient utilization of high-bandwidth Grid networks. The inherent heterogeneity of Grid networks, coupled with the inherent limitations of the standard network protocols like TCP, makes it difficult for applications running in Grid environments to easily realize near-full bandwidth capacity.

To address this bandwidth utilization challenge on dedicated Grid networks, we present a solution based on the UDP protocol with added reliability and the MPI Standard. MPI provides a well-defined interface for application programmers, allowing them to easily manage network heterogeneity by configuring select point-to-point channels using existing MPI idioms.

To study the efficacy of our approach we implemented a version of the RBUPD method in MPICH-G2, our Globus-enabled implementation of the MPI standard. We demonstrated our implementation at the Supercomputing 2003 conference in Phoenix, Arizona, using the NSF TeraGrid and computing resources located on the conference showroom floor. We observed an improvement in aggregate bandwidth utilization from 58 Mbps with MPICH-G2 using TCP alone to 9 Gbps with our technique. We conclude that our proposed method is an effective and easy way to achieve high-bandwidth utilization over high-bandwidth dedicated networks.

Acknowledgments

We thank the NSF TeraGrid for providing exclusive access to their resources during the Supercomputing 2003 conference and the system administrative support in the months prior to the conference. Funding for this work has been provided by the National Science Foundation Middleware Initiative (NMI) under contracts ANI-022983 and ANI-0330664 and by the NASA Information Power Grid program. This work was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract W-31-109-ENG-38.

References

[1] The Globus Alliance Web page, www.globus.org.

[2] I. Foster and C. Kesselman, *The Grid: Blueprint for a New Computing Infrastructure (2nd Edition)*, Morgan Kaufmann, 2004.

[3] The European DataGrid Project Web page, eu-datagrid.web.cern.ch/eu-datagrid.

[4] Information Power Grid Web page, www.nas.nasa.gov/About/IPG/ipg.html.

[5] TeraGrid Web page, www.teragrid.org.

[6] F. Berman, "High Performance Schedulers," in *The Grid: Blueprint for a New Computing Infrastructure*, I. Foster and C. Kesselman, eds.: Morgan Kaufmann, 1999, pp. 279-309.

[7] S. Brunett, D. Davis, T. Gottschalk, P. Messina, and C. Kesselman, "Implementing Distriburted Synthetic Forces Simulations in Metacomputing Envrionments," presented at Heterogenous Computing Workshop, 1998.

[8] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke, "A Resource Management Architecture for Metacomputing Systems," presented at 4th Workshop on Job Scheduling Strategies for Parallel Processing, 1998.

[9] P. Messina, "Distributed Supercomputing Applications," in *The Grid: Blueprint for a New Computing Infrastructure*, I. Foster and C. Kesselman, Eds.: Morgan Kaufmann, 1999, pp. 55-73.

[10] J. Nieplocha and R. Harrison, "Shared Memory NUMA Programming on the I-WAY," presented at 5th IEEE International Symposium on High Performance Distributed Computing, 1996.

[11] P. M. Papadopoulos and G. A. Geist, "Wide-Area ATM Networking for Large-Scale MPPS," presented at *SIAM conference on Parallel Processing and Scientific Computing*, 1997.

[12] T. Sheean, W. Shelton, T. Pratt, P. Papadopoulos, P. LoCascio, and T. Dunigan, "Locally Self Consistent Multiple Scattering Method in a Geographically Distibuted Linked MPP Environment," *Parallel Computing*, vol. 24, 1998.

[13] "Message Passing Interface Forum. MPI-2: A Message Passing Interface Standard," *International Journal of High Performance Computing Applications*, vol. 12, pp. 1-299, 1998.

[14] E. He, J. Leigh, O. Yu, and T. A. DeFanti, "Reliable Blast UDP: Predictable High Performance Bulk Data Transfer," presented at IEEE Cluster Computing, 2002.

[15] MPICH-G2 Web page, www.globus.org/mpi.

[16] N. Karonis, B. Toonen, and I. Foster, "MPICH-G2: A Grid-Enabled Implementation of the Message Passing Interface," *Journal of Parallel and Distributed Computing*, vol. 63, pp. 551-563, 2003.

[17] K. Czajkowski, I. Foster, and C. Kesselman, "Co-allocation Services for Computational Grids," presented at 8th IEEE International Symposium on High Performance Distributed Computing, 1999.

[18] S. Floyd, "HighSpeed TCP for Large Congestion Windows," IETF Draft, 2002.

[19] N. Karonis, M. E. Papka, J. Binns, J. Bresnahan, J. Insley, D. Jones, and J. Link, "High-Resolution Remote Rendering of Large Datasets in a Collaborative Environment," *Future Generation of Computer Systems*, vol. 19, pp. 909-917, 2003.

[20] W. Allcock, J. Bester, J. Bresnahan, A. L. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel, and S. Tuecke, "Secure, Efficient Data Transport and Replica Management for High-Performance Data-Intensive Computing," presented at Mass Storage Conference, 2001.

[21] H. Sivakumar, R. L. Grossman, M. Mazzucco, Y. Pan, and Q. Zhang, "Simple Available Bandwidth Utilization Library for High-Speed Wide Area Networks," *Journal of Supercomputing*, vol. to appear, 2004.

[22] Tsunami Web page, www.anml.iu.edu.

[23] I. Foster and C. Kesselman, "The Globus Project: A Status Report," in *Proc. Heterogeneous Computing Workshop*: IEEE Press, 1998, pp. 4-18.

[24] I. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," *International Journal of Supercomputer Applications*, vol. 15, pp. 200-222, 2001.

[25] A. Roy, I. Foster, W. Gropp, N. Karonis, V. Sander, and B. Toonen, "MPICH-GQ: Quality-of-Service for Message Passing Programs," presented at IEEE/ACM SC'2000, 2000.

[26] ASCI/Alliance Center for Astrophysical Thermonuclear Flashes Web page, www.flash.uchicago.edu.

[27] W. Schroeder, K. Martin, and B. Lorensen, *The Visualization Toolkit: An Object-Oriented Approach to 3D Graphics*: Prentice Hall, 1995.

[28] S. St.Laurent, J. Johnston, and E. Dumbill, *Programming Web Services with XML-RPC*. Cambridge: O'Reilly and Associates, 2001.